The Zodiac Policy Subsystem: a Policy-Based Management System for a High-Security MANET

Yuu-Heng Cheng*, Mariana Raykova†, Alex Poylisher*, Scott Alexander*, Martin Eiger* and Steve M. Bellovin†
*Telcordia Technologies
{yhcheng,sher,salex,mie}@research.telcordia.com
†Columbia University
{mariana,smb}@cs.columbia.edu

***Abstract*—Zodiac (Zero Outage Dynamic Intrinsically Assurable Communities) is an implementation of a high-security MANET, resistant to multiple types of attacks, including Byzantine faults. The Zodiac architecture poses a set of unique system security, performance, and usability requirements to its policy-based management system (PBMS). In this paper, we identify theses requirements, and present the design and implementation of the Zodiac Policy Subsystem (ZPS), which allows administrators to securely specify, distribute and evaluate network control and system security policies to customize Zodiac behaviors. ZPS uses the Keynote language for specifying all authorization policies with simple extension to support obligation policies.**

***Keywords*-policy-based management; MANET; computer network security;**

I. Introduction

With the development of increasingly complex systems, policies are widely adopted to allow users to customize and automate functional behaviors without the need for rebuilding or restarting a system. Existing policy-based management systems (PBMS) define customization for access control, obligated behavior for automation, etc., but do not address architecture security. Some existing policy systems are defined with centralized policy control where security is provided by the network transport authentication process, though some may utilize authentication information as part of the policy condition.

In this paper, we describe the Zodiac Policy Subsystem (ZPS) which supports the above policy types and operates within a distributed environment designed to heighten security in networks including MANETs.

In Section II, we describe the architecture of Zodiac and the requirements of a policy-based management for Zodiac. In Sections III, we describe the architecture and ongoing implementation of the Zodiac Policy Subsystem (ZPS). Section IV summarizes the problems addressed by the current design and indicates the directions of future work.

This material is based upon work supported by the Defense Advanced Research Projects Agency and Space and Naval Warfare Center, San Diego, under Contract No. N66001-08-C-2012.

The main contributions of this paper are:

- identification of the security, performance and usability requirements of PBMS for high-security MANETs;
- description of major architectural and design solutions to meet most requirements; and
- description of selected aspects of the implementation.

A full version of this paper is in [1].

II. The Zodiac High-security MANET

The DARPA Intrinsically Assurable MANET (IA-MANET) program aims to develop a "clean-slate" approach to mobile ad hoc networking emphasizing security. The approach must support network information integrity, availability, reliability, confidentiality, and safety. The network should enforce authentication and authorization of all actions with deny by default, resistance to Byzantine faults and insider threats, and define a selected set of functionality implemented in trusted hardware.

A. Zodiac

Zodiac (Zero Outage Dynamic Intrinsically Assurable Communities) is our solution that addresses the IAMANET requirements. Zodiac is based on a novel security and communication building block, the Dynamic Community of Interest (DCoI). A DCoI is a dynamic group of networked nodes running an instance of a distributed application or a supporting service. The Zodiac architecture is more fully described in [2]. However, a few concepts are important to understanding the relationship of ZPS to the rest of Zodiac.

DCoIs are implemented as virtual machine containers within a node. This design limits the effect of a successful attack within one DCoI as resources are allocated per container and processes have no direct access to the processes or files and data outside of their container.

The communication between Zodiac nodes within a DCoI is protected by its encryption, authentication and authorization mechanisms. A node ignores communications which it is not a member of. The encryption and memebership is managed by the Group and Cryptographic Services (GCS) subsystem in each DCoI.

B. Zodiac Policy Requirements

The Zodiac requirements lead to a unique set of requirements for ZPS. Computational (CPU, memory) and communication resources in Zodiac can be very limited. Thus, ZPS's processing footprint must be small, and policies themselves must be encodable in concise form. The ZPS user interface (UI) needs to allow administrators to create/update policies under stress. Apart from UI design, this necessitates a highly usable policy language.

In addressing Byzantine attacks, it is very undesirable to allow a management system on a compromised node to execute operations on a remote node. Each node must have the authority and responsibility to enforce policy to protect itself and the network.

In a permit-all, explicit-deny authorization scheme, only the known operations can be denied, so the set of permitted operations is unknown, which opens attack possibilities. Conversely, in a deny-all, explicit-permit scheme one can permit the exact minimal set of operations necessary. Additionally, unanticipated actions are denied, thus avoiding the issue that unanalyzed actions may result in security holes in the system. This is basic rationale for a deny-all scheme in a secure environment.

It is important to authenticate the author of a policy because only an explicit set of users are trusted to create policies. In policy distribution individual policies must not be corrupted and be up-to-date. The set of policies enforced in all containers of the same DCoI must be identical and self-consistent. This comprises the policy integrity requirements. Moreover, policy distribution should only occur within DCoIs in order to minimize the opportunity for exfiltration and must be timely to enable effective enforcement.

ZPS additionally relies upon other Zodiac subsystems for its own operation. For example, ZPS relies on GCS to provide keys for policy signing and verification.

III. ZPS Design and Implementation

We have designed ZPS to address the requirements above. In this section, we describe the design decisions in detail, including the types of policies supported, architecture, components, mechanisms for policy integrity protection and distribution, policy language modifications, and conflict analysis.

A. ZPS Architecture

ZPS is a fully decentralized policy system. Each ZPS instance evaluates its own set of policies and makes decisions to control the managed components, within a container. The only communication over the network is the policy contents. An alternative solution is to dispatch policy decisions from one or several locations over the network. For Zodiac, a fully decentralized solution is preferred as:

- policy contents do not change nearly as frequently as policy decision results,
- low network latency is far less critical for policy contents as it is for policy decisions, and
- it is easier to support need-to-know policy evaluation and Byzantine fault tolerance.

Policy content is, of course, sensitive on its own. Knowing the access control policies, for example, may reveal the usage of a DCoI. Because of this sensitivity, we not only protect policies in transit, but also ensure that even if our protections fail, the set of policies to which an attacker has access is minimized.

As described above, each DCoI is instantiated inside of a container. ZPS is duplicated in each container as in Figure 1. This design reduces the ability of an attacker both to inspect policies from other containers and to affect the behavior of containers other than the one to which she has gained control.

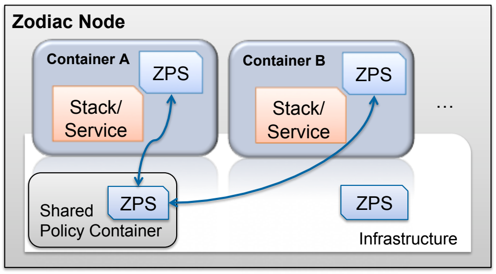

Figure 1: Components within a Zodiac node

The ZPS instances that reside in a container, the shared policy container, and Infrastructure evalutates policies for the DCOI, shared DCOI resources, and the node respectively.

Communications between ZPS instances are constrained to occur along the paths created and enforced by Infrastructure. This helps reduce the opportunity for unauthorized access to policy information.

B. ZPS Components

Figure 2 shows the ZPS components and their relationships with other Zodiac subsystems within a container. The *policy management components* (collectively is the ZPS) contain the logic to control the *policy-managed components* (Zodiac stack and services). Each policy-managed components implements the PEP (policy enforcement point) that enforces the operations imparted by the ZPS. These operations were carefully determined in the Countermeasure Characterizations (CMC) analysis.

The PDP manages the Zodiac system by reacting to requests and events from other subsystems, evaluating matching policies and directly invoking PEP operations. The requests include authorization and configuration requests. Services that started later than the ZPS can acquire its settings via a configuration request.

Events are defined to provide situation awareness for the Zodiac system. They serve as feedback information to the

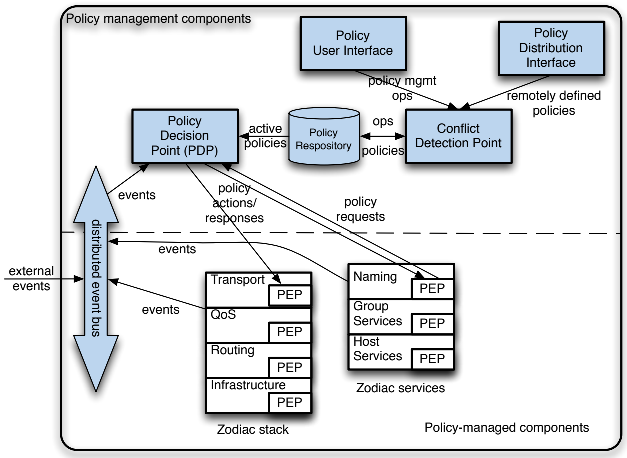

Figure 2: ZPS components within a container

PDP. The usage of the events needs to be carefully designed. Carelessly designed events can cause system instability. For example, if an event causes the system to publish the same event, the system will be trapped in an endless loop. Currently ZPS only recognizes the event that identifies the current information operations conditions (INFOCON). We specify obligation policies to apply different communication mechanism for the DCoI based on different INFOCON level.

Policies can be input into ZPS from either the *Policy User Interface* or *Policy Distribution Interface*. All policies are stored in the Policy Repository after the Conflict Detection Point ensures that the policies are conflict-free.

C. ZPS Policies

ZPS supports both authorization and obligation policies. Because of the default deny-all authorization requirement, only positive authorization is supported. Policies are further categorized into *DCoI policies*, *shared resource policies*, and *node policies*. The category is predetermined and orthogonal to the authorization/obligation categorization. Besides the information assurance benefits, the categorization also helps us determine the conflict domains of the different policies.

All policies are signed directly or indirectly by a trusted entity to ensure policy integrity and allow traceability. Only the policies issued by a trusted entity are accepted by ZPS. We assume that the trusted entity creates policies that follow the operational intent. Some operator errors are prevented by policy conflict detection. The validity of both the signature and the signer's privilege is verified upon each policy request since trusted entities may change over time (due to change of roles for adversary action, e.g., node capture). The former check relies on the existing encoding algorithms, and the latter check is implemented using a certificate revocation list. ZPS relies on GCS to perform these operations as part of its responsibility for all the cryptographic and key management in Zodiac.

For a system with a significant number of policies, verifying the policy signature for each policy upon each request may not be efficient. The policy evaluation performance may be optimized by removing policies with invalid signatures before storing the policy or upon revoking a certificate.

D. Policy Distribution

After initial network deployment, administrators may need to create a new policy or modify/remove an existing one. A modified policy set then needs to be distributed to all nodes belonging to the same DCoI. Since MANET connectivity can be unstable, the distribution mechanism should be able to handle intermittent connections.

When a node is temporarily out of reach, that node should obtain the modified policy set when connectivity is resumed. ZPS uses the reliable transport services provided by Zodiac to ensure that a policy set is correctly delivered once it is determined that distribution is required.

Since ZPS manages Zodiac based on the content of the policies, the policy set for a DCoI needs to be consistently duplicated among all DCoI members. Thus, policy distribution needs to be integrated with the group membership maintenance, i.e., GCS. When a node joins a DCoI, GCS notifies ZPS to distribute the policy set to the new node after a successful security protocol exchange. When a node leaves the DCoI, the ZPS in that node will remove all policies as the container is also destroyed.

During the lifetime of a DCoI, policy updates are distributed via multicast to all members with simple synchronization mechanisms for tolerating unstable network connectivity. In a future implementation, we also plan to re-key when policy distribution occurs. In order to get the new key, a node must have a copy of the current policy set. Currently, Zodiac statically positions policies with the correct signatures.

E. ZPS Policy Language

Three existing policy languages that we considered are Keynote, Ponder1 and XACML. Although all provided the basic functionality for the policy subsystem modulo some extensions, we considered Keynote the best fit, based on the consideration of readability, policy size, code size, and security features.

Keynote offers an evaluation environment where the default state is denial and policies specify the authorizations and actions allowed in the system. We extended Keynote to allow specification of obligation policies. Based on the occurrence of particular events, policy evaluation returns corresponding multi-dimensional vectors containing specifications for configuration parameter changes or actions that need to be executed by PEPs.

Though the Keynote language supports any combination of conditions, a potential security concern is using the negate operation (syntactically, '!') in conditions. This is because

ZPS policies are permissive, i.e., specify only the outcome or the operations permitted. Unexpected outcomes and declined operations are simply not mentioned in the policies. For example, the condition for *track* in the policy of Listing 1 is permitting "not purple" to join. In a system with only "blue" and "purple" tracks, this makes no difference. However, when a new track, say "red", is introduced, the policy implicitly permits both the "red" and "blue" tracks to join the DCOI, which is potentially undesirable.

Listing 1: Example of negate condition

```
...
Conditions: (track != "purple") &&
            (request == "join") -> "true";
...
```

Though we make negative conditions in Keynote a syntax error, from a usability perspective, a better UI would not allow this altogether.

F. Policy Conflicts

The first important point when considering policy conflicts is that we are dealing only with positive authorization policies. This alone eliminates the conflicts between negative and positive authorization policies, but does not resolve all possible conflicts.

The simplest conflict type is having different policy decisions for the same attribute in different policies (syntax-level conflict). This can result from policy definitions/updates coming from different sources.

Another conflict type that goes beyond the syntax is the use of overlapping attribute domains, especially for authorization policies. For example, two policies are given for a company which has two units and several departments in each unit. The first policy states that unit A is allowed to access only floors in building B, and the second states that department Y is allowed to access floor 3 of any building. Department Y is in unit A. This conflict type can be detected when the PBMS has the knowledge of the organizational structure and building/floor containment (attribute relationships).

The policies in ZPS specify actions allowed in the managed system under certain conditions or the configuration parameters. However, some combinations of conditions and/or parameter values may not be allowed because they do not conform to the desired behavior of Zodiac components. For example, setting a flooding degree for routing to 7 for a DCoI that has only 6 potential members. We call these incompatible combinations of policy interdependencies. They represent a conflict both when they exist in the same policy and when defined across policies with overlapping domains.

IV. Conclusion and future work

A high-security MANET poses several new challenges for the design of its PBMS, some of which have been addressed in the presented ZPS design as summarized next. It is likely that some solutions that fit the security-related requirements of ZPS are applicable to other network environments, e.g. wireline networks.

- *Authentication and Authorization*: ZPS uses node certificates to build a chain of trust to acquire credentials and authorizations (e.g., DCoI membership) and to provide an identity to its peers for policy enforcement.
- *Integrity*: Policy integrity in transit is enforced via signatures from authorized entities. PDP decision integrity is enforced by considering only authenticated policies and by placement of PDPs locally rather than across the network.
- *Confidentiality*: Confidentiality requires that policy content is known only by the creator and the intended receiver(s). Besides traffic encryption for policy distribution, each DCoI container has its own PBMS instance to process policy information.
- *Availability*: MANET link layer communication can be highly unreliable. As policies need to be distributed to all nodes in a DCoI, ZPS uses appropriate reliable protocols (unicast or multicast). Though ZPS denies all operations by default, bootstrap processes are permitted to ensure the ability to setup initial communication between nodes.

Deconfliction for DCoI policies that concern shared resources within in a node and the security requirements for information exfiltration is an ongoing effort. Additional work is planned on the automated suggestion of potential conflict resolutions to policy creators, an intuitive UI, and further updates to the Keynote language and its PDP.

In the near future, we plan to determine how to leverage the Zodiac secure transport mechanisms to distribute policies without creating circular dependencies where transport depends on a policy about what policies can be distributed.

References

[1] Y.-H. Cheng, S. Alexander, A. Poylisher, M. Raykova, and S. M. Bellovin, "The Zodiac Policy Subsystem: a Policy-Based Management System for a High-Security MANET," Columbia University - Computer Science Department, Tech. Rep. cucs-023-09, May 2009.

[2] S. Alexander, B. DeCleene, J. Rogers, and P. Sholander, "Requirements and architectures for intrinsically assurable mobile ad hoc networks," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, 2008.