Evidence-Based Access Control for Ubiquitous Web Services

Nishith Khantal
IIT, Kanpur
nkhantal@gmail.com

Johannes Helander
Microsoft European Innovation Center
jvh@microsoft.com

Benjamin G. Zorn
Microsoft Research
zorn@microsoft.com

Oscar Almeida
Texas A&M University
oscar_almeida10@yahoo.com

ABSTRACT

We focus on using evidence for making access control decisions in ubiquitous computing environments. These environments create new interaction scenarios that traditional access control approaches handle poorly. Our approach views access control as the filtering of messages between communicating services. Services are known only by name, and each service makes access control decisions based on local policies and evidence it gathers about other services. We implement our evidence-based approach with a mechanism analogous to a network firewall, filtering messages going to and from a service. Our evidence-based firewall is responsible for managing evidence, laying a foundation for extensions to it that gather, trade, and evolve evidence over time. This organization leads to an ecosystem of evidence providers that evolve as technology and markets develop. We describe the design of our evidence-based access model, discuss usage scenarios, and present preliminary results. The results suggest that this approach is flexible enough to accommodate interesting ubiquitous computing scenarios and efficient enough to implement on small devices.[1]

1. INTRODUCTION

Ubiquitous computing environments present new opportunities for technology to improve our lives. We want devices, (e.g., cell phones, music players, GPS units, robots, temperature sensor, etc.) in these environments to interoperate transparently and make it easy for us to communicate and share with others. This paper considers the problem of providing access control in such an environment.

One approach to access control is based on authenticating users (see Figure 1a, adapted from [26]). In this model, users are authenticated by providing a password to a central authority. Once they have established their identity, they cross the trust boundary and are granted credentials that then allow them to access resources. This approach is sometimes called the onion model because you are either inside or outside some level of possibly nested trust domains. The onion model becomes problematic when users belong to more than one organization or resources need to be shared across organizations.

Ubiquitous computing environments often cannot use a centralized model because the users requesting resources and the resources themselves are not known a priori. As a result, alternative approaches have been proposed [4, 5, 7, 12, 14, 22]. Making access control decisions in a distributed environment is the basis for much ongoing research and issues such as delegation, roles, distributed authentication, and key distribution have been considered extensively. Approaches that attempt to mimic how trust develops between humans, involving reputation and past behavior, have been proposed [13, 17, 21, 30].

In this paper, we present an access control model that emphasizes the formation, evolution, and evaluation of evidence in making access control decisions. Figure 1b illustrates our approach. In that figure, we see a service requesting access to a resource protected by another service by sending it a message. The resource service decides to allow access by evaluating its access control policy using the evidence it currently has about the requesting service. By evidence, we mean any knowledge that relates to the requesting or responding service, potentially including reputation [21], recommendations, passwords, HIPs [9], CAPTCHAs [27, 28], client puzzles [10, 11, 29], biometrics [24], proximity [19], peer rating [25], credit reports, and quizzes. Evidence can conceivably be gathered, accumulated, and evolved as necessary over time. If a service needs more evidence, it can challenge the requesting service directly (e.g., by requesting a password), or it can request evidence about the requesting service from another service, an *evidence provider*.

Our model captures identity in the body of collected evidence about other services that each service gathers. Access decisions are based on evidence and policies held locally by each service. Each service can have different evidence for the same requesting service, and the local evidence can be thought of as a partial view of the

[1]This paper appears in *W2SP 2008: Web 2.0 Security and Privacy 2008 Workshop*, May 2008, Oakland, CA.

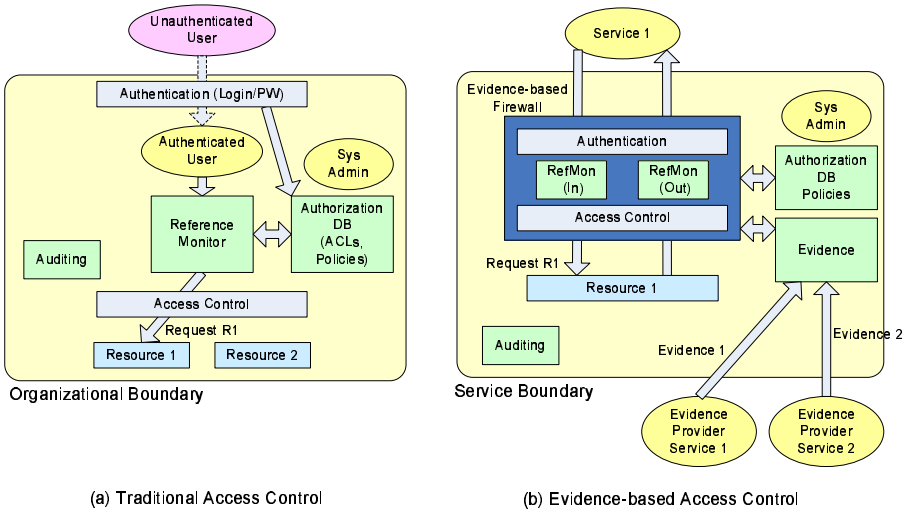

Figure 1: Alternate Models for Access Control

identity of the requesting service. Our evidence-based access model is motivated by the following beliefs:

- *Secure and convenient communication is needed between parties in different administrative domains and in cases where there are no pre-existing domains.*

- *Most data requires convenient, lightweight access control.* Data ranges in value across a spectrum: crucial, interesting, boring, annoying (e.g., spam or pop-ups), dangerous (e.g., contains a virus). [2] Traditional access control approaches focus on maintaining the integrity of crucial data and preventing dangerous data from reaching us. When the cost of an incorrect access control decision is high, effective but inconvenient security approaches are warranted. However, most of the world's data fall into the middle categories. Access control approaches that balance convenience against risk need to evolve to protect these data.

- *Forms of evidence are evolving and the commercial ecosystem of evidence providers, which already exists, will grow.* Evidence is central to our model because the business value of evidence in its many forms is just starting to be realized, and the kinds of evidence used in access control decisions are exploding.

In this paper, we define a model for access control between services in a ubiquitous computing environment that filters messages between services with an

[2]While dangerous data can masquerade as benign data, we do not address the problem of detecting such data.

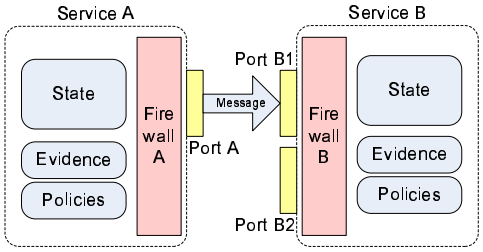

Figure 2: Evidence-based Firewall

evidence-based firewall (EBF)[3]. The evidence-based firewall manages the interaction between services and regulates what services have access to what resources. Figure 2 illustrates our approach, showing two communicating services that each have an EBF processing messages sent between the services. Based on access control policies and known evidence about the sender and receiver of the message, the EBF determines whether the message is allowed to be sent or received. Access control policies allow the EBF to make access control decisions based on a variety of sources, including the sender, receiver, and contents of the message.

We base our framework on the firewall concept because it is well understood, widely used, and effective. Firewalls limit both incoming and outgoing messages, necessary to prevent access and maintain privacy in ubiquitous computing environments. Furthermore, firewalls are also used to filter data that we classify as boring or annoying.

[3]We use the firewall analogy instead of the traditional reference monitor [2] because we model the two entities, requester and requestee, as peers, and assume that message flow will be filtered in both directions. While our firewalls are more intrusive than traditional network firewalls, we retain the name because it is familiar to a broad audience.

The contributions of this paper include:

- We define a set of goals based on the beliefs outlined above and describe an evidence-based access model intended to meet those goals.
- We show how the model addresses access-control issues raised by important ubiquitous computing scenarios.
- We describe a prototype implementation of our access model based on embedded web services and evaluate our prototype's performance.

2. EVIDENCE-BASED ACCESS MODEL

In this section, we introduce the high-level goals of our design and provide an overview of the concepts.

The goals of our model are to:

1. *Support communication between parties in different trust domains or in cases where no pre-existing trust domains exist.* This goal requires that parties can name each other securely and that the model interoperates with existing access models.
2. *Allow evidence associated with parties to be securely collected and evaluated for the purpose of allowing access to resources.* In particular, access decisions should be based on policies that are capable of balancing security with convenience.
3. *Create an ecosystem in which evidence providers can flourish.* This goal requires a system for creating evidence, associating evidence with parties, and trading evidence between parties. New forms of evidence and new policies must be allowed to be introduced freely.

In our work, we separate unique identifiers from the information used to make decisions about access control. We call the identifiers **entities** to highlight the fact that they do not necessarily represent a user, but can uniquely name any available service. We assume that entities are uniquely defined and can provide cryptographic signatures that are sufficiently strong for the task they are being used for based on existing technology, such as a public key infrastructure.

With the traditional concept of user, identity and trust are tightly coupled. Users verify their identity by providing a password, after which, the user and the capabilities of the user are co-mingled. In our model there is no preconceived relationship between entities. The relationship emerges locally at every service based on available evidence.

While entities can be uniquely named and addressed, their names are difficult to remember, communicate, etc. To support a more convenient way to address entities and their related services, we use URIs (uniform resource identifiers), and we call an entity/URI pair a **port**. Messages are sent to and from ports.

A **message** identifies the port it comes from, the port it is being sent to, and contains the structured contents of the message. We assume that messages contain a distinguished field called action, which we interpret as the verb of the message.

Arbitrary services can communicate with each other via messages as long as they know the appropriate port to send messages to. We assume that message integrity is maintained cryptographically and that existing optimization approaches, such as session keys, etc. are used.

We choose not to pre-specify action types (e.g., read, write, etc.) to allow for a rich ecosystem of interactions. For example, on a wiki page, there might be a "verified write" operation that would require the wiki administrator to verify the update.

In our model, a **service** is a container of state, with zero or more ports, that is the sender or receiver of a message, named by a port, whose behavior upon receiving a message depends on the message and the state of the service. Upon receiving a message, a service can change its state and/or send zero or more messages. Services can spontaneously send messages. Messages sent to and from services are first processed by the service's firewall. We take a data-centric approach to services, defining them in terms of the state that they maintain and the ports that they respond to. Services also contain behavior, represented by the actions that messages invoke.

A **resource** is a part of a service's state that can be serialized and sent through a port in a message. A **document** is a resource that has been serialized, for example, into XML. We call the sender of a message requesting access to a resource a **client**.

A **firewall** is a processor, associated with a service, that processes all messages that are sent to or from the ports of the service. A service's firewall may access all the service's state. For every message sent or received, the firewall implements the function f(message, state) that produces one of the following results: 1) the message, in whole or in part[4], is delivered to or from the port, 2) the message is discarded and an error message is returned, 3) the message is silently discarded. If an error message is returned, it may contain additional information, such as the port of an evidence provider. Firewalls manage the information needed to make access control decisions and execute the policies that determine the decisions.

Evidence is any state in a service that its firewall accesses in making an access control decision. **Poli-**

[4]Depending on policies, one possible outcome might be that, for example, sensitive data is removed from a message before it is sent.

cies are the procedures whose evaluation results in access control decisions. While one can make the case that there is a duality between policies and evidence, for simplicity we consider only cases where policies are procedural and fixed by the entity that a firewall protects and evidence is data that is accumulated on a per-requesting-service basis. An **evidence provider** is a service that interacts with the environment and other services and provides data that are pertinent to firewall decisions.

Evidence-based access control can be used to achieve traditional centralized access control if desired. Each user would be an entity and all the resources protected by a domain would require the same evidence, for example, that a user knows a password. Nothing in our model prevents multiple services from synchronizing their policies and evidence if desired.

We would like our approach to allow an evolving ecosystem of evidence technology (Goal 3). For example, if sensors that measure facial expressions are developed, firewalls could incorporate policies that use evidence, creating a market for providers of such evidence. Beyond providing new evidence, evidence providers might also provide evidence management services, including caching, replicating, summarizing, and/or distilling it into higher order evidence.

```
boolean Access?(port sender, port receiver, action, body) {
    boolean decision, error;
    policy p;
    evidence e;

    if (sender == me) {
                p = LookupPolicy(me, "out");
                e = LookupEvidence(receiver);
    } else {
                p = LookupPolicy(me, "in");
                e = LookupEvidence(sender);
    }

    (decision, error) = p(e, action, body);
    if (error) {
                if (protocol_requires_response)
                    Send(receiver, sender, error);
                return false;
    }
    return decision;
}
```

Figure 3: Access Decision Procedure

There are many ways that an access control decision can be made based on the sender, receiver, the message itself, and available evidence. We present pseudocode for a simple access control algorithm in Figure 3, which illustrates an algorithm with separate "in" and "out" policies, and uses evidence associated with the sender while making "in" decisions, and uses evidence associated with the receiver while making "out" decisions. We use such a policy organization in the scenarios that follow. While this figure depicts a Boolean policy, nothing prevents us from providing a more complex policy that allows more subtle access distinctions. For example, combining a non-Boolean policy with partial messages might allow a user with less privilege to make updates to some parts of a wiki while updates to other parts were discarded.

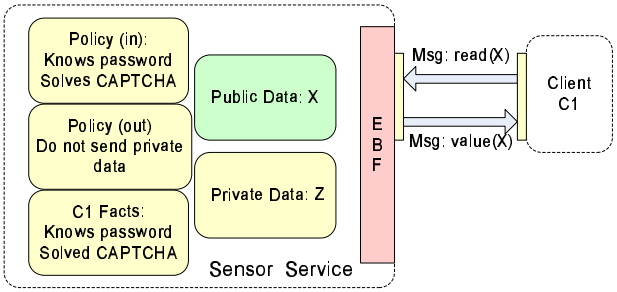

Figure 4: Scenario 1: Sensor Service with Request/Response

With any access control system, bootstrapping must be considered. While ubiquitous web services present some interesting challenges, we believe that as an initial solution, we can rely on existing approaches. For example, every device can have an initial, local evidence provider with a simple default policy, such as providing a password. Device manufacturers can also provide default remote evidence providers (proven to the initial local EP) as a means of bootstrapping the process of finding additional evidence providers. While this approach constrains the scenarios in which a device can be initialized (e.g., when the default remote evidence provider is unavailable), we will consider lifting this limitation in future work.

3. SCENARIOS

Ubiquitous computing has many compelling and interesting usage scenarios. We describe several such scenarios, including casually sharing soccer photos at a coffee shop, in a workshop paper that motivates this work [20]. Here, we present two scenarios, both of which we have implemented in our embedded web services framework, that illustrate the functionality and flexibility of our approach.

Scenario 1: Sensor Service Our first scenario involves a sensor service, where the sensor has public data (X) available to other services and private data that should never be sent (Z) (see Figure 4). We assume that the service has separate policies for filtering incoming and outgoing messages (as illustrated in Figure 3), and maintains evidence for each client. The incoming request policy (Policy-in) requires that the sender know a password and solves a CAPTCHA (e.g., asking the user to recognize a squished sequence of letters and numbers), while the outgoing policy (Policy-out) rejects messages sent with private data.

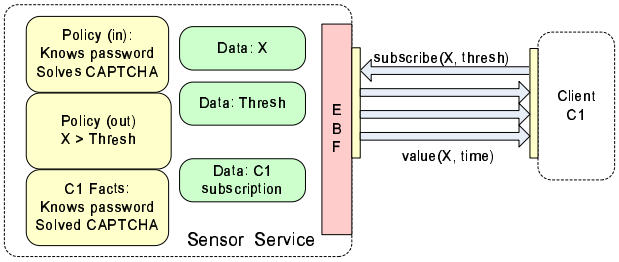

Figure 5: Scenario 2: Sensor Subscription

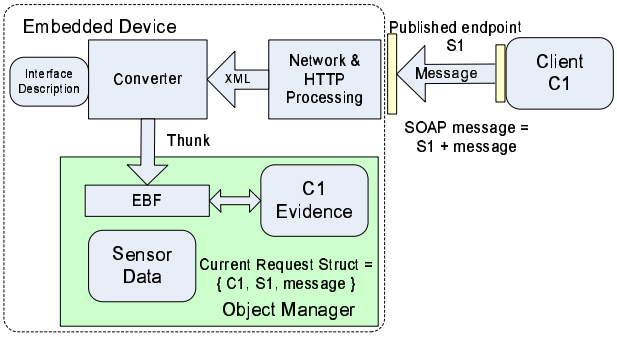

Figure 6: Embedded Web Services Framework

In the example, client C1 issues a read request for sensor data X. The EBF processes the message, applying Policy-in to the evidence it has about C1, and determines if the read request should be allowed. Assuming it is, the sensor service sends a return message to C1 containing the current value of X. Before that message is sent, the EBF processes it, applying the policy that checks the message content for private data before allowing the send. We discuss the case where the read request fails due to insufficient evidence when we describe our implementation below.

Scenario 2: Sensor Subscription Our second scenario involves a client subscribing to a service that publishes a sensor value (Figure 5). In this scenario, the subscription is initiated with a message from the client containing an additional argument specifying a threshold. The intent of the subscription is that as long as the client remains subscribed, messages containing sensor values are sent to the client at regular intervals if the value of X is greater than threshold. The sensor service maintains state associated with C1's subscription, including the threshold value specified C1 when the subscription was created.

In this example, the input policy is exactly the same as in the previous example—the client can only subscribe to the sensor if it knows the appropriate password and has solved a CAPTCHA. The output policy is client-specific and is controlled by the client-specified threshold value. This policy prevents outgoing sensor value messages from being sent when the value is below the specified threshold. From the service's point of view, the threshold data is just client-specific evidence that it manages.

4. IMPLEMENTATION

Our EBF implementation is based on an Embedded Web Services framework [18] implemented as part of the Invisible Computing project at Microsoft Research [15]. Figure 6 illustrates the architectural details of that infrastructure.

Our framework implements service interactions as SOAP messages sent between published web service endpoints. SOAP is a way of turning XML documents, such as serialized method calls, into messages. A SOAP message consists of an envelope that contains a number of headers, such as the intended recipient, and a body with the payload data. The core of the architecture is a data-centric object manager that has been implemented on a wide range of devices including very small ones. The framework allows SOAP to be used directly over hardware, such as serial lines, over UDP or TCP, or wrapped in HTTP. It interoperates with WS-Management, WS-Device Profile including WS-Discovery, and the Distributed Software Services Protocol (DSSP) that is used in the Microsoft Robotics Studio. The interoperation has been tested with a large number of other implementations in several interop workshops. The hardware requirements start from 4 Kbytes of ROM and 128 bytes of RAM on an AVR with a fixed functionality system that understands a couple of messages to a fully extensible system that includes all the protocol layers and the EBF, running in roughly 256 Kbytes of ROM and 32 Kbytes of RAM on, among others, an ARM7.

After the incoming SOAP messages are handled by the network and HTTP stack, the resulting XML is passed to a converter that translates the XML into the native representation of the service's runtime, e.g. C structs, so that the program can access the data directly. The converter automates the process of marshaling messages and is driven by an interface description, which itself can be viewed as the program that the converter interprets. The action field of the message is treated as the method of a method call. The method call is natively represented as a thunk, an executable stack frame. The resulting thunk is passed to the object manager, which executes it to call the appropriate method on one of its objects. Outgoing messages follow a similar path in the reverse direction.

In the process of creating the thunk, the converter generates the Current Request Struct, containing all the information necessary for the evidence-based firewall to make the appropriate access control decisions. The EBF is implemented in the object manager as an

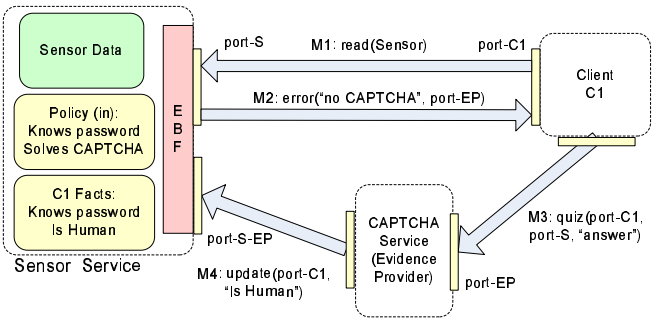

Figure 7: Message Pattern in Scenario 1

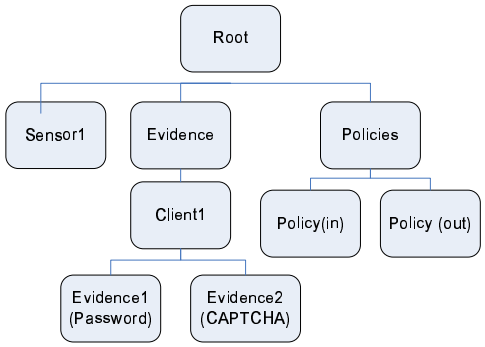

Figure 8: Resource Hierarchy in Scenario 1

interposition that runs just before the service method is called, or, on outgoing messages, just before any message is created and sent, such as just after the service method returns. When the EBF examines a message, it either allows the message to be sent, or discards it. When the message is discarded an error message is generated when one is expected by the protocol. We use a unified naming scheme for services and data, embedding the names into URL prefixes that create a hierarchy (see details below).

4.1 Scenario 1 Implementation

We use Scenario 1 to describe our implementation more concretely. As illustrated in Figure 7, we require the client to both know a password and demonstrate their humanity by solving a CAPTCHA. In this example, the initial request (M1) to read the sensor fails because the CAPTCHA evidence is not present, and an error message (M2) redirects the client to a CAPTCHA service (at port-EP). The client provides CAPTCHA evidence to the service (M3), also providing a port to the service that requires the evidence (port-S). The CAPTCHA service updates the client evidence (M4) using port-S-EP, which it derives from the client's port, port-S, to indicate that the CAPTCHA has been solved. The original request (M1) can then be retried and will succeed (providing the re-request occurs within a time frame during which the CAPTCHA evidence is considered valid). Note that the interaction between the CAPTCHA service updating the client evidence (M4) is itself a service request, which also must be validated through the EBF, requiring the CAPTCHA service message to be evaluated in the context of an input policy specific to updating evidence.

We capture the recursive nature of request processing through the EBF in the object-manager's naming system. Figure 8 illustrates how the object manager namespace is organized in Scenario 1. The figure shows that the namespace is organized into subtrees, where resource specific data is accessible directly based on the URL of the resource (e.g., /Sensor1) , while the evidence related to the resources and clients are accessed within a separate Evidence subtree based on the port (URL+entity) of the client. A separate Policy subtree stores the related policies. In our example, we assume only two policies are needed to filter incoming and outgoing messages.

The tree is populated in the process of creating it. Initially, the resource Sensor1 is installed at the root of the object manager namespace. The message that creates the resource contains the necessary parameters, such as the type of the new resource. This binds the resource to program code (a class) that knows about the particulars of the resource, such as how to fetch the current reading from the physical sensor. The type is also used by the converter in automatically serializing and deserializing the resource into a document that can be sent in a SOAP message.

Before the new resource can be used, access control policies are installed in the Policies subtree tree in a location that mirrors the actual resource's location in the main tree. The hierarchical namespace allows policies that are specific to a particular client or resource. By default, if no such specific policies are present, the default policy installed in the Policies root is used (as in the example).

Later, the client attempts to read Sensor1 by sending it a "read" message. For the purpose of illustration, assume the client has already provided a password to a service that verifies the password against the correct one, essentially performing a login. The login service can either be on the same device or somewhere else. The login service is an evidence provider (possibly local) that sets the knows-password field in the Evidence/Client1 resource. When the client now sends the "read" message the EBF protecting the sensor looks up the client evidence in Evidence/Client1 based on the sender port. It looks up the resource policy in Policies/Policy(In). It then applies the policies to the evidence producing TRUE, the message is delivered, or FALSE, the message is discarded. When a message is discarded, an optional information field (e.g., containing the port of the CAPTCHA evidence provider) will

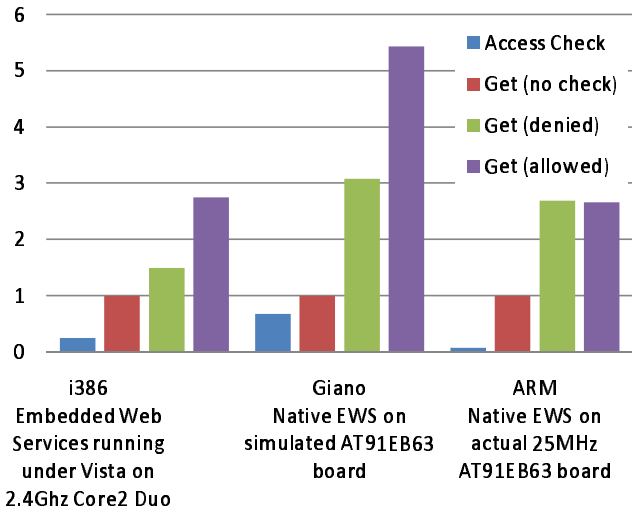

Figure 9: Execution Overhead of Evidence-based Firewall on Three Platforms. Values are relative to Get (no check) = 1 on each platform.

be sent to the client (assuming the protocol state expects a response).

5. RESULTS

As a preliminary evaulation of EBF performance, we implemented Scenario 1 and measured the execution time on three platforms: x86 running Vista; Giano, an embedded systems simulator [16]; and on a 25MHz ARM (see Figure 9). The results do not include any cryptographic signing, etc. We also do not perform any optimizations such as caching the result of access checks or precompiling the security policy expressions (which are now textually parsed and interpreted each time they are evaluated).

Figure 9 shows the relative cost of several operations, all normalized to the cost of a resource Get operation without any access checks (requesting and receiving a 32-bit sensor value). The first bar in each group shows the cost of an access check alone, assuming the check does not access any additional resources (e.g., reading a password, etc); the second bar is the cost of an unchecked Get; the third bar is the cost of a Get with an access check that accesses one additional resource and fails; and the fourth bar is the cost of Get with two access checks, one on the incoming and one on the outgoing message, both of which access an additional resource before returning the requesting value. For these results, we measured each operation 1000 times and computed the average. The mean time for a Get with no access control was 0.038 milliseconds (i386), 0.68 milliseconds (Giano), and 4.3 milliseconds (ARM) per call.

From the figure, we see that depending on the case, the overhead of Get with access checks can be several times slower than Get with no checks. Because access checks in our unoptimized implementation themselves require the equivalent of one or more Get operations, the observed slowdowns are expected. We anticipate that with precompiled policies and access check caching, the peformance can be significantly improved.

In order to show that an EBF can be used in practice, we demonstrated Scenario 1 (a simple sensor service) and Scenario 2 (a sensor service providing subscriptions) with our implementation at the 2007 Microsoft Faculty Summit [1]. The first demo implemented Scenario 1 using the Vista WinRM command line utility to speak the WS-Management protocol between the client and server. A second demo (Scenario 2) showed how Microsoft Robotics Studio (MSRS) could subscribe to an ARM7 based medical sensor using the DSSP protocol. For this scenario, the sensor was an accelerometer embedded in a walking stick for the purpose of detecting its user falling over. The client service, in this case MSRS, subscribes to the accelerometer receiving updates every five seconds.

6. RELATED WORK

There is a large body of work on the general issue of access control in a distributed environment, while less work focuses on the problems specific to ubiquitous computing environments. While some prior work has considered a general framework using evidence for access control, much of the previous work assumes a small, fixed set of evidence sources.

The work most closely related to our evidence-based access model is the dynamic trust model proposed by Xiu and Liu [30]. They also propose a service that collects trust evidence about its clients. They argue that the diversity of ubiquitous computing scenarios demands support for various forms of evidence. In subsequent work, they consider automated trust negotiation for such environments based on agents [23]. Their work differs from ours in that we separate naming from evidence, model access control as a message firewall, and emphasize creating an ecosystem for evidence services.

Ashri et al. describe an approach to making security decisions in a Grid computing environment based on security policies and information [3]. They describe the implementation of their model as a semantic firewall, which, like our evidence-based firewall, reasons about access control based on evidence and policies. Unlike our firewall, they do not anticipate an ecosystem of evolving forms of evidence. Also, because their firewall is embedded into a Grid services infrastructure, they focus on specifying tasks and the related workflow authenticated by site-specific access control policies, while we dissociate user identity from organizational boundaries and focus on the firewall mechanism to provide both access control and filtering.

English et al. consider the issue of dynamic trust in a global computing environment [13]. They choose to define trust as an estimation of likely behavior and focus on an E-purse scenario. Their approach differs from ours in that they look at combining various trust values to make decisions, while we focus on the entire lifecycle of evidence, including who manages it. Further, we dissociate names from evidence, allowing partial identities to emerge locally.

Haque and Ahamed consider properties of trust in a system where trust is based on recommendation [17]. They consider formal properties of trust and how to reason indirectly about trust through chains of recommendation. Our work focuses more on the sources of evidence and the model that allows evidence to be created, collected, and evolved. We can incorporate their results in how we choose to combine evidence in make a particular access control decision.

Microsoft CardSpace (also called InfoCard) has goals that are similar in spirit to ours [6]. Their approach, influenced by Cameron's Seven Laws of Identity [8] is an identity metasystem that allows the creation and management of multiple identities by a user that represent different facets that the user wants to present to third parties, depending on how much the user trusts them (e.g., an anonymous identity, a "verified" identity, etc.) CardSpace focuses on issues related to a human user, such as identity selection, and while it can support a variety of forms of evidence, this is not the main theme. With our EBF approach, the main focus is on the evidence itself. With this focus, we hope to stimulate research in ways to evolve, manage, combine, and generate new evidence.

7. SUMMARY

We describe an evidence-based access model that assumes no centralized domain, focuses on access control to non-critical data, and encourages a robust ecosystem of new evidence and policy providers. Our model binds names to evidence locally, so that every service has a potentially different view of the credentials of client services. We enforce access control by interposing a firewall on every incoming and outgoing message, allowing message filtering based on the sender, receiver, and the contents of the message. Because much data is neither dangerous or critical, relatively benign message filtering is an important part of any access control system. We illustrate how our model works for two ubiquitous computing scenarios, describe our prototype implementation based on embedded web services, and evaluate its performance. Preliminary results show that the overhead of filtering messages using an evidence-based firewall is not prohibitive for the applications considered, even without possible optimizations.

8. ACKNOWLEDGEMENTS

We thank Martin Abadi, Sheikh Iqbal Ahamed, Joris Claessens, Ted Hart, and Gideon Yuval for their valuable and constructive feedback. We also thank the anonymous referees for their invaluable feedback.

9. REFERENCES

[1] Oscar Almeida, Alessandro Forin, Philip Garcia, Johannes Helander, Nishith Khantal, et al. Embedded systems research at DemoFest'07. Technical Report MSR-TR-2007-94, Microsoft Research, July 2007.

[2] J. P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Air Force Electronic Systems Division (ESD/AFSC), Hanscom AFB, Bedford, MA, October 1972.

[3] Ronald Ashri, Terry R. Payne, Darren Marvin, Mike Surridge, and Steve Taylor. Towards a semantic web security infrastructure. In *Proceedings of Semantic Web Services 2004 Spring Symposium Series*, 2004.

[4] Moritz Y. Becker, Andrew D. Gordon, and Cédric Fournet. SecPAL: Design and semantics of a decentralized authorization language. Technical Report MSR-TR-2006-120, Microsoft Research, September 2006.

[5] Blaze, Feigenbaum, and Lacy. Decentralized trust management. In *RSP: 17th IEEE Computer Society Symposium on Research in Security and Privacy*, 1996.

[6] Keith Brown. Managing identity: The InfoCard identity revolution. http://technet.microsoft.com, July 2006.

[7] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[8] Kim Cameron. Introduction to the laws of identity. http://www.identityblog.com, January 2006.

[9] M. Chew and H. Baird. Baffletext: A human interactive proof. In *Proc. of SPIE-IS&T Elec. Imaging, Document Recognition and Retrieval X*, pages 305–316, January 2003.

[10] Drew Dean and Adam Stubblefield. Using client puzzles to protect TLS. In *Proceedings of the 10th USENIX Security Symposium*, August 2001.

[11] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 139–147, London, UK, 1993. Springer-Verlag.

[12] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. *RFC 2693: SPKI Certificate Theory*. The Internet Society, September 1999.

[13] Colin English, Sotirios Terzis, Waleed Wagealla, Helen Lowe, Paddy Nixon, and Andrew D. McGettrick. Trust dynamics for collaborative global computing. In *WETICE*, page 283, 2003.

[14] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[15] A. Forin, J. Helander, P. Pham, and J. Rajendiran. Component based invisible computing. In *3rd IEEE/IEE Real-Time Embedded Systems Workshop*, 2001.

[16] Alessandro Forin, Behnam Neekzad, and Nathaniel L.

Lynch. Giano: The two-headed system simulator. Technical Report MSR-TR-2006-130, Microsoft Research, September 2006.

[17] Munirul M. Haque and Sheikh Iqbal Ahamed. An omnipresent formal trust model (FTM) for pervasive computing environment. In *COMPSAC*, pages 49–56, 2007.

[18] Johannes Helander. Deeply embedded XML communication: towards an interoperable and seamless world. In *International Conference On Embedded Software (EMSOFT'05)*, pages 62–67. ACM, 2005.

[19] Johannes Helander and Yong Xiong. Secure web services for low-cost devices. In *ISORC*, pages 130–139. IEEE Computer Society, 2005.

[20] Johannes Helander and Benjamin Zorn. Medina: Combining evidence to build trust. In *Proceedings of the Workshop on Web 2.0 Security and Privacy 2007 (W2SP'07)*, May 2007.

[21] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[22] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: theory and practice. *SIGOPS Oper. Syst. Rev.*, 25(5):165–182, 1991.

[23] Zhaoyu Liu and Daoxi Xiu. Agent-based automated trust negotiation for pervasive computing. In *Proceedings of the International Conference on Embedded Software and Systems (ICESS'05)*. IEEE, December 2005.

[24] N. K. Ratha, J. H. Connell, and R. M. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Sys. Journal*, 40(3):614–634, 2001.

[25] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, 2000.

[26] R.S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, Sep 1994.

[27] von Ahn, Blum, Hopper, and Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of EUROCRYPT*, 2003.

[28] Luis von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, 2004.

[29] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New client puzzle outsourcing techniques for dos resistance. In *CCS '04*, pages 246–256. ACM Press, 2004.

[30] Daoxi Xiu and Zhaoyu Liu. A dynamic trust model for pervasive computing environments. In *Fourth Annual Security Conference*, 2005.