DYNAMIC TRUST MANAGEMENT

Matt Blaze, Sampath Kannan, Insup Lee, Oleg Sokolsky, and Jonathan M. Smith, *University of Pennsylvania*

Angelos D. Keromytis, *Columbia University*

Wenke Lee, *Georgia Institute of Technology*

Trust management forms the basis for communicating policy among system elements and demands credential checking for access to all virtual private service resources—along with careful evaluation of credentials against specified policies—before a party can be trusted.

A service-oriented architecture (SOA) separates functions into services, which process requests from peers over a network. In processing a request, the service can, in turn, send requests to secondary services and so on.

The Global Information Grid (GIG), an ongoing effort by the US Department of Defense (DoD) and Intelligence Community (IC), rationalizes and modernizes the architecture of US network-centric operations. It couples a common network architecture to advanced information assurance techniques and, as GIG's name implies, focuses on the information the network carries and the services it provides, rather than on the network's attributes.

There are clear tradeoffs among security, flexibility, and cost in possible designs for such SOAs. Traditional (pre-GIG) DoD network architectures have created logical *airgaps* between different networks such as the NIPRNET and SIPRNET, and services are replicated in each such network environment. Information security is, in principle, guaranteed with separated networks, since there is no network path from the more secure to the less secure network.

Although the GIG is a DoD-specific project, many of the trust management problems it exposes also occur naturally in existing and emerging commercial and other public networked computing environments, particularly those based on SOAs. In particular, traditional decentralized trust management architectures,[1] while useful, do not directly address questions such as policy changes under rapidly changing network conditions or revocation and autonomous versus centralized control. These problems occur in any large-scale system based on a rapidly changing, potentially unreliable network framework such as the Internet. Therefore, we believe that the GIG architecture is a useful platform and opportunity for studying trust in large-scale computing in general, not just in the military and government.

GIG CHALLENGES

In practice, the GIG architecture has several problems. First, all nodes on the secure network must be trusted to operate at their designated security level, so accidental physical access—such as that by a laptop user—can break the security model. Second, the architecture can make sharing appropriate information difficult or impossible. Finally, the broad division of information into security levels such as unclassified, secret, and top secret does not provide fine-grained access control based on the need-to-know principle, which is key in securing information.

The research problem faced by SOAs, then, is to provide fine-grained access control to services and information within the context of a shared network infrastructure. Our conception of the access-control challenge in dynamic environments might best be differentiated from previous ideas, such as role-based access control (RBAC), by calling it mission-based access control (MBAC).

Fine-grained access controls, such as those required for access on a need-to-know basis, require fine-grained specification of policies, sometimes to the level of individual users and objects, but certainly at the level of roles and services. We believe that fine-grained access control is possible by using a formal specification of policy with a policy language that can be understood by both managers and interconnected systems that must make decisions to permit or deny access. Once such a policy is specified, the specification can be used to check access decisions for files on a computer, database records, imagery, Web services, or real-time chat.

TWO EXAMPLES

Large-scale distributed malice provides our first example. In this scenario, malicious actors use *botnets*—connections of hijacked machines that coordinate operations with each other—to carry out a wide variety of actions.

The future promises more malice on a larger scale. A detected botnet should result in a security policy change such as access to services and protection of data management facilities. A central research question we address considers how to react at machine speeds to an apparently distributed adversary given that botnets operate from diverse locations. Clearly, the reaction itself must be global, and it must be rapid to minimize damage to information services. The "shields up" decision for each network, object, and service should let systems operate autonomously if required.

Dynamic team formation for disaster relief provides our second example. Suppose a search-and-rescue team is operating a disaster-relief effort in the aftermath of a hurricane or flood that has struck an urban area. Rescue team members possess GPS-augmented communications devices with a tracking capability that lets them visualize their locations relative to each other. Given that this is a worldwide disaster-relief effort, other teams must coordinate with an independent force-tracking technology.

For the duration of disaster relief, the location information should be shared because everyone has become part of the same team. Thus, this dynamic policy must authorize access to friendly location information and take into account issues such as personally identifiable information that cannot be shared between the communicating parties.

> **Trust management provides the basis for communicating policy among system elements.**

This effort raises questions regarding how accesses are authorized in such a situation and how accesses to particular services can be kept from extending to all networked services. It also presents a complex information-management challenge that must be met in a way that does not require complete access to networks and servers to provide a necessary capability. Other issues involve

- determining how to grant access rapidly in the face of a changing policy, as well as how to revoke it;
- deciding how resources must be defended from unauthorized accesses at a fine-grained level; and
- assessing how coordinated support for information-sharing with team members will affect the resources required of the GIG.

CHALLENGES FOR SOAS

In complex SOAs the problem becomes even more difficult. The goal might not be to rigidly enforce complete separation at all times due to mission demands and the flexible roles the network must take, such as support of dynamically changing disaster relief teaming. Rather, it might be desirable to maintain a high degree of separation between applications most of the time, but to relax or strengthen this separation under specific circumstances in response to emergencies or rapidly changing conditions.

For example, during specific operations or emergencies, the need for relaxation of normal policy might occur in response to explicit decisions by a central authority, or the triggering of a predetermined risk-management strategy—for example, for data in which the sensitivity decays over time. An increased security posture might be assumed when available network sensors, such as intrusion-detection systems, provide situation awareness that indicates an increased threat level.
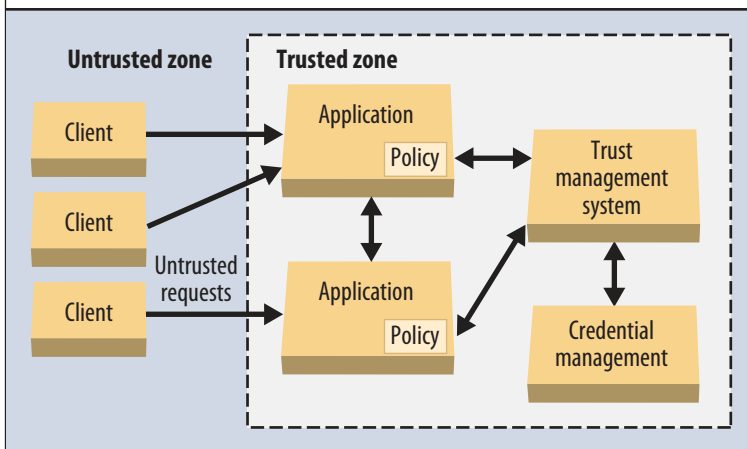
We see the challenge as dynamic provision of virtual private services (VPSs)—services for which access is controlled on the basis of a security policy. Trust management provides the basis for communicating policy among system elements. Trust management systems demand credential checking for access to all VPS resources, along with careful evaluation of credentials against specified policies before a party can be trusted. Thus, the default assumption is to not trust. An architecture based on trust management systems and languages therefore provides an extremely promising approach for analysis and compliance enforcement in systems with complex architectures. If successful, such an approach will enable the secure composition of services needed to adaptively achieve mission-critical tasks in the short timeframes that today's time-sensitive military environments mandate.

DYNAMIC TRUST MANAGEMENT

At first glance, the hierarchical deployment of existing trust-management systems fits well with the concept of service orientation. Specification and enforcement of a security policy for a given service are decomposed according to the service's structure and partially conferred on the secondary services in terms of their policies.

In our past work, we defined and formalized trust management as an explicit policy compliance layer for decentralized systems[1,2] and developed practical trust management languages and systems for small- and medium-scale applications such as distributed firewalls[3] and *virtual private services*.[4] A VPS contains components distributed over several hosts on a network. A central authority specifies security policies, but the host enforces the policies according to policy rules applicable to the VPS component deployed on that host.

Despite the seemingly good match, existing trust-management approaches are clearly insufficient for SOAs. The key problem is that policies specified by existing trust management systems are static: a VPS policy identifies precisely the subservices and hosts they are deployed on and prescribes the policy enforcement strategy. To support a modern SOA, a policy specification should be dynamic to accommodate changes in both the system and its environment.

Dynamic service availability

The network might have multiple hosts capable of performing the necessary service. To make things more complicated, service availability changes dynamically. Dynamic service discovery now forms an integral part of most modern SOAs, and should be accommodated by the SOA's trust management system. Not all alternative services can be equivalent from the security perspective. Some services might require that a request have a higher degree of trust to gain access to the service. Others might have inferior trustworthiness if they are offered by less-trusted hosts, and thus cannot be used to serve some classes of requests.

Situational dynamism

The system's changing environment provides the other source of dynamism in SOAs, when the same request might be processed differently depending on the situation. Suppose, for example, that a request requires access to a reliable and secure set of terrain data. In a particular situation, this set of data might be inaccessible, but less reliable or less secure data might be available. A static policy could reject this request because the required data is unavailable. However, even a less reliable result of the request might be critical to ensure an effort's success. A dynamic policy will let the system either deny the request or service it with inferior data.

In its simplest form, situational dynamism can be implemented by means of a set of predefined security modes. A security mode could be switched either manually by a person with sufficient privileges—by a commander in the field, for example—or by the enforcement engine automatically, according to a given criteria set.

TRUST MANAGEMENT ARCHITECTURE

Existing trust management systems such as KeyNote rely on a strict boundary between the trusted and untrusted zones. Figure 1 shows the KeyNote system's architecture. Requests from the untrusted zone sent by clients and peers are processed centrally by the trust management engine according to the policies of individual applications. This model works well for small- and
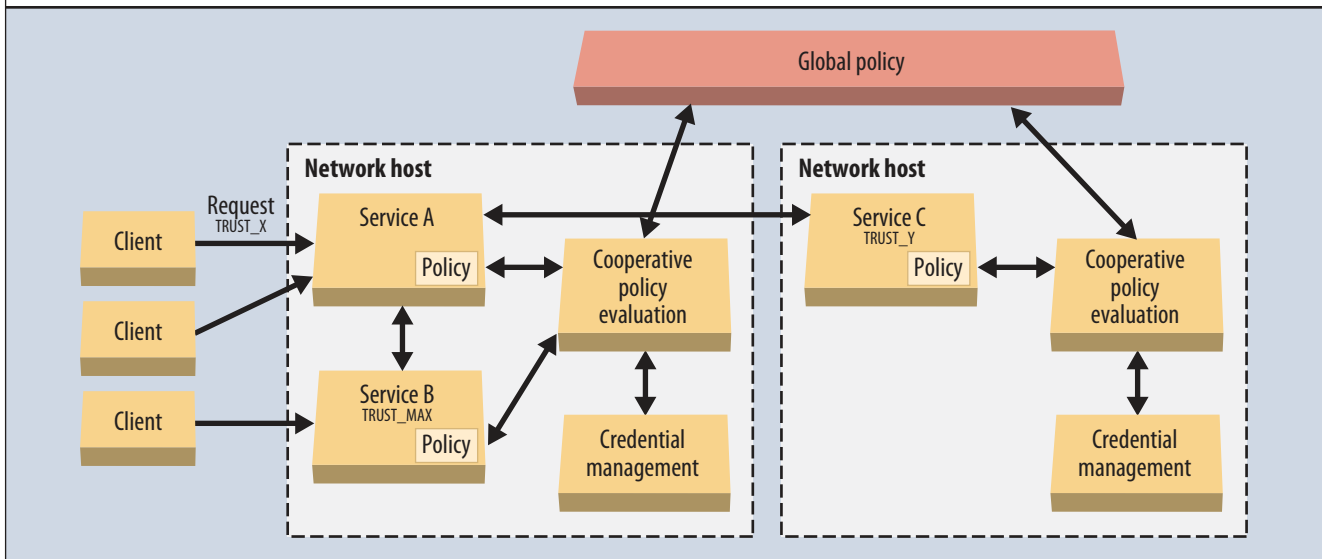
Figure 2. KeyNote system dynamic trust management architecture. Requests from the untrusted zone sent by clients and peers are processed by the trust management engine in a centralized fashion according to individual applications' policies.

medium-scale applications in which there are well-defined administrative and topological boundaries between internal and external services. But the model can make it difficult to support complex systems where these relationships are more fluid.

By contrast, dynamic trust management does not rely on fixed boundaries between trusted and untrusted components. Instead, each principal in the system, such as a service, derives a trust level for each principal with which it interacts. This trust level will be derived dynamically.

Figure 2 shows a possible architecture for such a system. The trust levels appearing in Figure 2 are shown from Service A's perspective, which receives a request from some client with the trust level TRUST_X. To process this request, Service A must send secondary requests to Services B and C. Service B is deployed locally with Service A and has the highest level of trust.

Service C, on the other hand, is deployed remotely and has a lower level of trust, TRUST_Y. The policy for Service A, then, is stated in terms of the dynamic trust levels for incoming requests and subservices that A uses. If the trust level of Service C is deemed too low to process the request, the request might be denied or an alternative to C might be sought.

The idea of cooperative policy evaluation[5] is the starting point for our dynamic trust management system. A global policy controls evaluation of trust levels for principals in the system.

TRUST POLICY LANGUAGE

The trust-management approach[1] frames security questions as follows: "Does the set C of credentials prove that the request *r* complies with the local security policy

P?" This approach subsumes traditional authentication and certification questions under an action authorization model. In this model, remote requests with security implications are authorized or denied based on local policy in conjunction with credentials and authenticated identity.

The policy and credential language with which we will conduct our work will be based on KeyNote,[6] with extensions we will add to support dynamic policies. In particular, we will introduce new constructs to the language that support an active trigger mechanism for policies and tested conditions. The trust management model implemented by the current KeyNote system (and most other systems) is entirely passive. Policies and assertions are written in a scripting language in which an authorizer trusts one or more licensees to perform actions that match certain conditions. For example, a simple access control credential might be written as:

```
Authorizer: "rsa-hex:1023abcd...."

Licensees: "dsa-hex:986512a1..." ||
"rsa-hex:19abcd02..."

Comment: Authorizer delegates read
access to

either of the Licensees

Conditions: (file == "/etc/passwd" &&

access == "read") -> "true";

Signature: "sig-rsa-md5-hex:f00f5673..."
```
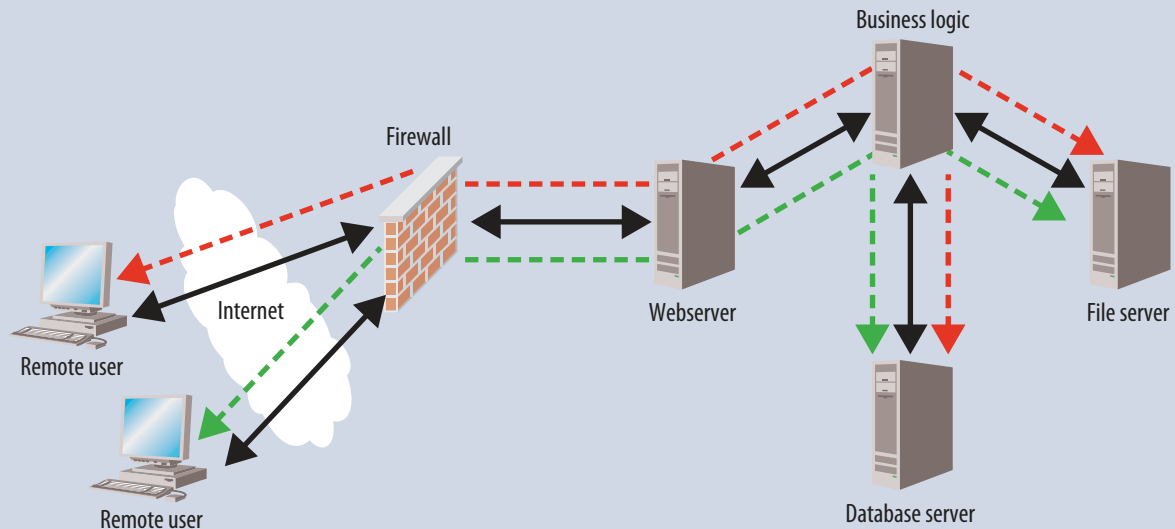
Figure 3. Interaction between components of a webserver in the context of sessions initiated by external users.

This approach has proven useful for small- and medium-scale systems in which all security-sensitive applications and services can query the trust management system explicitly whenever it receives a potentially dangerous remote request, but it does not easily support tight coupling to network conditions or actively *pushing* new policies into remote systems.

Developers are extending KeyNote to support an active model in which both the Licensees and Conditions can include not only passive pattern matching but also active triggers that can be executed automatically in response to changing conditions.

We are investigating two kinds of active triggers: predicates and actions. *Predicate triggers* cause a local policy to be evaluated asynchronously on a network element whenever the external state matches some predicate. *Action triggers* push policy change information out to other network elements and can appear as part of local policy.

In our current research, we address the exact syntax and semantics of these active trigger mechanisms, which allow highly dynamic trust management policies tightly coupled to network health and changing policy. For example, predicates could be triggered when the local network detects some botnet-style attack behavior, allowing the system to push out a policy that introduces more restrictive access control rules:

```
authentication_failures(hosts>4) ->
require_certificates()
```

Other approaches could be tailored to different threat levels.

COOPERATIVE POLICY EVALUATION WITH FEEDBACK

To evaluate dynamic policies, we are designing and implementing novel mechanisms for collaborative decentralized policy enforcement. To that end, we propose a new model, *dynamic policy evaluation*. In DPE, security policy decisions can be revisited at any time during the session's lifetime (with the term session informally defined as a temporally extended sequence of security-relevant interactions among those components of the distributed system that collectively handle a specific request) and may recommend actions beyond the typical *permit/deny* outcome of such security policies.

The access-control mechanisms that govern the distributed system's components participating in a session form a logical ad hoc clique for exchanging security-critical information during the session's lifetime. The clique avoids the performance and complexity of having all such components communicate with each other at all times. The exchanged information includes policy decisions made by the various components during the session, changes in the session environment—such as when traffic starts arriving over a wireless link—and information from other "sensors," including intrusion detection systems, behavior-based anomaly detectors, and credential revocation. The dynamic policy evaluation model is reevaluated as new information becomes available, and privileges could be revoked or restricted as a result.

Consider the simple system shown in Figure 3, consisting of a website that uses a firewall, a front-end webserver such as Apache, a back-end business-logic server running PHP or JavaBeans, a file server storing static content, con-

figuration files, executables and scripts, and a database storing order and customer information.

In this configuration example, an unauthorized wireless access point located inside the firewall perimeter lets outsiders access the webserver, whose security policy assumes that any traffic reaching it must have been authorized by the firewall. Given that there is no way to validate this assumption—which changed due to external, unforeseen factors—the outsider can be granted access under false premises.

Further, once admitted by the access-control process of one component, the user can interact with the remainder of the system without much supervision by that first component. An attacker can probe the system for weaknesses without fear of losing already established access. For example, an attacker who exploits a misconfiguration of the firewall to probe the internal webserver's scripts for SQL-injection vulnerabilities will have his access restricted only after an administrator (possibly prompted by an intrusion-detection system) takes action. Although the firewall continues to verify the conformance of each packet to policy, the attacker's misbehavior is invisible to the firewall.

We observe that, although the system components work together in handling application requests, there is no cooperation in determining the proper security context for authorizing these requests. Currently, there is no mechanism through which security policy can reevaluate the privileges of that user and indicate some necessary action. For example, the firewall policy might request a reauthentication of the user, the webserver might decide to handle that user's requests under a more restrictive policy, and the database might let the user issue queries but not update any tables.

Although it would seem straightforward to manually address the security problems in a small environment such as this example, configuration errors can lead to insecure postures even in configurations involving just one firewall.[7] The complexity of verifying security policy correctness and safety for a large nontrivial system—such as a financial-services firm with 50,000 servers, 90,000 desktops, 2,500 financial applications, hundreds of entry points, and a large supporting infrastructure—is beyond the state of the art.[8] Other examples of such systems include military networks, online gaming services, large ISPs and ASPs, and e-commerce sites.

There is no unified policy-based mechanism through which to scalably handle access control, intrusion detection, and other recovery mechanisms consistently across a large distributed system. We need a way for the security policies across all these mechanisms to continuously validate the assumptions upon which access was initially granted, taking into consideration additional information as it becomes available.

Further, because we cannot determine the true intent of a user or system component that appears to be misbehaving, the security policy must have a larger repertoire of reactions than simple accept or deny. Again in our example, possible reactions beyond completely revoking the user's access might be to slow down the handling of requests (potentially while notifying the administrator), redirecting traffic to an appropriately instrumented instance of the webserver that might be much slower but will detect a wide variety of attacks, or request additional authentication and migrate the server and that user's files to a honey-pot-like system that enables recovery from malicious changes to persistent storage.

DPE offers several advantages over traditional access-control models. First, it unifies access control and intrusion detection under a common security policy, allowing administrators to make better use of them. Second, it lets the

> **There is no unified policy-based mechanism through which to scalably handle access control, intrusion detection, and other recovery mechanisms consistently across a large distributed system.**

system react to changes in the security environment faster, while remaining under security policy guidance. Third, it allows integration of mechanisms that go beyond the simple permit/deny approach of access-control policies, enabling finer-grained reaction to potential misbehavior.

Our approach can conceptually be viewed as complementary to static policy-verification techniques: By allowing component policies to exchange information relevant to future and past decisions, we can continuously verify the assumptions upon which statically verified policies are based and confirm their soundness and any deviations while the system operates.

Currently, our work proceeds along three fronts: formal, systems design, and experimental.

- We are developing a model for DPE based on our previous work on trust-management systems.[6] Our starting point is the PolicyMaker[1,2] evaluation model; our concept of cross-layer communication stems from thesis work on the Strongman system.[9]
- We are investigating integration of intrusion detection and other security-event generators with access-control mechanisms and other appropriate response and recovery mechanisms, such as slowdowns in response to attack.[10] We plan to build research prototypes demonstrating the proposed model for realistic

environments, starting with deployments and experimentation in lab environments, and scaling up to department-scale infrastructures and beyond, as opportunities for collaboration and deployment in other environments arise.

- Our current plan seeks to experimentally validate DPE through participation in "capture the flag" experiments in a "quantitative trust management" effort. This experimentation will seek to determine our model's effectiveness and shortcomings, identify possible ways the system can fail, and develop techniques and mechanisms that can prevent or mitigate the impact of such failures.

> **Trust management provides a unified approach to specifying and interpreting security policies, credentials, and relationships.**

SYSTEM PROTOTYPING AND EXPERIMENTAL EVALUATION

To evaluate the effectiveness of our approach to dynamic policy, we will implement and validate the DPE algorithm experimentally in realistic service-oriented environments. We will evaluate the strongman scaled enforcement of access-control policies to large-scale environments by translating high-level general security policies into components specialized to address multiple local-enforcement points, but will leave policy correctness unaddressed. If the high-level policy made incorrect assumptions, local components could not detect this and recover and, due to the common translation process, would all be affected by the flawed assumption.

A self-recovering system must dynamically discover the set of components that should be exchanging information (a "community of interest," as it were) and identify the conditions under which policy revaluation should occur. Distributed firewalls[3] and distributed intrusion detection[11] provide a basis for statically determining possible component interactions and the types of information that might be exchanged. Strongman introduced the notion of a *composition hook*, a piece of information exchanged between two policy enforcement mechanisms residing at different layers of the network stack in the same system.

Composition hooks are provisioned at policy-generation time, based on the high-level policy specification, and provide a simple mechanism for dynamically coordinating different security mechanisms that coexist in the same system, such as IPsec and SSL in a webserver. This mechanism will be generalized to runtime events to permit ad hoc component interactions and integration with other dynamically generated information, such as IDS alerts.

We will use a distributed blackboard on which policy assertions post (and read) access-control decisions and other events of interest. Key foci in the detailed design of the system will be scalability and tradeoffs between performance and effectiveness. We believe that significant gains can be achieved in reducing false positives and creating more agile systems that can better react to changing circumstances. DPE policy evaluation will be based on the KeyNote model, which allows for answers to policy queries from among a totally ordered set of valid responses, such as "permit unconditionally," "permit with additional monitoring," "permit in honey pot," "deny," or "deny and ban the user."

The systems and software artifacts expected as the result of this work include middleware for creating sessions, integrated intrusion and anomaly detection capabilities, and behavior tracking. We plan to integrate existing security mechanisms that provide gradual responses, such as "shadow honey pots," filesystem/database tracking/journaling, network rate-limiting, and reauthentication into our system, and then develop new mechanisms as our understanding of cooperative policy concepts in the proposed work matures.

That maturation process will require developing new policy models and language support for policy specification. Our choice of KeyNote is beneficial because of its wide use and deployment on Apache webservers, which are representative of SOAs. Experimental evaluation of the prototype will be performed on a surrogate SOA configuration with multiple independent but collaborating webservers, consistent with SOAs. Each system will be equipped with a database, fileserver, and firewall components. The system will also have its defenses "red-teamed" using "capture the flag" exercises with careful postmortems to strengthen our agile dynamic defense architecture for MBAC.

SYNERGIES

Trust management provides a unified approach to specifying and interpreting security policies, credentials, and relationships.[2] We define some important trust management terms informally. An *access request* seeks access to a resource, possibly in a specified mode. A *policy* is a specification of conditions under which access may be granted. A *credential* is a claim of meeting the conditions of some policy. A *transaction* is an access request followed by the granting of the request and subsequent access to the resource. An *agent* or *component* is any entity that interacts with other agents in the system by means of transactions. An agent is trusted in a transaction if its access request is granted.

The KeyNote system's[6] goal is to define notions of trust using policy specifications and to check that a transaction request has the credentials necessary to satisfy the

relevant policy. Thus, rather than simply classifying the world as trusted and untrusted, this approach allows more-sophisticated policies and notions of trust. For example, trust that an unknown party's public key is correct can be built up by having trusted parties certify this to be the case. Thus, systems such as KeyNote let users have very fine-grained notions of trust and manage trust flexibly. An obvious but important point is that most trust management systems, including KeyNote, start with a complete absence of trust between parties. Only the active presentation of a satisfactory credential can overcome this lack of trust. We believe that an architecture based on trust management systems and languages offers an extremely promising approach for analysis and compliance enforcement in SOAs.

Several features of current trust management systems such as KeyNote are particularly attractive for SOAs. First, policies can encode complex rules and risk management strategies appropriate to a particular application or service. We can then analyze these polices for various required properties.[2] Second, "credentials" are digitally signed and written in the same language as policies, making it possible to centrally control and dynamically modify the policies that govern even highly decentralized distributed systems.[1]

Researchers have successfully applied the KeyNote trust management language and compliance checking architecture (albeit at a smaller scale) to several subproblems of the large-scale SOA problem. In particular, KeyNote has been used as the basis for policy control in network-layer security (IPsec) and to control the interaction between application- and network-layer policies, as in the webserver (SSL).[9] Further, KeyNote has been used to encode complex risk-management strategies in a micropayment architecture that combines offline authorization of low-risk transactions with online control over higher-risk actions.[12] KeyNote has also been used as the policy layer for flexible system-call-based process execution supervision, such as distributed file systems with credential-based access control.

Work on next-generation management of scalable trust (Strongman) demonstrated the scalability of a trust-management-based architecture to manage large collections of networked systems.[9] The separation of compliance checking from policy enforcement and the use of caching demonstrated particular advantages, ideas applied to the construction of a scalable distributed network boundary controller.

We continue to investigate the use of trust management techniques to specify dynamic policies in complex integrated service-oriented networks. For this work, we use the DoD GIG's service-oriented architecture as a focal point.

In this research's initial phase, we are developing prototype dynamic trust management policy services for a service-oriented architecture. We base our initial design on our existing trust management system and language, KeyNote, and on our prior work on distributed firewalls[3] and virtual private services.[4] The service will provide a standard compliance checking interface to the various services running on the architecture.

In our research's next phase, we will develop and analyze policies with properties that maintain strict separation between services while allowing exceptions. In particular, we will focus on supporting two kinds of exceptions. One will allow explicit centralized control, such as issuing orders that make classified information available to battlefield networks during operations. Another will encode risk management strategies that allow exceptions based on predetermined criteria.

Finally, we are developing improved trust management languages and systems that more explicitly support dynamic policies in service-oriented architectures, based on the semantic and performance experiences gained in the research's first phases.

Our focus will be twofold. First, we will explore adding trust-management language features that better support dynamic policies, based both on our experiences in the initial research and on the GIG's specific requirements.

Second, we will conduct experiments to measure the performance implications of incorporating the trust management layer in the various layers of such systems. A significant open research question is whether trust management is architecturally best implemented as a low-level operating system service, an application-layer service, or somewhere in between. **C**

References

1. M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," *Proc. 17th Symp. Security and Privacy*, IEEE CS Press, 1996, pp. 164-173.
2. M. Blaze, J. Feigenbaum, and M. Strauss, "Compliance Checking in the PolicyMaker Trust-Management System," *Proc. Financial Cryptography 98*, LNCS 1465, Springer, 1998, pp. 254-274.
3. S. Ioannidis et al., "Implementing a Distributed Firewall," *Proc. Computer and Communications Security* (CCS), 2000; www.itsec.gov.cn/webportal/download/2004_ccs-df.pdf.
4. S. Ioannidis et al., "Design and Implementation of Virtual Private Services," *Proc. IEEE Int'l Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (WETICE), Workshop on Enterprise Security, Special Session on Trust Management in Collaborative Global Computing, IEEE Press, 2003, pp. 269-275.

5. S. Ioannidis, "Security Policy Consistency and Distributed Evaluation in Heterogeneous Environments," doctoral dissertation, University of Pennsylvania, 2005.
6. M. Blaze, J. Ioannidis, and A.D. Keromytis, "Experience with the KeyNote Trust Management System: Applications and Future Directions," *Proc. 1st Int'l Conf. Trust Management*, 2003; http://nsl.cs.columbia.edu/projects/gridlock/newkeynote.pdf.
7. A. Wool, "A Quantitative Study of Firewall Configuration Errors," *Computer*, June 2004, pp. 62-67.
8. W.H. Winsborough and N. Li, "Safety in Automated Trust Negotiation," *Proc. IEEE Symp. Security & Privacy*, IEEE Press, 2004, pp. 147-160.
9. A.D. Keromytis et al., "The Strongman Architecture," *Proc. 3rdDARPA Information Survivability Conf. and Exposition* (DISCEX III), 2003; www1.cs.columbia.edu/~angelos/Papers/strongman.pdf.
10. A. Somayaji and S. Forrest, "Automated Response Using System-Call Delays," *Proc. 9th Usenix Security Symposium*, Usenix, 2000; www.csd.uoc.gr/~hy558/papers/somayaji-00automated.pdf.
11. M.E. Locasto et al., "Towards Collaborative Security and P2P Intrusion Detection," *Proc. 6th Ann. IEEE SMC Information Assurance Workshop* (IAW), IEEE Press, 2005, pp. 333-339.
12. M. Blaze, J. Ioannidis, and A.D. Keromytis, "Offline Micropayments without Trusted Hardware," *Proc. 5th Financial Cryptography (FC) Conf.*, 2001; www.crypto.com/papers/knpay.pdf.

Matt Blaze is an associate professor of Computer and Information Science at the University of Pennsylvania. His research focuses on cryptography and its applications, trust management, human scale security, secure systems design, and networking and distributed computing. He is particularly interested in security technology with bearing on public policy issues, including cryptography policy (key escrow), wiretapping and surveillance, and the security of electronic voting systems. Blaze received a PhD in computer science from Princeton University. Contact him at blaze@cis.upenn.edu.

Sampath Kannan is a professor in Computer and Information Science at the University of Pennsylvania. He is currently on leave and serving as the Director of the Computing and Communication Foundations Division at NSF. His research interests are in algorithms, program reliability, and security. Contact him at kannan@cis.upenn.edu.

Insup Lee is the Cecilia Fitler Moore Professor of Computer and Information Science at the University of Pennsylvania. His research interests include embedded and real-time systems, cyber-physical systems, medical device systems, model-based development, and quantitative trust management. Lee received a PhD in computer science from the University of Wisconsin, Madison. He is a Fellow of the IEEE. Contact him at lee@cis.upenn.edu.

Oleg Sokolsky is a research associate professor of Computer and Information Science at the University of Pennsylvania. His research interests include application of formal methods to model-based development and real-time systems, quantitative trust management, and runtime verification. Sokolsky a received a PhD in computer science from Stony Brook University. He is a member of the IEEE. Contact him at sokolsky@cis.upenn.edu.

Jonathan M. Smith is the Olga and Alberico Pompa Professor of Engineering and Applied Science at the University of Pennsylvania. He recently returned to Penn after serving as a program manager at DARPA/IPTO, where he initiated research programs in cognitive networking and distributed radio. His current research interests are in terabit-per-second networks and wireless network security. He is a Fellow of the IEEE. Contact him at jms@cis.upenn.edu.

Angelos D. Keromytis is an associate professor with the Department of Computer Science at Columbia University, and director of the Network Security Laboratory. His research interests revolve around systems and network security. Keromytis received a BS in computer science from the University of Crete, in Greece, and an MS and PhD from the Computer and Information Science Department, University of Pennsylvania. Contact him at angelos@cs.columbia.edu.

Wenke Lee is an associate professor in the School of Computer Science, College of Computing, the Georgia Institute of Technology. His research interests are in systems and network security, applied cryptography, and data mining. Contact him at wenke@cc.gatech.edu.