Smart Devices and Software Agents: the Basics of Good Behaviour

Howard Chivers, John A.Clark, and Susan Stepney

Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK.
[chive,jac,susan]@cs.york.ac.uk

Abstract. In this paper, security requirements for software agents and smart devices are derived by working from typical requirements for existing systems, exploring the changes that are envisaged as systems become more highly distributed, then identifying what these imply for a device or service in a pervasive environment. A similar treatment is given to threats, which give rise to both security requirements and design issues. This approach provides insights into security requirements that will be significantly different from today's distributed system policies: they demonstrate that pervasive computing requires a qualitative change in security policy and practice. The paper also explores trade-offs between security complexity and device functionality, and argues that the degree of policy management required in a device will be an important factor in this balance.

1 Introduction

The increasing use of software based mobile devices is already evident in familiar objects such as car control systems, navigation aids, and mobile telephones. A good illustration of the development such devices into general-purpose application platforms is the evolution of mobile telephones.

We envisage a trend towards an environment where computer applications will be hosted on a wide range of platforms, including many that are small, mobile, and regarded today as candidates only for restricted functionality. This highly distributed computing world [1] is becoming known as *pervasive*. Large scale distributed processing is also a theme in traditional computing: the Grid community is enabling collaboration by distributed services [2], and this parallels the evolution of the Internet towards service delivery [3].

This world will be sufficiently different from today's distributed systems to demand a re-evaluation of the requirements and mechanisms for security. New issues include the policies required and how they are distributed, the effect of device context and the dynamics introduced by mobility, and the need to consider distributed security mechanisms and their effect on the system as a whole.

Distributed security implies that security requirements and mechanisms need to be considered in the design of smart devices and their applications, whatever their scale. This paper derives security requirements for such devices by evaluating the issues that

emerge in a pervasive environment, within a framework based on current system security requirements and practice (e.g. [4]).

There are system functions (e.g. billing, licensing,) and properties (e.g. safety) that are outside this scope. It is likely that solutions to these will also introduce new security requirements.

2. Security Requirements

Existing system security requirements are often grouped under the headings of Identification and Authentication, Policy, Administration, Accountability and Audit, Confidentiality, Integrity and Availability, and Assurance. This section takes each of these in turn, outlines typical current security requirements, discusses new concerns that arise in a pervasive environment, and then summarises the implications for hosting environments.

Terminology. The following terms are used: *subject* - an actual end user. *principal* - an authenticated subject, or software acting on that subject's behalf. *object* - a data item or software service. *device* – a physical device that provides a software environment (hosting environment) for a service. *service* – a software application, usually uniformly packaged for remote invocation. *agent* – a service that carries out a function on behalf of a subject; some authors define agents as mobile and/or autonomous, the more general sense is intended here. *device* or *resource manager* – the person capable of setting the security policy of a service.

2.1 Identification and Authentication

Systems are required to reliably identify a subject and also associate attributes (rights, roles, privileges) with that identification. Distributed systems support remote identification by providing an authentication mechanism between principals and remote objects (e.g. between a user client and a server); authentication can be in either direction or both, depending upon the application.

Issues. A fundamental change in the nature of identification is introduced by the need for agents to operate autonomously, invoking services on behalf of a user at times when the user is not present at the system. Consider a complex device (e.g. a telescope) controller – it would be able to invoke services for a user that would be unsafe to allow a user to invoke directly, and when the user is not necessarily present. Since many services will need to invoke further services, and may do so with authority that is distinct from their user's, there is an inevitable blurring of the once clear distinction between principals and objects in the system; the result is the need for the identification of devices in addition to users.

The limits of individual identity present a further problem. It is clear that if an agent is able to invoke remote services in the system, then it will often need unique identification. But what of smart devices that provide relatively restricted services

(e.g. a car engine controller)? The history of such a device will build a unique state (e.g. service record), and it may perform customized functions specific to individual users (e.g. driver preferences). The result is that despite the fact that the equivalent non-smart device may have been *fungible* (readily exchanged), the smart equivalent may acquire a unique identity.

The process of identification and authentication needs to deal with the dynamic and mobile nature of smart devices. This is illustrated by the current security issues with Wireless LANs or the protocols required to authenticate a mobile phone [5].

Attributes of Principals. At its simplest, identification is the binding of an identifier with an actual individual. Identification infrastructures also often support user attributes (e.g. actual name, organization, project roles) that can be referenced via the identifier. In the case of incompatible roles (e.g. bank official/customer) it is sometimes necessary to authenticate user-as-role, but it is generally preferable to provide user attributes separately to the authentication mechanism. In a distributed system with diverse applications, the range of user attributes that needs to be managed expands to meet application requirements (e.g. include credit card details). Thus we see a trend toward separate identity attribute management services, such as Liberty Alliance [6].

This design tactic of packaging a security mechanism as a service that can act on behalf of federations of users or resources is an important scalability enabler, and is likely to become a design theme in very large-scale systems; however, the functional convenience of packaging user attributes in this way needs to be weighed against the potential risk to individual privacy.

Finally, although the discussion of attributes has focused on users, there is a similar need to establish and promulgate attributes for devices (e.g. type, location, software environment), some of which may need to be dynamic.

Implications. Devices and their services need to meet the following concerns:

- Identity is required for devices as well as people.
- A rich and potentially dynamic set of identity-associated attributes needs to be supported and promulgated; privacy concerns will limit implementation options.
- Mutual authentication is needed between devices, and must be able to deal with dynamic location and communication.
- Designers must consider if smart devices are intended to be fungible, bearing in mind the overhead of maintaining individual, as opposed to collective, identity.

2.2 Policy

A policy is a set of security rules that specify what actions are allowed in a system, together with other ancillary information such as how, where and when to log events. The essence of most policies is to be able to identify the subjects and objects and to define the accesses allowed between them. A typical example is the well-known access mechanism in UNIX, which provides read/write/execute object attributes for owner/group/all subjects.

In current distributed systems the most common approach has been to extend identities across the network, either directly or by a mapping global into local identities [7], and then rely on the local policy.

Issues. Perhaps the most radical change is to question what such policies are designed to achieve. Protection of system resources has tended to be the key goal, with less emphasis on privacy, which has been treated as an aspect of confidentiality. However, users have an expectation that personal data will be used only for the purpose for which it was provided, and the fact that this is transparently not the case in today's networks is a growing cause of concern. The European Union Directive on data [8] also seeks to ensure uniform privacy standards in EU member states, including their external trading practices. Pervasive computing increases the threat to privacy by increasing the scope for location tracking, by holding a richer set of user attributes in the system, and by delegating rights based on user attributes to software agents.

The policies implemented in a pervasive system therefore need to take privacy into account to a far greater extent. This trend is illustrated in the Shibboleth project [9] where it is possible to authenticate with attributes other than identity (e.g. team membership) and only those attributes necessary to invoke services are propagated.

Authorisation. In existing systems the local security policy determines if a given subject is authorised to access services or data. A fully distributed world exposes the range of concerns embedded in such a policy. Consider a user accessing third party data, using a project budget to pay for computing services. The authorisation takes into account what services the project is authorised to access, and the policies of the data owner and the resource manager, as well as user attributes. In a distributed system these parties may be separate, and a hosting environment needs to dynamically compile the concerns of these stakeholders before authorising a transaction.

A number of researchers [10, 11] have proposed *trust management systems* that include chains of signed attributes as well as user delegations, with an authorisation function that is able to traverse and resolve the resulting threads of authority. Other researchers [12, 13] have proposed policies that explicitly deal with privacy. Trust management systems are capable of expressing flexible distributed policies; it remains to be seen if they are sufficiently dynamic for the pervasive environment, and if they are able to accommodate emerging privacy requirements.

Aggregation Constraints. Some policies can be expressed only as an aggregation (for example, project budget, permitted departmental storage). These resource constraints pose difficulties for distributed systems since they apply to collections of users and their applications that would otherwise be independent. To optimize utilization of such a resource, the policy implementation generally requires a mediating service, together with late or dynamic binding to agents requiring access to the resource.

Mobility. Mobility changes the physical and the software context of a device, with both direct and indirect effects on its security policy. The direct effect is the constraint placed on a device by its location. This may be physical (e.g. temperature),

availability of local services (e.g. logging), available communication methods, and, particularly when crossing national boundaries, legal, social or safety requirements. Indirect effects are that policy stakeholders (users, data owners etc) may not wish to apply the same policies in different locations; for example: it may not be lawful to export an algorithm; a data owner may not wish to place data in a competitor's location; smart cards may work only in the proximity of their owner. This places a requirement on devices to be able to infer their physical [14] and network contexts, and update their security policy accordingly.

Scalability. The type of policy and its distribution mechanism may present scalability problems in devices, and may shape the forms of policy that are possible, and their implementation. For example, the current Grid approach of substituting local for global identities has the problem that policies are of dimension subjects x resources: in principle each device needs to know every subject. Again, federation services are likely to provide the key (e.g. CAS [15]).

Implications. This discussion of policy illustrates the scope and complexity of the distributed security problem. However, devices may have limited computer power, intermittent communication capability, and restricted ability to sense their environment (e.g. smart cards are generally not powerful enough to execute a full range of cryptographic algorithms). Thus they may not be able to implement complex security policies. However, simply avoiding security concerns is not a solution: we have already noted user concerns about privacy, and security mechanisms are also needed to support integrity and availability.

One resolution is to balance limited security with limited functionality. For example, a smart car controller may have a simple security policy (e.g. odometer may not be written except by increment, only service centers can update service history), with the policy fixed for the life of the device. The penalty for such a fixed policy would be that it would not be possible to execute arbitrary software services (e.g. vehicle entertainment or navigation) if that later became desirable.

We therefore propose that device designers categorize smart devices from the perspective of how flexible they need to be in terms of policy management:

- *Fixed Policy*. A smart device with fixed security characteristics.
- *Updateable policy*. A device where a limited number of pre-set attributes can be updated (e.g. user of a smart lock).
- *Flexible policy*. A device that is capable of updating its policy from services in its environment. (e.g. to support dynamic response to changing device context).

Other device security requirements that emerge are:

- The device must be able to trace and resolve authorisations and attributes from several external stakeholders (e.g. subject, resource manager, subject's organization or virtual affiliation), relating to what actions are permitted.
- Privacy may require Authentication by means other than identity, and may limit the use to which a subject's attributes can be put.

- The authorisation process must support a dynamic policy environment, to accommodate both changing context and resource allocation policies.

2.3 Administration

Security administration involves the configuration management of security functions, policies and mechanisms. Administrators also manage intrusion detection mechanisms, and the technical and administrative responses to an attack.

Issues. Policy management requirements follow from the nature of the policy. The identification of stakeholders who are the source of permissions and constraints is the root of the policy configuration. In conventional systems the administrator maintains a table of users who act as owners for objects; the distributed equivalent is a list of stakeholder services from which policy constraints may be obtained. This trend is already illustrated by the storage of root Certificate Authority public key certificates in many systems. Similarly, instead of internally managing user attributes, the resource manager needs to maintain a list of trusted services that can authenticate a service request. So, management of fundamental security data is still required in pervasive systems, but the data are likely to be in a different form.

These requirements can again be interpreted in ways that are appropriate to the device or application. Consider a smart device that is able to associate itself with a user based on proximity. The underlying security requirement remains: the device is able to call on a known system service that authenticates that user. If the device has a flexible policy, the manager must be able to specify how that service is to be obtained.

Intrusion Detection. One of the distinguishing features of open distributed systems is that they are constantly under attack, and constantly penetrated by attackers; the challenge is to use their size and diversity to self-heal. To a limited extent this is achieved today – the size of the Internet makes the detection of attack mechanisms likely before they have been widely exploited, and there are commercial (e.g. suppliers of anti-virus software), educational and governmental organizations[1] who are prepared to fund the ongoing research and promulgate results. The lesson for distributed systems is to design them in such a way that large or systematic attacks are likely to cause observable behavior, which can then be investigated. Conventional security wisdom can help in the design of smart devices with such properties: devices should be configured to regard confusion (e.g. invalid service requests, failed authentication) as a potential attack and support some form of external signaling mechanism (see 2.4).

Containment. An important containment strategy is *least privilege*: operate every service with only the privilege necessary to carry out the required function. This is not just good security practice, but generally good design practice, since it tends to trap the propagation of errors. Other forms of containment may also be possible in highly

[1] For example, The CERT Co-ordination center. http://www.cert.org/

distributed systems, for example, the duplication of services with voting on the result, or detection of inconsistencies within a session.

Intrusion Response. In a distributed system, problems are likely to originate from outside of the boundary of any given resource manager, but the actions required (such as establishing interim service, tracking and isolating the cause, damage assessment and repair) all require organizational co-ordination and an accurate and shared understanding of system dynamics. Needing agreements between all pairs of management organizations to determine responsibility and action for problem tracking does not scale, so this is another case where federations are necessary.

Implications. Security administration places many requirements at the organization and system level, and few on individual devices. However, there are some:

- The need to support policy management by maintaining root locations of services from which policies can be obtained.
- The need for mechanisms to detect unexpected or inconsistent invocation, which require a supporting service of accountability and audit (2.4).

2.4 Accountability and Audit

The usual requirements for accountability are to associate a subject with a security event, to be able to select which events are recorded and to ensure that records are preserved and retrievable for analysis. Current practice in distributed systems leaves security logging to the local system, but ensures that the identification can be traced to an original subject. There are emerging distributed notification and logging application frameworks[2], but we are not aware of their use for security.

Issues. Privacy requirements (see 2.2) create an immediate problem for accountability: if a user is identified only as the member of a group, who is to be held accountable for any subsequent misuse of the system? This issue requires a new approach to accountability; there are a number of possible solutions, including the use of session pseudonyms that could be traced to a user by their local administration, but even these may risk aggregation attacks on privacy. It is clear that users' policy requirements and accountability are no longer independent issues.

The preservation and retrieval of a security log presents practical problems in a fully distributed system. The straightforward answer of requiring a distributed network logging service resolves the problem for more complex devices (subject to a design which is robust and scalable) but not all devices will be able to persist data, and communication may be intermittent. This is another instance where it is necessary to find a mechanism appropriate to the scale of the device. For simple devices it may be sufficient to raise an alarm when an attempt at misuse is detected, by entering an externally detectable 'safe state'. If the device can be reset only by its resource

[2] for example, Apache log4j

manager, who is able to retrieve any stored information about the incident, then the requirement will have been met.

Implications. Accountability and logging cannot be avoided, since the overall system needs to detect and respond to misuse. Where possible an application should use a distributed logging service to record events; where that is not feasible this requirement may be met in simpler ways that still allow the detection of attacks.

2.5 Confidentiality

Confidentiality in present systems is supported by associating owners with objects, and allowing owners to place access constraints on those objects. Where requirements for confidentiality or privacy are particularly important (e.g. credit cards) then access constraints inside systems are supported by communication confidentiality services. Communication Confidentiality is also required for some system functions, for example the distribution of cryptographic keys.

Issues. There is no reason to suppose that access control of objects within a device will be less important that it is at present: it is a fundamental mechanism supporting the practice of least privilege in a system, including the separation of system and application software. Least privilege mechanisms are also needed to support user privacy; however, the discussion above (2.2) indicates that privacy requirements are more extensive than confidentiality, and justify separate consideration.

Again, there may be a problem with devices without the capability to support the equivalent of an operating system kernel. One possible solution is for such devices to be stateless, maintaining state for the delivery of a service to a single invoker, and then disposing the state before a change of use (for example, see [16]).

Confidentiality has been a relatively unimportant aspect of communications, except for special cases: for example, defence applications, or the protection of credit card and similar information. This is already changing as communications become more open; the evolution of mobile telephone protocols illustrates the process and the extent that confidentiality can become a user concern. A similar process is taking place with wireless LANs; users are recognizing that broadcasting business communications may not be desirable, and even if they are not concerned about their data the use of encryption denies an attacker an easy option for mapping a system prior to mounting an active attack. Information about location, or which devices are communicating, may also need to be contained for privacy reasons. A greater need for communication confidentiality is therefore consistent with the vulnerability of increasingly open communication methods and also with privacy requirements.

Some researchers [17] suggest that software agents will be mobile to the extent that they will move between host environments while they are executing. Even assuming that the security policy in the new host makes this feasible, static access restrictions on internal objects would have to be preserved in transit, and this may also require an end-to-end confidentiality service.

Multi-way security context. Consider a tree of service invocations, the leaves of which need to communicate in order to carry out their function. This is a common model for computing tasks, with a scheduler at the root of the tree, but is equally applicable to delivery chains such as video streaming. The agents in the delivery chain communicate with their logical neighbors, perhaps via intermediate physical devices that are not part of the process. They need to authenticate each other as valid parts of the same service, and ensure end-to-end process integrity and perhaps confidentiality. System security mechanisms are needed to ensure that the security context of a complete process is shared by the services within the process, and is not otherwise accessible.

Implications. Devices need to support internal access control, if necessary in the simple form of a stateless service.

Confidentiality becomes a more prevalent communications requirement, because more accessible methods of communications are likely to be used. A device also needs the ability to build communication security services to collaborating devices via intermediaries that are not part of the same security context.

2.6 Integrity and Availability

The primary focus of integrity is maintaining service and defending against accidental or deliberate acts that can degrade or deny service to users. Quality of service is usually defined and measured in terms of non-functional attributes of services that users can perceive, such as latency, volume of storage or throughput.

Issues. The ultimate contract for quality of service is likely to remain with the user or organization prepared to pay for a service, but distribution complicates the picture: it becomes difficult to establish how each individual device contributes to overall quality, and particularly difficult to instrument marginal failure. The problem is illustrated by the need to provide sophisticated logging services in current distributed applications [18]; these difficulties are compounded in a more dynamic environment.

Conflicting user policies are possible, for example, one user's advertising may be another's junk mail. As at present, when such conflicts become offensive to one user, or consume appreciable resource, action is necessary at the system level to restore a balance. How such control is to be exercised in a pervasive system is yet to be established, and highlights the tension between agent autonomy and system stability.

The perception of availability by a user depends on the application. At present, denial of service attacks seek to disable a server by the exhaustion of one or more of its resources. As distributed systems become used for both critical and commonplace functions, then less dramatic modulations of service may become important to users: consider the opportunity cost to a stockbroker of delayed market data.

At the same time increasing numbers of devices increase the access opportunities for attackers (see 3, below), and the nature of some devices is such that failure may be permanent, either for physical reasons (the device itself, or inability to access it), or because the only safe response to an attack is to disable itself.

Implications. The need to defend against attacks on service integrity will grow rather than reduce; devices should be designed to resist resource exhaustion attacks, for example, by rationing as resources approach saturation. Other forms of defense need to be applied at the system level, including intrusion detection and response.

2.7 Assurance

Assurance that a device is able to support the security policies it claims is based on the isolation and identification of its security mechanisms and the lifetime protection of those mechanisms through measures such as change control, configuration control, and operating procedure management.

Issues. Almost none of the current assurance practices can be applied to a complete distributed system. If system level assurance is required, new assurance mechanisms have to be developed based on reasoning about system properties, while assuring the properties of devices and services individually.

One aspect of assurance that becomes important in pervasive systems is establishing that a device is what it claims to be: at one extreme that it has not been impersonated, at the other that it will enforce any claimed restrictions on its own behaviour. The former may be a question of identity, the latter a question of assurance, which is technically difficult to achieve over the lifetime of a mobile smart device. Furthermore, the question of demonstrating that a host is actually running a given piece of software has, at present, no solution (see summaries at [19, 20]).

Implications. Some assurance in the implementation of security mechanisms in smart devices is necessary; the design principle of separating security and functionality is applicable to all devices with security mechanisms.

Mechanisms need to be developed to provide dynamic assurance of both software environments and mobile code.

3. Threats

The following discussion highlights those threats that are either new, or more prominent, in pervasive systems.

3.1 System Composition

The potential for vulnerabilities due to system composition is, in large part, driven by the specific policy management and authorisation system, and how they resolve inconsistencies that result from different users' perspectives. Some other issues that arise in the composition of the system include:

Critical failure points. Single points of failure can be categorized as physical or logical. Even network topologies with good scaling or distance properties may be disproportionately vulnerable to the physical failure of specific nodes or links, and networks with 'border' devices that concentrate traffic are obvious physical targets.

Logical failure can occur at any point in the design and implementation process, but the resulting vulnerability may become widely dispersed in the system. Design flaws include faults in protocols or encryption algorithms. Implementation flaws include physical failures, such as vulnerabilities in RF receivers, and software defects such as implementation errors in protocol libraries.

Aggregation. Although the need for user privacy can been expressed in a policy, such policies may be vulnerable to being bypassed by data aggregation. This is an extension of the problem of propagating trust via delegation: how does a user meaningfully restrict the use of data in services that are invoked only indirectly?

Pervasive network attacks. Network Worms and Viruses are well-established threats, and will remain so. With very large networks, however, it may be possible to construct self-perpetuating dynamic attacks that operate at the network rather than the component level. A familiar example is a cycle of emails triggered by inadvertent circular re-direction; other mechanisms of devising cycles or waves in a highly functional network are likely to emerge, with the possibility of large-scale persistent denial of service.

Implications. The designer of a device or service must avoid design features that could create a local vulnerability or point of attack. It is also be necessary to show that device policies do not result in undesirable emergent system behavior [21] that could be exploited by an attacker, or invoked by accident.

3.2 The Local Context

The local context of a device is its physical environment, the communications and sensors it may access, and neighboring devices. The general threat is that an attacker may subvert the local context to mislead a device about some aspect of its environment, either causing damage or mis-invoking functionality.

Physical. If the device has sensors then it may be possible to manipulate them directly, or use extreme temperatures or other environmental factors to modify device performance.

Many forms of communication are subject to physical attack, and the ease with which this can be carried out determines the true threat. Active remote attacks against wireless networks are now commonplace, and there is a danger that 'WarDriving' could be extended to using electromagnetic pulse weapons; even simple weapons could damage RF circuitry. Passive attacks on communications could be an unacceptable risk to confidentiality and privacy, and could enable system mapping prior to an active attack.

Software. There is a risk that subverted or spoof devices could be introduced into the network (e.g. an IP masquerading attack or Trojan Horse service) and manipulate the system to violate its security policies (e.g. migrate data in violation of confidentiality). An important case is where a subverted host seeks to migrate an attack through the system. Historically, most methods by which a remote device can be induced into executing software have been used to propagate Trojan Horse attacks (e.g. opening emails with active content, buffer overflow attacks), and so any feature that executes remote code is a potential vulnerability. Protecting a host against an agent, and an agent against a host, are still unsolved problems [18,19].

Denial of Service attacks tend to use expected service requests in unexpected ways or numbers, eventually modifying a device's capability by exhausting or saturating a physical resource (cpu cycles, memory, file pointers…). There is little new in principle about this form of attack, but novel types of device introduce new services and resource constraints that may then be exploited. For example, communications that use calling channels or ports can be saturated before their bandwidth is exhausted.

A common limitation in smart devices is power (either instantaneous power or total energy) and this needs to be treated similarly to avoid denial of service based on power exhaustion (see 'Sleep Deprivation Torture', [15]).

Implications. The current experience with wireless LANs is that pervasive functionality may provide points of access that can also be exploited by an attacker. Such access points include 'anonymous' logins, accounts without passwords and developer and system administration back-door services, as well as broadcast information about services. Capabilities of this sort need to be subjected to careful security risk assessment.

Designers should be aware that the first stage of an attack might be to *map* the system. Mechanisms that make mapping hard provide useful defense in depth, particularly if an attacker risks detection during the mapping phase. In the case of broadcast communications, encryption may be valuable for just that purpose, even if it is too weak to provide long-term confidentiality.

Authorisation is needed between co-operating devices to avoid spoof device attacks. It is less clear how to defend against remote subverted software; such attacks tend to be identified by their actions or by other fingerprints (e.g. signature control ports). Since migration of code may become one of the enablers of pervasive computing, it deserves special attention: devices that feature the capability to run such a service must defend against the importing Trojan Horse software, for example, by ensuring that it executes in an encapsulated software environment.

Devices should be design to minimize hard failure on resource exhaustion including device power, for example, by resource rationing strategies.

Redundancy may increase the difficulty of environmental manipulation as an effective attack on the system, even if such manipulation is effective against individual devices.

3.3 Devices

Devices, which may be small and mobile, are vulnerable to a variety of attacks resulting from their physical nature. Modern smart cards are a good example: it is possible to obtain them legitimately and then attack them at leisure, and even to assemble large collections for study. Physical theft or substitution is also not difficult. Their relative size and portability limits the designer's capability to protect the device, so they may have limited tamper proofing and be vulnerable to a wide range of environmental attacks.

In practice it may be difficult to dispose of device state: although the device may faithfully prevent access to old state, and perhaps even 'zero' memory or magnetic media, there are physical remenance mechanisms in most forms of storage which could leave the device vulnerable to a determined attacker with physical access.

Implications. The physical design of a device is a security mechanism, designers must minimize the possibility of physical remenance and device substitution.

3.4 The User's Perceptions of security

From the security point of view, users are naïve, so security mechanisms that are designed to involve the user's judgment are vulnerable. Current examples include manual agreement to PKI certificates and the management of private keys. Users are also conditioned by a set of assumptions about security based on past experience: the use of mobile telephones is a good example of a presumption of privacy that was not originally supported by the technology, and is still not supported by the environment where users choose to make personal or business calls. The principal threats are:

- *Social engineering*. Persuading a user to do something incorrect, such as resetting a password or simply misusing the system [22].
- *Confusion.* A user acting incorrectly because of lack of understanding (accepting a new PKI certificate from the wrong source), or carrying our a normal action in the wrong environment (typing a password into a spoof screen).

Implications. Device designers should be aware of users preconceptions and work within them. For example – given a pen and diary, either could become smart and store the data, but users are accustomed to the fungibility of pens and the need to protect and store diaries.

On a positive note, attackers may also be confused about the overall properties of a very large system and so the successful implementation of distributed intrusion detection may well allow early detection of potential attacks, in a similar way that many virus types are reported before they are found 'in the wild' (for example, see CERT co-ordination center statistics).

4. Conclusions

This discussion of security, drawing from the perspective of existing system requirements, rather than from the device technology, effectively illustrates the range of security requirements that are required for host environments in pervasive systems. The question of security is unavoidable; it is not possible to build a reliable and effective computing infrastructure if the devices supporting the system do not include security mechanisms.

The practical problems of size and low computing power of many devices can be mitigated by a careful consideration of the balance between security complexity and required functionality, and we offer some suggestions along these lines. In particular, we classify the degree of policy management required in a device into three types: *fixed policy*, *updateable policy* (perhaps just one policy attribute, such as user) and *flexible policy* (able to update its policy from services in the environment). We believe that this is a useful indicator of security complexity and will be of value to designers seeking to achieve this balance.

Although the purpose of this paper is to elicit requirements for devices, we note that there are a number of open questions at the system level; they include:

- How should security policies be expressed and implemented to combine the interests of multiple stakeholders, implement privacy, and express aggregated constraints in a dynamic environment?
- How should users, devices, and resources be federated to facilitate scalability, and what services are required to support these arrangements?
- How can a balance be achieved between the needs of privacy and accountability?
- How can security assurance be provided for large distributed systems?

By starting our analysis from a classical security perspective, rather than a device perspective we provide an insight into security requirements for devices and software agents. The result, however, is rather more than a simple extension of today's distributed system policies; the security issues in a pervasive computing world are qualitatively as well as quantitatively different, to the extent that the success of pervasive computing requires the re-invention of system security practice.

References

1. Weiser, M., *The Computer for the 21st Century*. Scientific American, 1991. **265**(3).
2. Foster, I., et al., *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Implementation*. 2002, Global Grid Forum, http://www.gridforum.org/ogsi-wg/
3. Kreger, H., *Web Services Conceptual Architecture*. 2001, IBM, http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA
4. *Trusted Computer System Evaluation Criteria (Orange Book)*. 1985, US Department of Defence, DoD 5200.28-STD

5. *Network Related Security Features*. 2001, 3rd Generation Partnership Project, GSM 03:20
6. *Liberty Architecture Overview*. 2002, Liberty Alliance, http://www.projectliberty.org/
7. Foster, I., et al., *A Security Architecture for Computational Grids*, in *Proc 5th ACM Conference on Computer and Communications Security*. 1998. p. 83-92.
8. *Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data*, in *The European Union*. 1995.
9. Erdos, M. and S. Cantor, *Shibboleth Architecture*. 2001, Internet2, http://middleware.internet2.edu/shibboleth/
10. Thompson, M., et al., *Certificate-Based Access Control for Widely Distributed Resources*, in *Proc 8th Usenix Security Symposium*. 1999: Washington, D.C.
11. Li, N., J.C. Mitchell, and W.H. Winsborough, *Design of a Role-Based Trust-Management Framework*, in *2002 IEEE Symposium on Security and Privacy*. 2002, IEEE Computer Society: Berkeley, California. p. 114-130.
12. Fischer-Hübner, S. and A. Ott, *From a Formal Privacy Model to its Implementation*, in *Proceedings of the 21st National Information Systems Security Conference (NISSC '98)*. 1998: Arlington, VA.
13. Jiang, X. and J.A. Landay, *Modeling Privacy Control in Context-Aware Systems.* IEEE Pervasive Computing, 2002. **1**(3): p. 59-63.
14. Gellersen, H.-W., A. Schmidt, and M. Beigl, *Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts.* Journal of Mobile Networks and Applications, 2002(Special Issue on Mobility of Systems, Users, Data and Computing in Mobile Networks and Applications (MONET)).
15. Pearlman, L., et al., *A Community Authorisation Service for Group Collaboration*, in *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*. 2002, IEEE.
16. Stajano, F. and R. Anderson, *The Resurrecting Duckling: Security Issues in Ad-Hoc Wireless Networks*. 1999, AT&T Laborotories,
17. Nwana, H.S., *Software Agents: An Overview.* Knowledge Engineering Review, 1996. **11**(3): p. 205--244.
18. Gunter, D., et al., *Dynamic Monitoring of High-Performance Distributed Applications*, in *Proceedings of The 11th IEEE International Symposium on High Performance Distributed Computing*. 2002.
19. Farmer, W.M., J.D. Guttman, and V. Swarup, *Security for Mobile Agents: Issues and Requirements*, in *Proc. 19th NIST/NCSC National Information Systems Security Conference*. 1996. p. 591--597.
20. Karjoth, G. and J. Posegga, *Mobile Agents and Telcos' Nightmares.* Annales des Telecommunications, 2000. **55**(7/8): p. 29--41.
21. Stepney, S., *Critical Critical Systems*, in *British Computer Society, Proceedings of Formal Aspects of Security FASEC '02*. 2002.
22. Cybenko, G., A. Giani, and P. Thompson, *Cognitive Hacking: A Battle for the Mind.* IEEE Computer, 2002. **35**(8): p. 50-56.