Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2011)

Procedia Computer Science

International Symposium on Frontiers in Ambient and Mobile Systems (FAMS)

Towards Shrink-Wrapped Security: Practically Incorporating Context Into Security Services

Gleneesha Johnson, Paulo Shakarian[a], Neha Gupta, Ashok Agrawala

Department of Computer Science, University of Maryland, College Park, College Park, MD 20742

[a]*Dept. of Electrical Engineering and Computer Science, United States Military Academy, West Point, NY 10996*

Abstract

The mobile workforce is rapidly increasing, and technological advances make it feasible for these workers to have ubiquitous access to a variety of resources with various protection requirements. The dynamic computing environment of these workers mandates a security paradigm in which security is tightly coupled with a user's current situation. We have proposed a security paradigm to achieve this, called Shrink-Wrapped Security, in which security is constantly adapting to a user's current situation, and a comprehensive amount of security-relevant context is used to characterize a user's situation. We present an approach that uses generalized annotated programs (GAPs) to practically incorporate such context into security services, with a focus on access control. This allows us to represent context in a principled manner; consistently make security-related decisions; easily make temporary, ad-hoc changes to a security policy; and give a user feedback when access is denied so that she can make the appropriate adjustments.

Keywords: Dynamic, Access Control, Mobile Users, Shrink-Wrapped Security, Generalize Annotated Programs

1. Introduction

The mobile workforce, which consists of employees that do not have one fixed place of work, and are linked to a corporate base using a mobile computing device, is rapidly increasing. According to the International Data Corporation, the mobile workforce is expected to comprise 73% of the total United States workforce this year [1]. Technological advances, such as the increasing abundance and convenience of powerful and portable computing devices, combined with the prevalence of wireless broadband availability, create an environment in which these workers can access a multitude of corporate data and resources anytime and anywhere. Sometimes these workers may need to access sensitive resources. In fact, a recent survey of mobile workers conducted by Symantec showed that 62% of respondents planned to access confidential data on their mobile devices [2]. So while technological advances facilitate the practicality of ubiquitous resource access, which has its benefits such as increased productivity, it is critical to ensure that the access is secure and appropriate.

Traditional approaches to security were developed when users were typically computing in a static, stationary environment, and therefore base security-related decisions on static attributes such as identity or role. In the dynamic computing environment of mobile workers, users may: 1) use a variety of mobile computing devices with varying configurations; 2) connect over various networks; and 3) be in varying physical settings when requesting access to remote resources. To achieve effective security in this dynamic computing environment, security decisions must consider the user's context (e.g., co-location, network characteristics, and device characteristics), which can change frequently and rapidly. Context is commonly defined as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves [3].

Shrink-wrapped security, a security paradigm introduced in [4], in which a tight coupling is provided between a user's current situation and security, is essential. Shrink-wrapped security is achieved by constantly adapting security to a user's current situation, and utilizing a comprehensive amount of security-relevant context, to characterize the situation. Various security services (e.g., cryptography, authentication, and access control) can be shrink-wrapped, but in this paper we focus on access control. Shrink-wrapping access control involves not only adjusting a user's permissions, but also the policies (rules that determine access), dynamically based on a user's situation.

In previous work we presented steps towards achieving shrink-wrapped security by developing an approach to identify relevant context and a framework to facilitate the secure acquisition and management of such context [4, 5]. Identifying relevant context has been noted as a key challenge to context-aware system development. To identify relevant context, our approach involves using a taxonomy of security-relevant context, in which context is classified along two dimensions, *the relevant entity*[1] , and the *affected security objective*[2], along with goal-oriented guidelines. The guidelines involve identifying the high-level security objectives of the target security service and the relevant entities for the target domain. For example, the security objectives for access control are *confidentiality* and *integrity*, and the relevant entities for the domain of mobile users attempting to access remote resources are: *the user, the user's computing device, the surrounding environment*, and *the communication mechanism*.

In this paper, we present another step towards shrink-wrapped security with an approach to practically incorporate the *use* of security-relevant context into security services, with a focus on access control. We present the application of generalized annotated programs (GAPs)[6]to this problem, which allows us to represent context in a principled manner; consistently make security-related decisions; easily make temporary, ad-hoc changes to a security policy; and give a user feedback when access is denied so that she can make the appropriate adjustments.

2. Motivating Scenario

Suppose a company has a set of resources, including various files, sensitive business data, and business applications. These resources have different protection requirements based on their importance. Let us assume that each resource belongs to one of three classes, *least important, somewhat important* and *very important*. Let us also assume that different types of resources have different access operations, which define allowable ways that they can be accessed. For example, the access operations for a text file may be *read only, append*, or *read and write*. In order for a user to perform a certain access operation on a resource belonging to a certain resource class, they must have the required permission. A permission is a tuple <*access operation*, *resource class*> that specifies an authorized interaction.

Following the shrink-wrapped security paradigm, our objective is to utilize context to dynamically adjust a user's permissions so that at any given time she only has permissions that are appropriate based on her current situation. In order to explain what we mean by appropriate, let us consider the fact that security in any system should be commensurate with its risks. Risk is a function of the likelihood of threats to the system being realized and the resulting impact[7]. By considering the values of relevant contextual attributes[3], the likelihood of threats to the system being realized can be estimated. The class of a resource will determine the impact of such an exploit (which is organization dependent). Based on these two factors, the most appropriate security measure(s) can be employed, which in the case of access control amounts to granting or denying a requested permission.

Using the guidelines in [4] we have identified a set of contextual attributes that we feel characterize the security-relevant aspects of the situation of users in our domain, and identified possible values for each attribute. We have conducted a survey of security experts to determine how the values of the contextual attributes affect the likelihood of various security threats being realized (see figure 1)[4]. This allowed us to associate a threat likelihood with each value of an attribute and establish an ordering of an attribute's values from least secure to most secure (values with

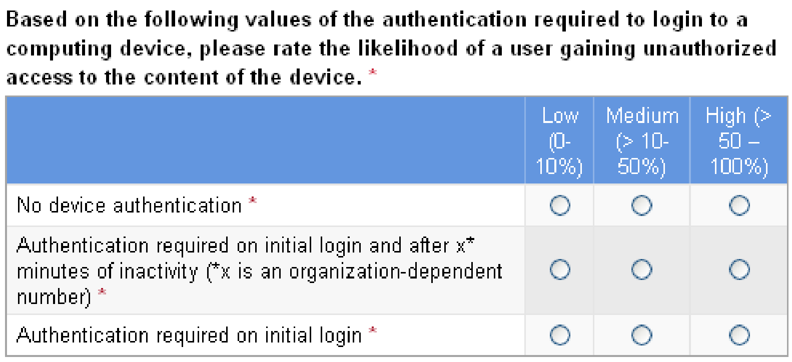
[1]This dimension stems from Dey's definition of context

[2]We consider the fundamental security objectives: confidentiality, integrity, and availability

[3]A contextual attribute is a measurable context primitive (e.g., a users location), and context is the full set of contextual attributes that comprise the situation of an entity[8]

[4]Due to page limitations we have not included a list of the identified attributes nor a full discussion of the survey results in this paper

higher threat likelihoods are less secure than those with lower threat likelihoods). For example, the order of the three identified values for the contextual attribute, *device authentication requirements*, is {*No device authentication, Authentication required on initial login, Authentication required on initial login and after x* minutes of inactivity*}. We also collected information on the perceived level of relevance of each attribute, as some attributes may be more pertinent to making informed security-related decisions than others. Both the threat likelihood and relevance are numbers in the interval [0, 1]. In the next section we present an approach to allow security administrators to incorporate such information into the security policies that govern access to resources.



Based on the following values of the authentication required to login to a computing device, please rate the likelihood of a user gaining unauthorized access to the content of the device. *

	Low (0-10%)	Medium (> 10-50%)	High (> 50 – 100%)
No device authentication *	○	○	○
Authentication required on initial login and after x* minutes of inactivity (*x is an organization-dependent number) *	○	○	○
Authentication required on initial login *	○	○	○

Figure 1: Example Survey Question

3. Using Context to Determine Access with GAPs

Our objective is to develop a principled technique to determine if permissions should be granted or not, based on a user's context. As with previous work such as [9] and [10], we look to represent context with a logic-based framework. In our framework, we have a logic-program that encodes an administrator's policies. This information is supplemented by a user's context. We aim to determine the overall threat level of a user's current situation by combining the two. Our key intuition lies with our choice of the logic-programming paradigm - specifically generalized annotated programs (GAPs). GAPs are logic programs that allow for real-numbered values called *annotations* to be associated with atomic propositions. In using this paradigm, we gain two advantages: 1) we can associate real-numbered values with contextual attributes (rather than just boolean ones), and 2) we can directly leverage results from [6] to correctly determine the threat level entailed by a user's context. In this section, we describe how GAPs can be utilized to incorporate context into access control policies and highlight some additional advantages to our approach.

3.1. Using GAPs for Context-based Security

We represent every contextual attribute, relevant entity, and security objective as an atomic proposition in the set Prop. We also have additional atomic propositions that represent aggregate security levels, which an administrator will in turn utilize to make decisions about access control. We divide Prop into four subsets - $\mathcal{A}, \mathcal{E}, \mathcal{O}, \mathcal{G}$ - containing atomic propositions associated with contextual attributes, relevant entities, security objectives, and aggregates respectively. Specific to our application, we shall also introduce two functions. The first is the function $threat : \mathcal{A} \rightarrow [0, 1]$, which represents the threat likelihood (e.g, .1 for low threat, 0.5 for medium, and 1 for high) associated with the current value of an attribute. The second function, $relev : \mathcal{A} \rightarrow [0, 1]$, represents the relevancy of each attribute (i.e. .1 for low relevancy, 0.5 for medium relevancy, and 1 for high-relevancy). These values are entered a-priori. Administrators may use default values, such as those obtained from our survey of security experts, or they can set them themselves. Now that we have defined our set of atomic propositions, we will define GAPs as per [6].

Definition 1 (annotated atom/GAP-rule/GAP [6]). *GAPs are defined as follows.*

- *Given* $A \in \mathsf{Prop}$ *and annotation* x *(which is a number in* $[0, 1]$*, a variable symbol, or a function over* $[0, 1]$*),* $A : x$ *is an* ***annotated atom****.*

- *Given annotated atoms $A_0 : x_0, A_1 : x_1, \ldots, A_n : x_n$, the following: $A_0 : x_0 \leftarrow A_1 : x_1 \wedge \ldots \wedge A_n : x_n$ is a **GAP rule**. The annotated atom $A_0 : x_0$ is the **head** and the conjunction $A_1 : x_1 \wedge \ldots \wedge A_n : x_n$ is the **body**.*
- *A generalized annotated program **(GAP)**Π is a finite set of GAP rules.*

Specific to our application, we have some restrictions on the composition of GAP rules. For rules with an atom from set $\mathcal{A}$ in the head, the annotation must be a function of the associated *threat* and *relev* functions for that atom. Further, for such rules, the body must be a tautology (see Rules 1-6 of Figure 2). Intuitively, we want the threat level assigned to a contextual attribute to be a function of the threat likelihood and relevancy. If the head of the rule contains a relevant entity atom, then the body can only have contextual attribute atoms associated with that entity[5]. Likewise, rules with security objective atoms in the head can only have contextual attribute atoms associated with that objective in the body. We now provide an example of how GAPs can be used by an administrator to make security decisions based on context.

Example 1. *Suppose we have very simple context providers that can only determine the strength of the user's password* pwd*, the authentication technique used by the user* auth_tech*, who the user is co-located with* co-location*, the antivirus status of the user's computing device* antivirus*, the currency of the user's system patches* patches*, and the connection encryption* con-encrypt*. In Figure 2, we have a portion of a GAP for this scenario Π.*

Rules 1-6 are standard. Each contextual attribute is assigned an initial annotation based on the relevancy of the attribute and the value returned by the threat function. Here, we simply weight the value returned by the threat function using the relevancy value returned by the relev function.

The remaining rules would be created by an administrator. Note, that while the administrator adheres to our previosly mentioned restrictions, she has much flexibility in creating the rules. For example, in rule 12 she aggregates the annotations of the relevant entities by selecting the entity that poses the greatest threat. However, in 14, she aggregates the annotations of the security objectives in a different manner - using the average.

$$\begin{aligned}
\text{pwd} : \textit{relev}(\text{pwd}) \cdot \textit{threat}(\text{pwd}) &\leftarrow && (1)\\
\text{auth\_tech} : \textit{relev}(\text{auth\_tech}) \cdot \textit{threat}(\text{auth\_tech}) &\leftarrow && (2)\\
\text{antivirus} : \textit{relev}(\text{antivirus}) \cdot \textit{threat}(\text{antivirus}) &\leftarrow && (3)\\
\text{co-location} : \textit{relev}(\text{co-location}) \cdot \textit{threat}(\text{co-location}) &\leftarrow && (4)\\
\text{con-encrypt} : \textit{relev}(\text{con-encrypt}) \cdot \textit{threat}(\text{con-encrypt}) &\leftarrow && (5)\\
\text{patches} : \textit{relev}(\text{patches}) \cdot \textit{threat}(\text{patches}) &\leftarrow && (6)\\
\text{user} : \sqrt{x_1 \cdot x_2} &\leftarrow \text{pwd} : x_1 \wedge \text{auth\_tech} : x_2 && (7)\\
\text{computing device} : x &\leftarrow \text{antivirus} : x && (8)\\
\text{confidentiality} : 4 \cdot \left(\sum_i \frac{1}{0.01 + 0.99 \cdot x_i} \right)^{-1} &\leftarrow \text{pwd} : x_1 \wedge \text{antivirus} : x_2 \wedge \text{auth\_tech} : x_3 \wedge \text{patches} : x_4 && (9)\\
\text{integrity} : \min_i x_i &\leftarrow \text{pwd} : x_1 \wedge \text{antivirus} : x_2 \wedge \text{auth\_tech} : x_3 \wedge \text{patches} : x_4 && (10)\\
\text{availability} : x &\leftarrow \text{antivirus} : x && (11)\\
\text{overall} : \max(x_1, x_2, x_3, x_4) &\leftarrow \text{user} : x_1 \wedge \text{computing device} : x_2 \wedge \text{communication mechanism} : x_3 && (12)\\
&\quad \wedge \text{surrounding environment} : x_4 && (13)\\
\text{obj} : \text{avg}(x_1, x_2, x_3) &\leftarrow \text{confidentiality} : x_1 \wedge \text{integrity} : x_2 \wedge \text{availability} : x_3 && (14)\\
& && (15)
\end{aligned}$$

Figure 2: Example GAP (Π) for Context-based access control. See Examples 1-2.

GAPs have a formal semantics defined in [6] that can be used. An interpretation I is simply an assignment of values to all atomic propositions - formally a mapping from the set Prop to $[0, 1]$. Let $\mathcal{I}$ be the set of all possible interpretations. We now define the operator $\mathbf{T}_\Pi : \mathcal{I} \rightarrow \mathcal{I}$, which, given an interpretation, produces a new interpretation

[5]Recall that contextual attributes are classified with our taxonomy by the *relevant entity*, and *security objective*

based on the program Π. We can apply this operator multiple times until the interpretation converges. In[6] it is proven that upon convergence, it has a least fixed point.

Definition 2 ($\mathbf{T}_\Pi$ and Multiple Applications of $\mathbf{T}_\Pi$). *Given GAP Π and interpretation I:*

- *The operator $\boldsymbol{T}_\Pi(I)$ produces an interpretation that assigns atom A an annotation that is the supremum of all annotations assigned to A in the head of a rule in Π which satisfies I. Formally: $\boldsymbol{T}_\Pi(I)(A_0) = \mathbf{sup}\{x_0 \,|\, A_0 : x_0 \leftarrow A_1 : x_1 \wedge \ldots \wedge A_n : x_n$ is a rule in Π and for all $1 \leq i \leq n$, $I \models A_i : x_i\}$.*
- *For application i of $\boldsymbol{T}_\Pi$, we write $\boldsymbol{T}_\Pi^{(i)}$ and define it as follows: for $i = 0$, $\boldsymbol{T}_\Pi^{(i)} = \boldsymbol{T}_\Pi(I_0)$ (where I_0 assigns zero to all atoms), otherwise, $\boldsymbol{T}_\Pi^{(i)} = \boldsymbol{T}_\Pi(\boldsymbol{T}_\Pi^{(i-1)})$.*

Not only are we guaranteed that $\mathbf{T}_\Pi$ has a least fixed point, but [6] also show that this fixed point precisely captures the maximum annotations of the all atomic propositions entailed by Π. Hence, the annotation assigned by the least fixed point of the operator, corresponds to the highest threat level associated with each atom. We can use this information to make decisions about access control.

Example 2. *Consider the GAP Π presented in Example 1. Now suppose that $relev$(patches) $= 1$ and for any atom $A \neq$ patches, $relev(A) = 0.5$. Additionally the threat function returns the following values: $threat$(pwd) $= 0.5$, $threat$(auth_tech) $= 0.1$, $threat$(antivirus) $= 0.5$, $threat$(patches) $= 1$, $threat$(co-location) $= 1$, and $threat$(con-encrypt) $= 0.1$. If we compute the least fixed point of the $\boldsymbol{T}_\Pi$, we obtain an annotation of 0.5 for user, an annotation of 0.6 for computing device, and an annotation of 1.0 for overall. For every access operation-resource class pair, the administrator should set a maximum tolerable threat level for various entities, security objectives, and other aggregates. Using this, a user can be granted or denied access based on the result of this fixed-point computation.*

It is important to note that the fixed point computation takes polynomial time and obtains the same solution regardless of the arrangement of the rules in the GAP specified by the administrator. Hence, we can quickly make consistent decisions about access in a principled manner that is a direct, logical consequence of the administrator's policies.

3.2. Advantages of Using GAPs for Context-Based Security

There are several additional advantages to using GAPs for context-aware security. First, it is very easy for an administrator to make ad-hoc changes to a security policy - and easily undo those changes later. Consider the following example.

Example 3. *Following from Examples 1-2: A security administrator was just informed by human intelligence sources that a rival firm was placing blackhats equipped with sniffers near a certain coffee shop. Obviously, this could seriously impact the confidentiality of resources. He can easily create a function, computable server-side, that can determine if the user is in the vicinity of the coffee shop in question (e.g., by considering GPS or cell-phone tower data). She also (temporarily) adds a new security atom - coffee that is assigned a value of 1 if the user is in a certain radius of the coffee shop and 0 otherwise. Now the administrator adds the following two rules to Π.*

$$\mathsf{confidentiality} : x \leftarrow \mathsf{coffee} : x \tag{16}$$

$$\mathsf{coffee} : threat(\mathsf{coffee}) \leftarrow \tag{17}$$

Rule 16 assigns an annotation to the security objective confidentiality that is the same annotation of coffee. Rule 17 simply assigns the coffee atom the value ascribed by the threat function (relevancy is not considered here as the administrator clearly feels this is urgent). If a user is near the coffee shop, and we re-compute the least fixed point of $\boldsymbol{T}_\Pi$, as done in Example 2, the annotation assigned to confidentiality is now 1.

Another benefit of using GAPs is that we can give a user feedback about why a certain access was denied. This is due to the constructive nature of the fixpoint operator. In our current implementation we can simply track each rule that causes the annotation of the atom in the rule head to exceed the tolerable threat value specified by an administrator. The union of security attributes in the bodies of all such rules is then the set of atoms that led to the security decision in question. Therefore, a user can be informed of the aspect(s) of her context that caused the denial.

4. Related Work

Previously, logic programming has been employed for access control in various ways. In the classic work of [11], the authors introduce flexible authorization manager (FAM) programs that allows an administrator to specify access policies for users and groups. This was later followed by [12], who utilizes constraint logic programming to provide role-based access control (RBAC). However, in an ad-hoc, mobile environment, the user's context must be considered as well. Context was not considered in these previous works. The authors of [9] show that context can be represented with first-order logic. In [10] an extension to Prolog is developed to provide native support for context-aware applications. In [13], logic programming is used to support secure ad-hoc collaborations based on context. However, we are unaware of any previous work that utilizes *annotated* logic in context-aware security decision making. As a result, contextual attributes must be identified with a boolean variable instead of a range of values as we do here by leveraging GAPs. Our use of GAPs allows an administrator more flexibility in how the various contextual attributes are combined, which can lead to the creation of policies that are not possible to express in other frameworks.

5. Conclusion

To enable effective security for mobile users, it is important to address the disparity between traditional, context-insensitive approaches to security and this dynamic computing environment. We believe that addressing this issue requires the tight coupling of security with a user's situation, in other words shrink-wrapping security. In this paper we present another step towards achieving shrink-wrapped security by applying the logic programming paradigm, specifically generalized annotated programs, to context-aware security decision making, with a focus on access control. This approach has many benefits, including: allowing us to represent context in a principled manner; consistently making access control decisions; easily making temporary, ad-hoc changes to a security policy; and giving a user feedback when access is denied so that she can make the appropriate adjustments. We have implemented the **T** operator for GAPs in Java, and created interfaces that allow an administrator to create GAP rules, and specify maximum tolerable threat levels for permissions. Future work includes evaluating the system in a test environment.

References

[1] IDC, Worldwide mobile worker population 2007-2011 forecast.

[2] Tis the season to be....mobile? survey results: Mobile security habits during the holiday season (December 2010). URL `http://www.symantec.com/connect/node/1584861`

[3] A. Dey, G. Abowd, Towards a better understanding of context and context-awareness, in: CHI 2000 workshop on the what, who, where, when, and how of context-awareness, Vol. 4, 2000, pp. 1–6.

[4] G. Johnson, Towards shrink-wrapped security: A taxonomy of security-relevant context, in: Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on, IEEE, 2009, pp. 1–2.

[5] G. Johnson, A. Agrawala, E. Billionniere, A Framework for Shrink-Wrapping Security Services, in: Services Computing (SCC), 2010 IEEE International Conference on, IEEE, 2010, pp. 639–640.

[6] M. Kifer, V. Subrahmanian, Theory of generalized annotated logic programming and its applications, J. Log. Program. 12 (3&4) (1992) 335–367.

[7] C. Pfleeger, S. Pfleeger, Security in computing, Prentice Hall PTR, 2006.

[8] M. Covington, M. Sastry, A contextual attribute-based access control model, in: On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Springer, 2006, pp. 1996–2006.

[9] A. Ranganathan, R. H. Campbell, An infrastructure for context-awareness based on first order logic, Personal Ubiquitous Comput. 7 (2003) 353–364.

[10] S. W. Loke, Logic programming for context-aware pervasive computing: Language support, characterizing situations, and integration with the web, in: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence, WI '04, 2004, pp. 44–50.

[11] S. Jajodia, P. Samarati, V. S. Subrahmanian, E. Bertino, A unified framework for enforcing multiple access control policies, SIGMOD Rec. 26 (1997) 474–485.

[12] S. Barker, P. J. Stuckey, Flexible access control policy specification with constraint logic programming, ACM Trans. Inf. Syst. Secur. 6 (2003) 501–546.

[13] A. Toninelli, R. Montanari, L. Kagal, O. Lassila, A semantic context-aware access control framework for secure collaborations in pervasive computing environments, The Semantic Web-ISWC 2006 (2006) 473–486.