A Secure Composition Framework for Trustworthy Personal Information Assistants*

V.N. Venkatakrishnan
Department of Computer Science
University of Illinois at Chicago
venkat@cs.uic.edu

Wei Xu I.V. Ramakrishnan R. Sekar
Department of Computer Science
Stony Brook University
{weixu, ram, sekar}@cs.sunysb.edu

Abstract— *In this paper, we provide a framework that supports composition of individual agents that enables users to accomplish complex tasks that would otherwise be laborious and difficult with mere use of traditional keyword based search engines. A key benefit of our approach is that in the framework the personal information handled by the agent system is guaranteed to be free from accidental leakage to websites that are not trustworthy, thereby ensuring the privacy of end-user data. We describe our approach with a prototype example which suggests that such highly usable, trustworthy agent systems can be built and deployed quickly with modest implementation efforts.*

1 INTRODUCTION

The growth in the number of users who routinely use the web to search for information on goods and services continues unabated. In this expanded use of the web it is not uncommon for end users to view the web as a database and search for information in ways that cannot be directly accomplished by traditional keyword-based web search engines, e.g., *"make a list of used Jaguar cars advertised in New York City area, such that each car is a 1999 or later model, has good safety ratings, and its selling price is less than its Blue Book value"*. Answers to such queries are of considerable interest, since they provide valuable information to compare and contrast competing products and services. However, answering such complex queries requires extracting and integrating information from multiple sites into one coherent view.

*This research was supported in part by Computer Associates, ONR grant N000140110967, and the Sensor CAT at Stony Brook (a NYSTAR Center for Advanced Technologies).

1.1 Motivating example

Let us consider a typical travel planning scenario. Suppose that a user plans to have a trip to Las Vegas from New York. She wants to buy an air ticket from a budget airline such as JetBlue or SouthWest after comparing the schedule and prices of the flights offered by these two airlines. Let us suppose that she also needs to book a hotel room near her destination address, and get the driving directions from the airport to the hotel. Surely users can directly go to these individual web sites and do all of this effort themselves. However, this process is rather inconvenient and tedious (involving a lot of user interactions such as clicking follow-up links, involving repeated input of query conditions).

The deployment of several travel portal web sites (e.g. Expedia.com, Orbitz.com) which provide an integrated interface for on-line reservations of air tickets and hotel rooms and other travel services has eased the burden on this effort. Two important disadvantages make these portal efforts less desirable:

- *Difficulty of customization.* One drawback of such portal sites is that they are usually rigid in the choices of search engines that are deployed and the sites that are visited, and do not offer facilities for customizing the search engines or the set of sites visited to the user's choice. For instance, information of flights from airlines such as JetBlue and SouthWest are usually not included in these portal sites such as Expedia or Orbitz. In addition, users may also have their own preferences in choosing hotel or driving direction searching engines.
- *No certification of security.* Usually, almost all of these portals give almost no technical certification that the privacy of information that is being supplied to the portal site is indeed kept private. Note that there are of course legal assurances that are provided, and certification of encryption schemes that are used in the process of transmission of data, but none of these give any assurance about how private data is being used in the portal. In the absence of any such assurance mechanism, users would prefer to use their credit card

information only at the time of purchase, and will prefer not to store their private information in such portal based websites.

To address the first issue, agent systems such as Personal Information Assistants [1] are beginning to emerge. PIAs are software robots (otherwise popularly refereed to as web agents, shop-bots, etc) that can navigate web sites and extract information automatically on behalf of a user or a system. Users can now delegate complex keyword searches (e.g., Google), parametric queries (e.g. part number information at Texas Instruments), or web-site wide harvesting activities (e.g. bids from e-bay) to PIAs. Thus, they enable users to rapidly access rich, real-time information from one or more web sources without having to locate the sites or even know the source of the information, and without requiring active co-operation from information providers owning the Web servers. Users only need to provide the PIA with searching conditions and payment information once, the PIA will automatically navigate to the relevant sites and fetch the desired searching results, such as the prices and schedule of different flights from different airlines, the location and compare price of the nearby hotels, and display the results after finishing all these tasks.

Furthermore, the underlying technology behind such systems can be tailored to drive portals such as Orbitz.com that were mentioned earlier. Henceforth, we do focus only on PIAs and do not discuss portal based approaches any further in this paper.

This brings us to the second issue of data security and privacy that is addressed in this paper. Note that PIAs are attractive from a security viewpoint, as they facilitate decentralization of private data: Users can keep their private data in their own systems and allow the PIAs to access and provide such data to commercial sites only when necessary, as opposed to the idea of storing them in a third party website that they may not have full confidence in.

There are two crucial aspects to security that confronts the deployment of such PIAs.

- *Information privacy.* Suppose a user wishes to deploy the airline reservation PIA to make travel reservations on the web. The important question is the following: Can the user provide sensitive data such as her credit card and other personal information (such as passwords to websites) to such a PIA, and have some assurance about its trustworthiness? In another instance, can a PIA that merely fetches information (and does not conduct any e-commerce transactions) be trusted to keep the user's profile information private?

- *PIA Integrity.* In addition to these confidentiality issues, there is a (second) trustworthiness aspect of PIAs. They may base their purchase decisions by deriving information from sites that may be untrusted. Although the agents themselves may come from trusted sources, the information that is supplied to them from websites may not be trustworthy. Furthermore, this information may be used in a crucial decision by the PIA. To cite an example, an agent may derive information about certain stocks from some sites that may offer advice on buying stocks. However, this information may not be completely trustworthy, and thus must not be directly used in purchase decisions.

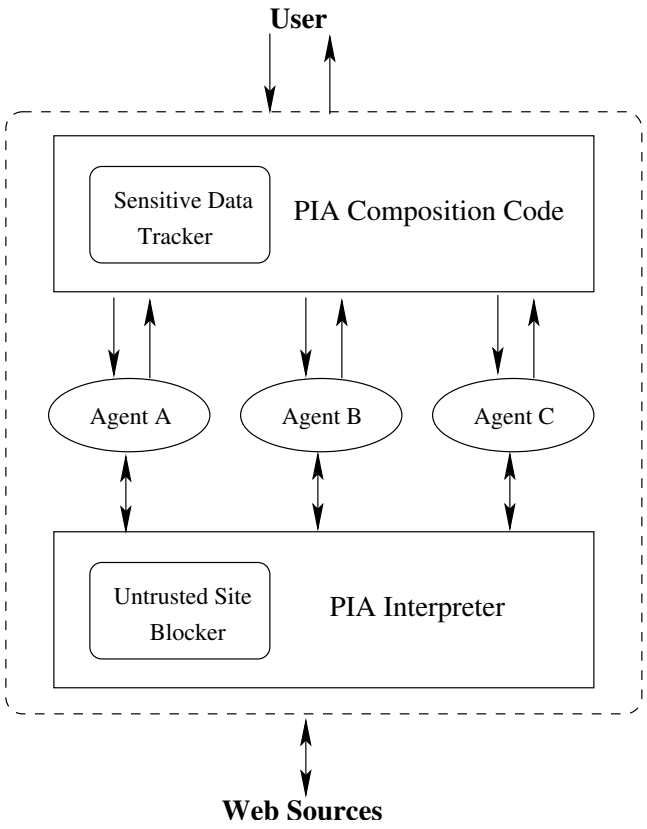

Figure 1: Framework architecture

In addition, by "gluing" such agents together (*agent composition*), one can construct higher-level PIAs by combining the behaviors of individual agents. For example one can envision the travel reservation PIA to be composed from two agents for obtaining fares from JetBlue and SouthWest, plus a hotel reservation agent, and a distance querying agent for Mapquest.com and so on. To build a travel planning application as described before, we need to compose all these individual agents together. In this paper, we present a framework that provides secure composition of such agents. (For the remainder of this paper, we shall use the term agent for referring to those that query a single domain, and the term PIA for the composite program).

The security community has been investigating with program analysis techniques for securing programs from such

information leaks [6]. Existing techniques for analysis of programs (such as type analysis [10, 4]) are not particularly suitable in the case of PIAs, mainly due to the fact that PIAs are highly data driven. To address these issues, we propose a technique based on program transformation to enforce confidentiality policies at runtime. The individual agents and the composition PIA code are first examined through a static analysis, and then the PIA code is transformed with in-line checks that track the flow of sensitive information to websites visited by the (composed) PIA. Whenever a potential information leak is detected that is not in accordance with the user's security policy, the activities of the PIA are aborted, and an exception is raised.

Figure 1 gives an overview of our framework. The individual agents are represented by ovals in the center of the figure. These agents are glued together using the PIA composition code. The PIA interpreter executes each individual agent by sending the corresponding web requests. Two additional components are part of our security infrastructure:

- *Sensitive Data Tracker.* This component is part of the PIA composition code, and consists of in-lined checks that track the use of sensitive information supplied by the user.
- *Untrusted Site Blocker.* This component is part of the PIA interpreter. This component makes use of the security policy that is in force, which states the domains that are trusted by the user. If sensitive data is supplied to a site that is not in the list of domains mentioned in the policy, then this component restricts access to those websites and prevents the PIA from performing any further actions.

This paper is organized as follows: In Section 2 we give a brief background about agent creation and deployment. Section 3 describes the compositional framework, and Section 4 discusses our approach for ensuring trustworthiness of PIAs. In Section 5, we present an evaluation of our prototype implementation. Section 6 discusses related work, and we conclude in Section 7.

2 WinAgent System

We first see how each individual agent can be created. Our agent build environment is the WinAgent system for creating and executing agents which can extract structures and organize the data from these pages in user-specified presentation format (such as HTML, XML, Text, WML, VoiceXML). More importantly the end user can create and execute agents using just a web browser. Essentially the user needs only highlight in the browser examples of data of interest in a web page and the links to be followed and/or forms to be filled to reach this page. From the highlighted examples the system creates an agent by "learning" navigation and extraction expression. In this section we will present an overview of how to create and execute an agent in the WinAgent system. A more detailed description of the WinAgent system can be found in [1].

2.1 Building an agent

The WinAgent system consists of an agent builder through which a user can create a navigation map for a web site. The agent builder is embedded as a tool bar in Internet Explorer. Resembling a VCR, there are five buttons ("Get Item", "Get Region", "Record", "Stop", "Play") on the tool bar, through which users can interact with the system for creating and interpreting navigation maps.

The agent creation is initiated when the "Record" button is clicked. From now on until the "Stop" button is clicked, the user's actions are monitored and recorded in the navigation map. Using the "Get Region" button the user specifies the region of interest by highlighting it on the page. The user uses the "Get Item" button to select an example item of interest. The algorithms used in the agent builder for creating navigation maps by learning from a user's web browsing and interactions are described in [1].

This navigation map encodes information about how to navigate to pages of interest in a web site, and consists of a collection of page-maps that specify how to extract data needed from these pages.

2.2 Launching the agent

Users can use the "Play" button on the agent builder toolbar, or the independent agent manager, to specify a navigation map and launch the agent interpreter on this map.

The agent interpreter interprets this navigation map by automatically navigating to the web site, following specified links, filling out forms and extracting the targeted data from the pages as specified in the page-maps. The output of the interpreter is an XML document with attribute names that were supplied by the user when the agent was created.

2.3 Presentation of extracted data

The raw data extracted from the agent is organized as an XML document in a hierarchical fashion. At level one of the output tree, there are objects corresponding to the first level of navigation. These objects contain their extracted attributes as children. They also contain level two objects corresponding to the second level of traversals and extractions and so on.

The WinAgent system supports transforming such a nested XML into one of many desired presentation formats, such as comma-separated records, VoiceXML, HTML, etc. Such transformations on XML are expressed using XSLT stylesheets. A generic GUI framework is provided for rapidly creating such XSLT's, with an interface for users to plug in a specific formatting module for transformation.

3 COMPOSITION

Let us revisit the PIA example again. As we already noted, the PIA queries the websites for Southwest Airlines and JetBlue Airways and retrieves the lowest fare and also retrieves the lowest hotel fare at the destination city. Now, it is a waste of effort to build a complete monolithic PIA exclusively for doing this set of tasks. We would like to build individual agents that retrieve information from each of the above websites, and then *compose* them to achieve the overall functionality. The decomposition of the functionality into individual agents has the following benefits:

- *Enables composability and re-use.* Individual PIAs can be built for various websites and then composed together in a variety of situations. Consider another PIA application that synchronizes one's personal schedule against reservation availability for Southwest flights. Now the agent that obtains reservation information from southwest.com can be re-used in this situation by composing it with another agent program that operates on the user's personal schedule. One can imagine a "portal" of such agents that is available for composition and implementation of more PIAs.

- *Eases creation and deployment.* As already mentioned, one of the main features of the WinAgent system is that the creation of individual agents through macro recording facilities. It is simpler when such creation of PIAs is restricted to one website, so that the extraction toolbar could be used effectively for this purpose, instead of burdening the agent creator with multiple websites and matching schemes. In addition, the logic that is part of the composition code is de-coupled from the actual pattern-matching logic that is part of the agent system, and hence results in a clean, extensible and lightweight design of the WinAgent system.

We shall use some formalism to introduce the composition framework. We can construe every agent program as a function that maps a set of inputs to a set of outputs. Let $(i_1, i_2, i_3, \ldots, i_n)$denote a set of inputs to an individual agent A. Then let the corresponding outputs be $(o_1, o_2, o_3, \ldots, o_m)$. Now each of the individual inputs can come from the following types: `integer, real, string, array, record`. Now, each composition can be understood as a function that maps the outputs of agent A to inputs of agent B.

The composition constructs are written in a simple, imperative language that uses the above-mentioned primitive and complex types. The language includes the following constructs: basic arithmetic, logical expressions, facilities for string matching and editing, loops and procedure calls.

Such compositions can be achieved in different ways using semi-automatic or manual approaches. [5] presents one such approach for composition. We do not discuss aspects of automating this composition in this paper, and, in the following section focus on the security aspects of our framework.

4 ENFORCING PRIVACY AND INTEGRITY

Before describing our approach to securing PIAs, we present an assumption that underlies our approach. Note that the PIAs themselves are not obtained from untrusted sources, and our effort is only in the direction of adding trustworthiness to an already existing agent framework, and to prevent any accidental disclosure of sensitive information. It would be very difficult, if not impossible to address security issues if the PIAs themselves are untrusted. Also, as mentioned earlier, the decentralized nature of PIAs make them a compelling alternative to portal based approaches for handling private data, and our efforts are in the direction of making them more trustworthy.

To achieve this, two critical problems, privacy and integrity, must be addressed. To protect privacy, the sensitive information should be prevented from being sent to untrusted sites; while to protect integrity, the computation of trustworthy data should not be comprised by data from untrusted sources. Privacy and integrity in in fact are duals of each other [6]. Being very closely related any solution that is suitable for privacy can be modified to address integrity as well, as shown in [6]. In this section, we present techniques to ensure privacy protection, and sketch how integrity can similarly be addressed.

4.1 Sensitive data tracker

Our approach is to use a program transformation technique to transform the composition code for detecting policy violations. Runtime updates and checks are inserted into the transformed code and are used to track information flows. Our transformation is based on the transformation technique proposed in [9], where we also prove the correctness of the transformation with respect to the non-interference property [3], which ensures that the confidential information cannot be leaked to untrusted outputs.

In our approach, two security levels: *high* and *low*, are defined for the variables in the composition code. The *high* variables represent sensitive data (e.g., credit card numbers, social security numbers). The *low* variables represent untrusted sites or data coming from such untrusted sites (e.g. a map search engine that should not take any credit card numbers), and hence any assignment of sensitive information to these variables constitutes a policy violation.

Information flows are tracked at runtime through the use of *security* variables. The security variables are auxiliary variables that are inserted by the transformation for each

program variable and used for recording the current runtime security level of their own corresponding program variables. When an execution reaches an assignment statement to a *low* variable, the value contained in the security variable corresponding to the source variable is examined for the presence of sensitive data (i.e. whether the security variable value is *high*). If such sensitive data is indeed present, an error is generated, and the execution of the PIA is terminated.

The simple idea illustrated above would work only for *explicit* flows in which data flows through various assignment statements. On entering a conditional branch, there is an *implicit* flow of information from the conditional to all the variables assigned in both the branches. To compute this set of variables a preliminary static analysis is used. The transformation then makes use of the result of this analysis.

The analysis computes the following: for any expression e, $var(e)$ is the set of all variables that are used in the expression. The `if S` denotes a (possibly compound) statement, the set $upd(\mathtt{S})$ denotes all the variables that get assigned in the statement `S`. The analysis computes a conservative upper bound in case a precise estimation of the set of variables that are updated is not possible.

Using the results of the analysis, the program is transformed such that, on entering a conditional branch, the value of the security variable corresponding to the condition expression (associated with enclosing nested conditionals) is stored in a variable called pc_{aux} that is used to track implicit flows. For all assignments that happen inside both branches, the security variables of the left-hand side expressions are updated with the value of the implicit flows, that is obtained by a disjunction with pc_{aux}. Thus, pc_{aux} includes the combined effect of the enclosing conditionals and loops. When a conditional branch is exited, the value of pc_{aux} is restored to its previous value that existed before entering the branch.

For all the variables that are represent sensitive input data or trusted sites, their security variables are initialized to true. All the other security variables are initialized to false. More details on the transformation and its proof of correctness are available in [9].

To support the integrity protection, we just need to reverse the definitions of *high* and *low*. The detected illegal information flows will then represent integrity policy violations.

4.2 Untrusted site blocker

To prevent a trusted agent from accidentally leaking sensitive information to untrusted sites, the PIA interpreter is modified to include a component called *untrusted site blocker*. This component refers to the current security policy which lists domains that are trusted by the user. During the interactions between agents and sites, this component examines the sensitivity level of every field of data in each form which will be supplied by a agent to any untrusted sites. If the *untrusted site blocker* detects that any sensitive data is being supplied to a site that is not in the list of domains mentioned in the policy, then the *untrusted site blocker* restricts access to those websites and prevents the PIA from performing any further actions.

5 Evaluation

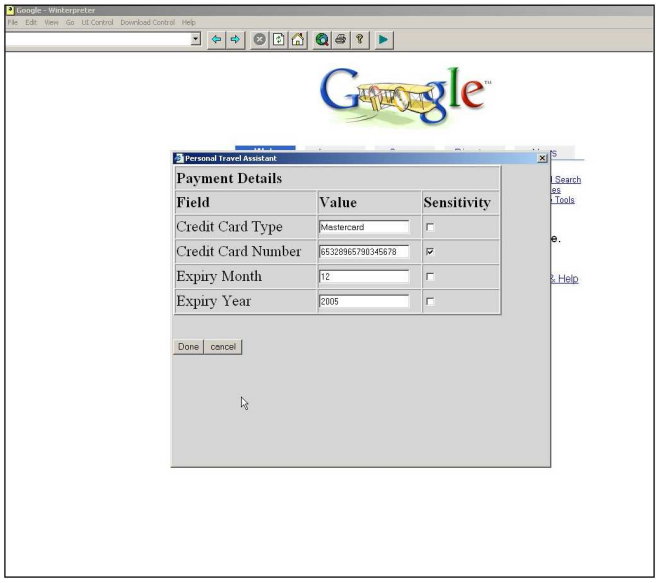

Figure 2: Specifying data sensitivity

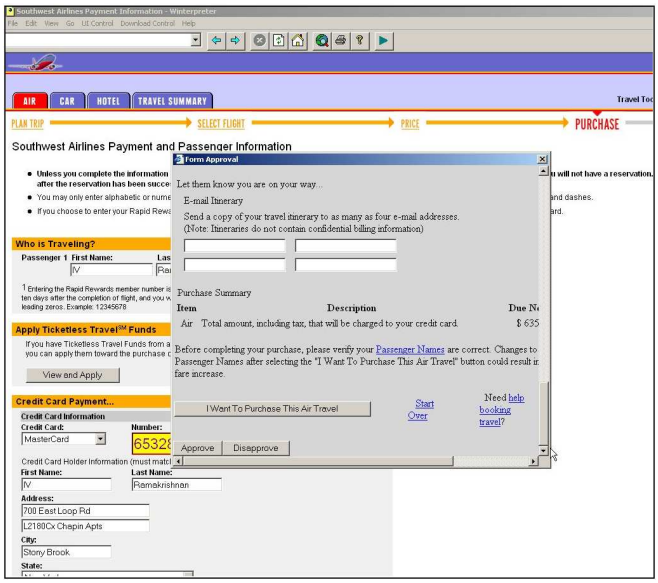

Figure 3: Approving sensitive data transmission

We have implemented the motivating travel planning PIA as a prototype application of our secure composition frame-

work using the WinAgent system. Figure 2 shows the interface for users to specify the data sensitivity level, while Figure 3 shows the review screen where users can decide whether or not to approve the sensitive data transmission. A demo video of this prototype is available at [11].

Building such a composition PIA seems complex, but in our efforts we found it to be a relatively simple job. It only took 1–4 minutes to create each of the individual agents using the WinAgent agent builder. To compose these agents together, we had to understand their inputs and outputs and the comparison logic, thus it took about three hours to manually write the composition code. It is quite straightforward to transform the composition code to track information flows based on the rules of each kind of statement in the program, and we spent about one hour on this. Of course, the foundation for this rapid prototyping and development was the WinAgent system consisting of the builder and interpreter which has been operational for the past one and half years.

When this PIA is used to compare prices and book air tickets and hotel rooms for booking this business trip, the user first needs to input the source/destination addresses, her credit card information, and the sensitivity levels of these data. After that, the PIA will automatically navigate to the relevant sites and retrieve the desired flight/hotel information. The user then will get selection windows to make choices from the search results and proceed through the PIA's transaction approval windows to complete the reservations. This whole process of using the PIA takes about 4 minutes and 10 mouse clicks, and the user needs to fill in 3 forms with 12 fields in total (including source/destination addresses and credit card information). In contrast, if the user herself manually browses these web sites and completes the same searches and reservations, she needs to have more than 40 mouse clicks, fill in more than 10 forms using the same 12 fields of data, and input the URL of these web sites multiple times. In addition, the user has to remember the sites (so that the results obtained are available for bookings at a later instant) and manage multiple browser windows which could become cumbersome and tedious. From this, we can see the PIA greatly eases the user's burden of unnecessary manual interactions with various web sites for retrieving information of her interest, and provides nearly the same level of assurance in handling the user's personal information.

6 Related work

Although there are several agent systems [5, 2, 7] that support composition of individual agents, we believe that the current work is the first to look at data security issues in an agent-composition setting. For instance, SWORD [5] uses a rule based expert system to guide the composition, however, no assurance argument of security of the composite service is provided.

The security community has been investigating with program analysis techniques for securing programs from such information leaks [6]. Existing techniques for analysis of programs (such as type analysis [10, 4]) are not particularly suitable in the case of PIAs, mainly due to the fact that PIAs are highly data driven.In [8], Huang et al., provide a mechanism for addressing data security in PHP programs through static analysis and runtime monitoring. Their approach involves a typing based solution that is in fact more expressive than the previous approaches. However, the expressiveness in their approach does not scale to avoid false positives in information flows such as implicit information flows.

7 Conclusion

In this paper, we presented a framework for secure composition of agent to enable creation and deployment of PIAs. A unique aspect of our framework is in addressing data security through the use of program analysis and runtime monitoring techniques. We illustrated our approach with a prototype example that suggests that our approach is suitable for a wide variety of emerging web applications.

Acknowledgements We would like to thank Akshat Khandelwal, Anupama Lolage, Priyanka Vasudevan and Jayant Kashyap, for assistance with various aspects of the Winagent system implementation.

References

[1] A. Gandhre, P. Santhanagopalan, P. Singh, D. Ramavat, I.V. Ramakrishnan, and H. Davulcu. Creating and managing personal information assistants via a web browser: The WinAgent experience. In *Workshop on Information Integration on the Web*, Toronto, August 2004.

[2] L. Liu, C. Pu, and W. Han. Xwrap: An xml-enabled wrapper construction system for web information sources. In *International Conference on Data Engineering (ICDE)*, San Diego, CA USA, February 2000.

[3] John McLean. Proving noninterference and functional correctness using traces. *Journal of Computer Security*, 1(1), 1992.

[4] A. C. Myers. JFlow: Practical mostly-static information flow control. In *ACM Symposium on Principles of Programming Languages (POPL)*, January 1999.

[5] Shankar R. Ponnekanti and Armando Fox. Sword: A developer toolkit for web service composition. In *Eleventh World Wide Web Conference (Web Engineering Track)*, Honolulu, Hawaii, May 2002.

[6] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1), January 2003.

[7] A. Sahuguet and F. Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowledge Engineering*, 36(3):283–316, 2001.

[8] Securing Web Application Code through Static Analysis and Runtime Protection. Yao-wen huang and fang yu and christian hang and chung-hung tsai and der-tsai lee and sy-yen kuo. In *Thirteenth World Wide Web Conference*, New York City, 2004.

[9] V.N. Venkatakrishnan. *Enforcement Techniques for Expressive Security Policies*. PhD thesis, Stony Brook University, 2004.

[10] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security (JCS)*, 4(3):167–187, 1996.

[11] Personal information assistants website and on-line demos. http://www.lmc.cs.sunysb.edu/~winagent.