Enforcing policies in Pervasive Environments *

Anand Patwardhan, Vlad Korolev, Lalana Kagal and Anupam Joshi
Computer Science and Electrical Engineering Department
University of Maryland at Baltimore County
1000 Hilltop Circle, Baltimore, MD 21227
{anand2, vkorol1, lkagal1, joshi}@cs.umbc.edu

Abstract

This paper presents an architecture and a proof of concept implementation of a security infrastructure for mobile devices in an infrastructure based pervasive environment. The security infrastructure primarily consists of two parts, the policy engine and the policy enforcement mechanism. Each mobile device within a pervasive environment is equipped with its own policy enforcement mechanism and is responsible for protecting its resources. A mobile device consults the nearest policy server, notifies its current state including its present user, network presence, other accessible devices and location information if available. Using this information the policy server queries the "Rei" engine to dynamically create a policy certificate and issues it to the requesting device. The system wide policy is described in a semantic language "Rei", a lightweight and extensible language which is able to express comprehensive policies using domain specific information. The "Rei"policy engine is able to dynamically decide what rights, prohibitions, obligations, dispensations an actor has on the domain actions. A policy certificate is created and issued to the device. The policy certificate contains a set of granted permissions and a validity period and scope within which the permissions are valid. The policy certificate can be revoked by the policy enforcer based on expiration of the validity period or a combination of timeout, loss of contact with an assigned network.

X.509 based Public Key Infrastructure is used to provide identification and authentication.

* This research was supported in part by NSF awards 9875433 and 0242403, and a grant from NIST.

1. Introduction

1.1. Device Characteristics

Mobile devices including cell-phones, PDAs, laptops have gained widespread acceptance. These devices are compact, lightweight and typically have limited storage, computation and communication capabilities. They are capable of some form of wireless communication viz. Bluetooth, 802.11, CDPD or infrared. Also such devices are usually associated with a single user and carry personal data as well as corporate data.

Although undisrupted and continuous connectivity for wireless devices is hard or perhaps impossible to guarantee, connectivity of these devices has continued to improve and most devices nowadays are *almost* continuously connected to the internet via some form of wireless access. This improved connectivity enhances user experience but at the same time poses new security concerns. Devices can have several interfaces viz. IrDA, Bluetooth, USB, Serial port, 802.11 etc. Each unguarded interface exposes a potential source of attack.

The ported or simplified versions of software deployed on these devices are often more vulnerable than on the systems for which they were originally intended. Power-aware applications reduce or leave out in-built security mechanisms. Security issues involving wire-line devices (desktops and servers) that have conventional secure physical locations and are subject to some domain specific administrative policy have several well understood and worked out solutions. Unlike wire-line devices however, the untethered and compact mobile devices are more susceptible to be lost, misplaced or stolen. Moreover, their presence is not restricted to a particular network. Mobile users are expected to traverse a large number of wireless networks. In the absence of a security infrastructure wireless communications can be overheard and the devices themselves are susceptible to attacks.

1.2. Device environment and usage patterns

With the proliferation of handheld devices and growth of wireless hotspots which provide multitudes of services for such devices, in places like cafes, gas stations, malls, airports; privacy of data exchanged in such interactions, needs to be addressed. Some capabilities of the devices may have to be disabled or restricted in order to protect the device from others around it. In such pervasive environments, durations of communication are variable, and the characteristics of data exchange amongst entities is very different from wired devices [4], [6]. Also in pervasive environments the devices and/or the subjects that will interact are largely dynamic and cannot be enumerated a priori. e.g. two individuals wishing to exchange electronic business cards via their PDAs in a cafeteria, might never have met before.

Government agencies and companies often lease out such devices to their employees. These devices are carried around by their owners/lessees outside physically secured premises. Portable devices can be lost, stolen or go missing. These devices can go missing or unaccounted and may have carried sensitive data and capabilities at the time they were lost or stolen. FBI data [1] shows that less than 3 percent of stolen or lost laptops are ever recovered. This poses a serious security threat especially if the device is carrying sensitive data or has other special capabilities which if not revoked can prove to be extremely damaging to the organization.

We propose a security infrastructure that uses a semantic language to express security policies and use policy enforcement mechanisms on the mobile devices in an effort to mitigate the threats posed to the device. We describe a proof of concept implementation where we use the semantic policy language "Rei" [15] and a policy engine which we use to dynamically create policy certificates for mobile devices based on their context information.

For example, we want the device capabilities like 802.11, Bluetooth and IrDA to be unrestricted while it is in a trusted network. However if the device is in a untrusted/foreign network the enforcement mechanisms on the device disable PIM, Bluetooth and IrDA communications, and limit web access via 802.11 interfaces to certain sites/networks.

Note that the security kernel on the mobile device itself is trusted to actually enforce the issued policies. The policy server on the local wireless network will first verify the authenticity of the device before granting network resources to the device. Thus, any untrusted device will be denied network resources.

The paper is organized as follows. Section 3 describes the design objectives, design considerations and device capabilities. Each component of the policy enforcement infrastructure is described in sections 4 to 9. Finally two example scenarios are presented in section 10.

2. Related Work

2.1. Device Security

Susilo [16] identifies risks and threats pertaining to use of handheld devices connected to the internet. The first kind of threat is the use of the mobile device as a carrier of malicious code. Since the mobile device is not subject to physical security as are the fixed computers in the wired networks, it is susceptible to attacks and hence can potentially host malicious code while it is in some untrusted network and try to propagate it, once back in the home network. Another scenario involves a temporary user granted access to the network injecting malicious code into the network. Susilo [16] suggests a combination of a personal firewall on the mobile device and another firewall at the access point to protect against this scenario.

Another threat identified is one arising from applications that are capable of running on multiple platforms viz. .Net applications. Multi-platform malicious code can be difficult to detect and potentially affect all devices supporting the framework.

Susilo [16] underlines the importance of having some kind of protection mechanism on the handheld device itself.

2.2. Lightweight secure communication

Jenkin and Dymond [13] propose the use of one time pads for ensuring data privacy. To reduce the computational burden of encryption and decryption in PKI or similar infrastructures, use of one time pads is suggested. The one time pad is stored on the on-board storage of the device or some (secure) removable media. The encryption process involves XORing the bits of the data stream with those of the one time pad. The original data can be recovered by XORing the data stream with the same bits used in the encryption process. This relieves the computational burden for the handheld device, but requires large storage requirements. Also distribution, secure storage and storage capacity make it infeasible for most ordinary situations. Also this kind of communication is limited to parties which share the one time pads, making pad distribution and secure storage a matter of concern.

This kind of privacy mechanism will be appropriate for only a fractional part of the total data traffic, perhaps the most sensitive parts of it. In our approach we use the PKI infrastructure. SSH sessions are used in issuing policies. Digital signatures are used to verify the policy issuer's identity and non-repudiation. The iPaq 3800 series that we are using is capable of handling these computations. Most of the data traffic is not encrypted. In fact even policies themselves need not be encrypted, instead digital signatures on policy digests are provided for verification. These checks are enforced only when new policies are received and policy updates are not frequent.

2.3. Policy Language

We use the Rei policy language for expressing security policies. Rei is a policy specification language defined in OWL-Lite.

Extensible Access Control Markup Language (XACML) [8] is a language in XML for expressing access policies. XACML allows control over actions and supports resolution of conflicts. However, as it is based in XML, it does not benefit from the interoperability and extensibility provided by semantic web languages. It also does not model speech acts or handle conflict resolution across policies. KAoS is a policy language based in OWL and is similar to Rei as it can be used to develop positive and negative authorization and obligation policies over actions. KAoS policies are descriptions of actions that are permitted (or not) or obligated (or not) limiting its expressivity as policies are restricted to OWL. However, KAoS allows the classification of policy statements enabling conflicts to be discovered from the rules themselves. The Rei engine includes run-time conflict resolution but cannot predetermine conflicts. Another advantage of KAoS is that, if policy descriptions stay within OWL-Lite or OWL-DL, then the computation is decidable and has well understood complexity results. In terms of speech acts, however, KAoS only models delegations, whereas, Rei includes an integrated approach to speech acts for policy management, which is useful in autonomic, distributed systems. Rei also provides specifications and tools for policy analysis and consistency checking that KAoS does not.

SWRL is a rule language for describing Horn like rules in OWL [9]. Currently SWRL is still work in progress and there are very few tools for it like Hoolet [5] that are just being developed. We are considering re-doing Rei in SWRL in a few months when SWRL and some supporting tools are more mature. This will probably be an improvement and will strengthen Rei because we will not be introducing our own way of encoding rules.

2.4. Policy Enforcement on handheld devices

Jansen et al. [11], [10] describe an implementation of assigning and enforcing policies on handheld devices using Java smartcards. The organization policy is distributed via tamper-proof smartcards. All the devices are smartcard enabled. The policy to be enforced is read from the smartcard, which requires authentication by a username and pin. After authentication a card monitor continuously monitors the existence of the smartcard. If the smartcard is removed the device reverts to the default policy of the device. [10], [12] describe the use of a policy specification language, a policy distribution mechanism and certificate representation.

3. Design Approach

3.1. Design objectives

3.1.1. Access privileges subject to constraints In an organizational setting, known users of the system are given certain rights and privileges. Some of these users may have the right to delegate some rights to others on a temporary basis. These temporary access privileges granted to other users are usually short-lived, yet all possible delegatees themselves cannot be enumerated beforehand. The system administrator may however wish to restrict the actions that the delegatees may take, using the resources that they have been granted temporary access to. Furthermore, the system administrator may want to ensure that such delegations can be made only within the organization's wireless network coverage. The policy specification should be able to express actions and the constraints which should be satisfied for granting those actions.

3.1.2. Network specific policies In certain situations it might also be required of a trusted device to enforce a policy dictated by the local network it is currently attached to. In this case the network policy is used to decide what the device is allowed to do. Thus distributed policy management is required. This is possible with the use of the semantic language "Rei".

3.1.3. Accountability Additionally, the system administrator may need to enforce strict policies about how users can utilize resources in the system. Organizations lease out such devices to their employees and the employees are accountable for the proper use of the device. Lost or misplaced devices have to be accounted for and possibly traced. Other than the financial loss, sensitive data may be compromised along

with the loss of the device. Thus the system administrator needs to be able to specify the policy which will ensure the safe use of the device and limit damage in case it is lost or stolen.

3.1.4. Automatic guards Though most wireless devices come equipped with some kind of wireless interface and wireless access is commonplace, it is still infeasible or perhaps impossible to be able to consult a trusted policy issuing authority "anytime and anywhere". The intention of having a policy enforcer is not merely to restrict the capabilities of the device, but rather to serve as a guard against misuse, unintentional or erroneous violation of the organization policy. This is a security feature added to the device. e.g. the user need not reconfigure the wireless interfaces of the device every time he/she walks out of the secure network. Such reconfigurations should happen automatically based on the state of the device.

3.1.5. Capability Restriction System administrators may want to be able to disable some functionality of the device if it is not operating within specified wireless network premises. The organization security policy may restrict carrying sensitive data outside the premises, in which case the policy enforcer located on the mobile device can purge sensitive data or encrypt it if the device detects that it is no longer within the organization premises. The user need not have to be bothered with these details. The security policy is enforced based on defaults provided to it. The user is also prevented from inadvertently breaching the policy.

3.1.6. Adaptive Behavior Mobile devices can have numerous "wireless encounters" with other unknown and/or anonymous devices. While expressing security policies for such mobile devices all such encounters cannot possibly be enumerated beforehand and neither can the device store a large number of such entries nor depend on some augmenting device to provide such an enumeration. Conventional security practices for wireline networks are not directly applicable here [6]. The behavior of a mobile device should ideally adapt to its environment. e.g. the organization policy may allow the device to perform certain actions within a trusted environment, but not in an unknown environment. The device could rely on a trusted authority to provide it with access constraints. However consulting a trusted server for every access is also not always possible, since undisrupted connectivity to such a server is hard to guarantee.

3.2. Design considerations

It is widely accepted that the most reliable security mechanisms need hardware support and, most devices nowadays come equipped with some basic hardware protection. Tamper-resistant smart cards are widely used for secure storage of passwords, credit card numbers, private keys and other sensitive data. More sophisticated devices like the HP iPaq 5450 [3] come equipped with a biometric fingerprint reader. IBM and Microsoft are in a strategic alliance with IBM providing an "Embedded security subsystem" [2] supported by Microsoft's operating system. The inbuilt security chip provides hardware based protection. It is capable of storing electronic credentials mentioned earlier. Most device manufacturers support or plan to support secure storage for such credentials.

PKI or PKI based infrastructures are thus easily deployable on these devices. The devices that we used in our testbed comprised of several iPaq 3800 series Pocket PCs which have sufficient computation power and 128-bit key encryption and decryption is not a burden.

3.3. Device Capabilities

The device that we primarily used is an iPaq 3870. A wireless card sleeve was used to provide wireless access to the device. Inbuilt capabilities of the device include IrDA and Bluetooth. With this configuration, the device is capable of accessing the internet via a 802.11b wireless network. Other than web access, the IrDA port and Bluetooth capability can be used to transfer Personal Information Management(PIM) data. The cradle of the device also allows communication via the serial and USB ports. The Familiar Linux kernel(ver. $2.4.18 - rmk3$) we are using, currently does not support communication via the USB port.

We want to show how these device capabilities can be selectively restricted to enforce a specified policy. Based on the network status and the device status the device posture could be one of the following. The device and network state is modeled in the Rei language policy.

Device Posture

- Basic (Level 0)
- Normal (Level 1)
- Alert (Level 3)
- Shutdown (Level 4)

Device Environment

- Home Network
- Benign Network
- Hostile Network

- Dead-zone

Based on the posture of the device and the sensed environment of the device, the appropriate policy should be applied. The device can sense the environment by listening for heartbeats from trusted web entities called beacons. When it hears heartbeats from the "home" beacon, the device can safely deduce that it is still within the home network and the "home" policy is applied. Suppose the device is in another known network of some affiliated organization, the device will recognize the heartbeats of the trusted foreign beacon. The foreign beacon can specify a different policy server for that network, and the device then complies to the policy specified by the new policy server. In the absence of heartbeats from trusted beacons, the network is presumed to be a hostile environment. In the event that there is no network access the device assumes that it is in a dead-zone. This state is largely based on the status of the 802.11 based network access. In our current implementation, network state is determined based only on broadcast messages sent out by the beacons on the 802.11b network.

3.4. Architecture

As shown in figure 1, there are essentially three components on the client side(mobile device) viz. the "Policy Manager", "Context Manager" and the "Policy Enforcer". The "Beacon" is located on a local network device possibly co-located with the wireless base station. The server hosts the "Policy Server" and the "Policy Engine". The beacon periodically broadcasts heartbeats that the sentient Context Manager on the device continuously monitors.The role of the Policy enforcer is to enforce the currently selected policy, whereas the Context Manager is responsible for monitoring the context of the device and selecting the appropriate current policy. The device boots up with an initial default policy. The context manager listens for updates from the policy server. The device can be transit between the home network and other known or unknown networks. Beacons are deployed across the wireless networks, which periodically broadcasts heartbeats. The context manager listens for these heartbeats and based on the information contained in the heartbeat, can determine if it is within a trusted network. This state is continuously monitored. In the event that heartbeats are not heard for a prolonged interval of time, the context manager assumes that it is no longer within the trusted network and immediately reverts to the policy prescribed for untrusted networks.

Pietro and Mancini [7] point out that it is important to restrict the web presence of a service to reduce the complexity and traffic for a given network infrastructure. In our design the issued policies are valid only

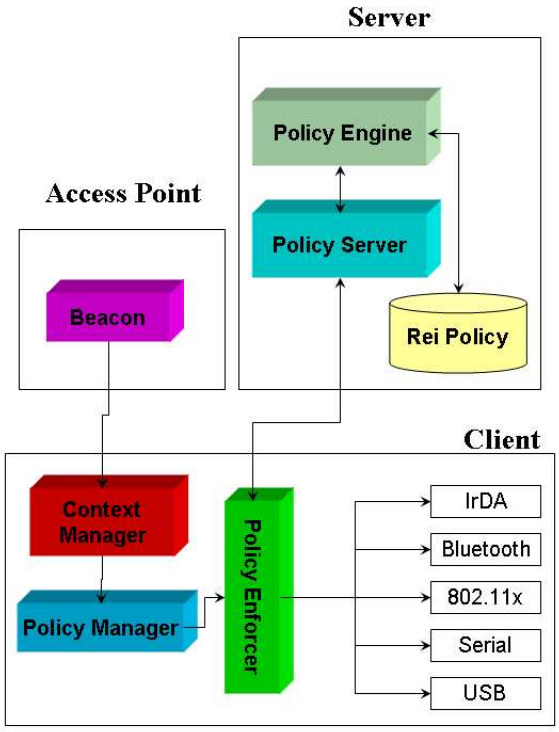

Figure 1. Policy Enforcement Infrastructure

within the scope of the broadcast, that is the hop-count of the broadcast determines how far the heartbeats will be heard. As soon as the user is outside this scope, the policy is no longer valid and the device reverts to the highly restrictive default policy. Thus this mechanism has two effects, viz. it restricts the scope of operation to a particular area, use of granted privileges is disallowed outside this scope and, secondly a context is provided to the device so that only the relevant service interfaces may be exposed in communicating with the handheld device. A device possessing some capabilities allowed by the enforced policy will allow the device to access local services, whereas remote services can always be exposed via proxies if necessary.

Heartbeats are signed by the owner entities and can be verified by other entities involved. Using a PKI infrastructure with X.509 certificates is feasible in this scenario. Trust issues are resolved using CA certificates installed in the mobile device.

Each of the modules shown in figure 1 are described in the following sections. Section 4 describes the policy language "Rei", section 5 describes the policy engine and section 6 describes the Beacon. Sections 7, 8 and 9 describe the context manager, policy manager and

the policy enforcer respectively.

4. Policy Language

We use the "Rei"(pronounced "ray") policy language [15] for expressing security policies. Rei is a highly expressive and extensible declarative policy specification language well suited for describing security policies in pervasive environments. Rei includes constructs for expressing rights, prohibitions, obligations and dispensations. It also includes constructs for setting positive or negative modality preferences and allows for stating priority between policies. It models speech acts like delegation, revocation, request and cancellation. This allows policies to be expressed in a less exhaustive way and also allows for distributed policy management. [15] and [14] show how policies can be used for guiding the behavior of entities within a domain. The advantage of using policies lies in being able to modify the security functionality without having to change the implementations of the entities themselves.

Rei defines a policy as a set of rules describing deontic concepts like permission, prohibition, obligation and dispensation over possible actions in the environment with respect to the requester, the action and the context. For example, a privacy policy about not disclosing an SSN number would be a prohibition over taking any action that results in the SSN number being disclosed. Rei allows the inclusion of Prolog-like variables that extend the expressivity of OWL. These variables allow relations like *uncle of*, *same age as*, and *different group from*, that are not directly possible in OWL. Rei also models speech acts for remote policy management like delegation and revocation that affect permissions and prohibitions, and request and cancel that affect obligations and dispensations. Another set of specifications included in Rei are those for meta policies. These are used to resolve any conflict that may arise. For example, if a user is both permitted to and prohibited from performing a certain action, then the meta policies are used to decide whether the permission or the prohibition holds.

Rei is capable of describing deontic concepts over entities and actions based on their properties. Policies can be written in "Rei" that are based on properties of entities and other domain conditions. Actions can be generically described specifying a subject "X" a target "Y" and imposing constraints on both "X" and "Y" to satisfy certain properties. e.g. in plain English, the policy would describe "Action A with target Y can be granted to subject X, provided X satisfies certain properties and Y satisfies certain properties". Consider the following scenario: The "A" lab policy states that devices owned by "A" lab in possession of people affiliated with "A" lab, are allowed to use the capabilities of these devices inside the "A" lab, but not if they leave the lab.

The Rei ontology can be augmented with a suitable domain specific ontology which provides a sufficient vocabulary to describe the security policy. In our current implementation Rei policies are written in RDF. For more expressive policies DAML+OIL, OWL-Lite can also be used.

Thus a policy written in Rei is able to specify in abstract terms the safe or acceptable use policy for the set of trusted devices. The policy enforcer along with the context manager ensure that the appropriate policy is enforced and updated periodically. Due to constraints of space, we refrain from providing a detailed description of the actual policies we have used in the implementation, however the policies can be found online at *http://www.csee.umbc.edu/~anand2/rei*.

5. Rei Policy Engine and Policy Server

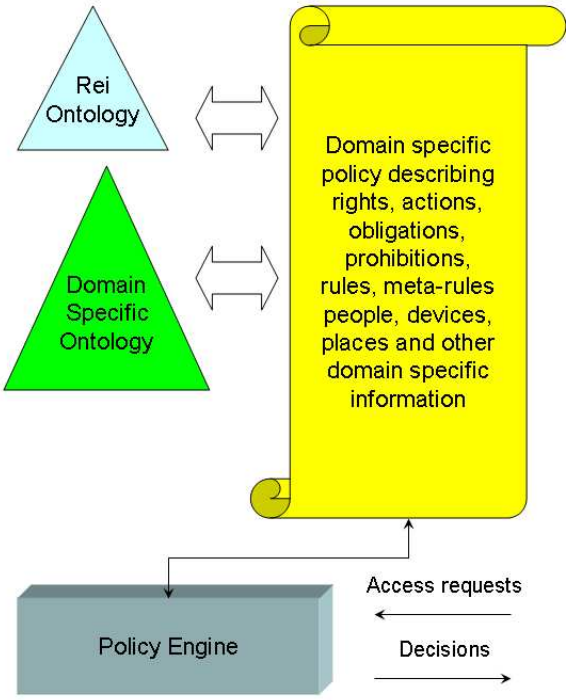

Figure 2. Rei Policy Engine

The security policy is described to the Rei Engine using the Rei Ontology. As shown in figure 2, a domain specific ontology may also be used to describe domain

specific information. The Rei policy engine reasons over the policies described to it in the Rei policy language. The Rei Engine has a Java front-end and uses Prolog for its reasoning engine. The role of the Rei engine is to grant access or deny access to requests made by principals in the domain. The policy server is responsible for handling access requests from the various devices in the system, presenting them to the Rei Engine and then distributing these policy certificates to the requesting entities.

The Policy Server first presents the Rei Engine with the current state information of the device in question, which normally includes in the least, the device identifier, the person in possession of the device and the location of the device. The Policy Server then consults the Rei Policy Engine to create a new policy certificate with the granted requests. The policy server then issues this newly created policy certificate to the requesting device. For this particular scenario, the Rei Engine is loaded with the local network acceptable use policy. Later queries to the Rei Engine provide additional information about the location of the device, user of the device etc., when the new policy certificate is to be issued. The policy server issues the request for resource access to the policy engine. The policy engine reasons over the current status of the system and based on the policy for the role of the subject, issues a policy certificate.

The policy certificate contains the set of permissions or capabilities that are being issued to the device. Additionally the policy certificate contains a validity period and the issuer's identification information. The policy certificate also contains a digital signature which can be verified at the requesting location by the local policy enforcer. Details of the contents of the policy certificate can be found in [11].

6. Beacon

The beacon periodically broadcasts heartbeats intended for the mobile devices. The context manager located on the device listens for these heartbeats. Beacons are intended to serve as location identifiers, so that the mobile devices are informed of their current location from the heartbeats they hear. The heartbeats of these beacons are digitally signed. We use X.509 certificates with the PKI infrastructure. The mobile devices have a set of installed public keys, so the mobile devices can verify the signatures contained in the heartbeats. The heartbeats also provide a set of URIs that indicate the nearest policy server for that network.

7. Context Manager

The Context Manager monitors heartbeats from beacons. A beacon is trusted if the signature is valid and is from a trusted source. Trust is based on installed public keys. Upon receiving the first heartbeat and verifying the signature in the heartbeat, the Context Manager notifies the Policy Manager of the new Policy Server, provided in the heartbeat. The heartbeats provide context information and URIs of Policy Servers. The Context Manager notifies the Policy Manager if heartbeats are not heard for a prolonged interval of time. Both the Context Manager and the Policy Manager are in user space, whereas the Policy Enforcer is in the kernel.

8. Policy Manager

The Policy Manager is responsible for retrieving policies from the Policy Server. It is provided with the Policy Server URI by the Context Manager. The Policy Manager is responsible for verifying the policy certificate using the installed public key certificates. The Policy Manager also logs events and errors generated by both the Policy Enforcer and the Context Manager. The Policy Manager uses SSH sessions to present credentials and retrieve policy certificates from the Policy Server.

9. Policy Enforcer

The Policy Enforcer is the access mediator located on the device. It consists of a set of kernel-resident policy enforcement mechanisms that perform access permission checks based on the policy presented to the device by the Policy Server. The Policy Enforcer is implemented as a set of kernel patches applied to various parts of the kernel including the serial port driver, the IrDA protocol stack, TCP/IP stack, socket manager, PCMCIA card manager, file system etc. A detailed description of the implementation details [11] is available.

The Policy Enforcer is responsible for enforcing the "current" policy that has been verified to have been issued by a trusted source. If no such policy has been received, the Policy Enforcer enforces a default policy which is highly restrictive and allows minimal communication, sufficient for fetching a new policy from policy server.

The Context Manager on the device listens for "heartbeats" from trusted beacons. The beacon signs its own heartbeats and also specifies the hop count of the datagrams that it broadcasts. This restricts usage of the issued policy within range of the heartbeats of the policy server.

10. Example Scenarios

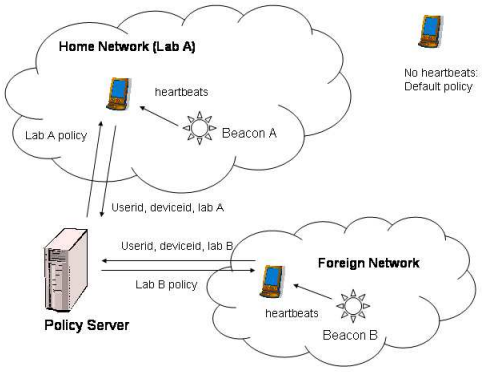

Figure 3. Home network and foreign network

10.1. Home Network

Consider the following scenario depicted in 3. Bob is a Ph.D. student affiliated with the lab "A". He has been issued a mobile device that belongs to lab "A". All the lab devices are equipped with the policy enforcement mechanisms described earlier. The lab "A" policy allows all Ph.D students who are affiliated with the lab to be able to use lab "A"'s resources and use the full capabilities of the device they have been leased, while in the "A" lab.

Bob has authenticated himself to the device and, initially the policy enforcer has enforced the default policy on the device, which allows minimal communication. As Bob walks into the "A" lab, the device hears the heartbeats from a beacon. The device verifies that the signature is from one of the beacons it trusts. The context manager then reads the contents of the heartbeat message and signals the policy manager to retrieve the policy server's address, and issue a request for a policy certificate to the policy server. The default policy ensures that such minimal communication is allowed, though other capabilities of the devices remain disabled. The policy server is provided with the device identifier, the user and the location of the device(based on which beacon's heartbeats were heard). The policy server now transforms this information into domain specific information in the Rei language. Then the Rei Engine is queried for access requests based on the device capabilities. A device specific policy certificate is then created, signed by the policy server and issued to the requesting device. The policy manager on the device issues this policy to the policy enforcer. The new policy is then enforced for the time duration specified in the policy certificate. In this case, since Bob is a Ph.D. student and the device was leased to him, he will be able make unrestricted use of the device capabilities within the lab.

When Bob leaves the lab, and is out of range of the beacon, the device can no longer hear the heartbeats. The context manager on the device resets a timer each time it hears a heartbeat. When no heartbeats are heard for a prolonged interval(twice the heartbeat interval), the timer goes off and the context manager resets the device to use the default policy. The policy certificate is valid only for the time interval specified within the certificate and heartbeats from a trusted beacon can be heard.

10.2. Other trusted Networks

Now suppose that Bob, leaves the "A" lab but is still within the university campus and walks into another lab "B" which has a trusted beacon. This lab however has a policy that foreign devices should only be able to use web services via 802.11 but not use IrDA or Bluetooth. This policy may conflict with lab "A"'s policy that all Ph.D. students be allowed unrestricted use of the device's capabilities. However the meta-rules specified in the Rei language can be used to resolve these conflicts. e.g. the meta-rule may resolve the conflict by specifying that the lab policy where the device is present should have priority over all other policies. Additionally the University policy may have priority over all the lab policies.

11. Prototype Implementation details

The devices we used in our prototype implementation were 3800 series iPaqs, running Linux kernels. The context manager located on the iPaq verifies the signature in the heartbeats using a pre-installed set of X.509 public key certificates(of deployed beacons and policy servers). If either the signature is incorrect or cannot be verified, the context manager ignores the heartbeat. The heartbeat messages contain policy server information (URI of the policy server) from which the policy certificate should be requested. Once the policy server information is available for a particular network, the policy manager contacts that policy server, authenticates itself using (PKI) and requests a policy certificate. In the policy certificate request, the userid and device capabilities are listed, in addition to context information from the heartbeat. The policy server then issues a policy certificate, which is basically a list of capabilities of the

device. e.g. IrDA is disallowed. The policy manager issues this to the policy enforcer, which actually enforces it (like a set of firewall rules). Once enforced, all further attempts to use IrDA result in the attempt being blocked and an informative pop-up on the display of the device, notifying the user that his/her IrDA access attempt has been blocked, since the current security policy does not allow it. Before enforcement however, a signature verification on the policy certificate itself is performed to verify authenticity. If the signature is not verified, the default "shutdown" state results. The policy certificates come with a validity period and also specify a timeout interval for the heartbeats. Allowing for some heartbeats to be missed, a timeout (equal to twice the actual heartbeat interval) results in the policy manager reverting to a default highly restrictive policy. Once a policy certificate expires or is invalidated by a timeout, the device remains in the default "shutdown" state till a new policy certificate is available. The policy server is Java based and uses the Java interface to the Rei Policy Engine. The security policy for a network is described using the Rei ontology and an augmenting ontology, which describes additional domain specific entities like locations, device types, device capabilities, people etc. The Rei Engine uses these ontologies and the policies described to it in RDF and makes decisions on whether to allow or disallow particular access requests. Samples of augmenting ontologies and security policies can be found online at *http://www.csee.umbc.edu/~anand2/rei* and *http://www.csee.umbc.edu/~lkagal1/rei*.

12. Conclusions and Future Work

In this paper we have presented a proof of concept implementation of a policy enforcement infrastructure for mobile devices. We have used a semantic policy language "Rei" to express security policies. "Rei" allows policies to be expressed in higher levels of abstraction without requiring knowledge of all possible entities. Policies can be expressed in terms of domain specific information. The policy engine is used to make decisions of allowing or disallowing access requests from actors in the domain.

In our prototype implementation we demonstrated how a policy can be expressed in the "Rei" policy language using the Rei Ontology and an augmenting domain specific ontology to describe rights, prohibitions, obligations, dispensations an actor has on the domain actions. We showed how a mobile device equipped with a policy enforcer can be used to dynamically change its behavior and capabilities in a pervasive environment using this security infrastructure. We demonstrated the use of the expressivity of a high level semantic language "Rei" for describing system wide policies, the dynamic creation of device level policies, policy distribution and, enforcement of these policies on mobile devices.

As noted earlier, the devices with the policy enforcers are themselves trusted devices and cooperate with the security infrastructure. The policy enforcers serve as automatic guards that enforce the correct policy based on current state of the device. This infrastructure addresses security concerns resulting from vulnerabilities in the software or hardware implementations of the device. The security infrastructure does not protect against intentional misuse or attacks.

An alternative to issuing policies from a Policy Server is to use smartcards that contain the policy certificate [11]. The smartcard adds to the hardware requirements of a device. However it is the least obtrusive, since the policy can be enforced so long as the the card monitor notifies the existence of the card. In case of the Policy Server, the sentient program listens for heartbeats from the beacon. It may happen that during periods of severe network congestion heartbeats are lost and the devices suddenly revert to their default policy which will be very disruptive for the users. However in the case of the Policy Server, the policy certificates are created dynamically and are adapted to the context of the device. Also, listening to heartbeats is usually free since most mobile devices come equipped for wireless connectivity, no additional hardware is required. In case of smartcards, the policy is statically issued and stored on the smartcard, it does not change or adapt to changes in a pervasive environment.

For future work we are exploring the possibility of using the device information and the beacon information echoed by the device to be able to trace a device. A beaconing module on the device itself can help trace the device if it is lost.

References

[1] 2002 CSI/FBI Computer Crime and Security Survey, www.gocsi.com.

[2] IBM and Microsoft Security, http://www.pc.ibm.com/us/security/index.html.

[3] iPAQ H5450 Pocket PC, http://h40055.www4.hp.com/ipaq/5450.html.

[4] A. Aziz and W. Diffi e. Privacy and authentication for wireless local area networks. *IEEE Personnal Communications*, 1(1):25–31, 1993.

[5] S. Bechhofer. Hoolet swrl reasoner. http://owl.man.ac.uk/hoolet/, 2004.

[6] V. Bharghavan and C. Ramamoorthy. Security issues in mobile communications, 1995.

[7] R. Di Pietro and L. V. Mancini. Security and privacy issues of handheld and wearable wireless devices. *Commun. ACM*, 46(9):74–79, 2003.

[8] S. Godik and T. Moses. Oasis extensible access control markup language (xacml). OASIS Committee Secifi cation cs-xacml-specifi cation-1.0, November 2002.

[9] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. http://www.daml.org/rules/proposal/. http://www.daml.org/rules/proposal/, 2003.

[10] W. Jansen, T. Karygiannis, M. Iorga, S. Gavrila, and V. Korolev. Security Policy Management for Handheld Devices. In *The 2003 International Conference on Security and Management(SAM'03)*, June 2003.

[11] W. A. Jansen, T. Karygiannis, S. Gavrila, and V. Korolev. Assigning and Enforcing Security Policies on Handheld Devices. In *Proceedings of the Canadian Information Technology Security Symposium*, May 2002.

[12] W. A. Jansen, T. Karygiannis, V. Korolev, and S. Gavrila. Policy Expression and Enforcement for Handheld Devices. Technical report, NIST, May 2003.

[13] M. Jenkin and P. Dymond. Secure Communication between lightweight computing devices over the internet. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, January 2002.

[14] L. Kagal, T. Finin, and A. Joshi. A Policy Based Approach to Security for the Semantic Web. In *2nd International Semantic Web Conference (ISWC2003)*, September 2003.

[15] L. Kagal, T. Finin, and A. Joshi. A Policy Language for A Pervasive Computing Environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. June 2003.

[16] W. Susilo. Securing Handheld Devices. In *10th IEEE International Conference on Networks*, August 2002.