Can I Let You In?
On Event-Centric Context-Aware Authorization

Philip Attfield*, Paul Chenard*, Marta Piekarska†, Julia Narvaez*, Mike Hendrick* and Simon Curry*

*Sequitur Labs
33404 Redmond Fall City Rd. SE,
Fall City, WA, 98024, USA,
E-mail: <name>.<last_name>@seqlabs.com

†Technische Universität Berlin
Security In Telecommunications,
Ernst-Reuter-Platz 7, 10587 Berlin, Germany,
E-mail: marta@sec.t-labs.tu-berlin.de

Abstract*—Current mechanisms for control and protection of computing resources were conceived decades ago. At that time constraints on power management, connectivity and the types of computing assets were far simpler. Today's mobile and distributed information systems are vulnerable to much wider and sophisticated threats. Thus, they require more flexible, extensive and powerful policy-based protection. This paper contributes a framework for policy-based authorization and illustrates its implementation. Details describing the architecture, methodology and tool flow for reliable synthesis of custom policy-based authorization are presented. The hypothesis is that access control applicable to a variety of devices should be event-centric and context-driven. The integrity and security of the authorization systems as well as the end-to-end trust that is guaranteed in the process used to create them are discussed. The applicability of the solution and its ability to mitigate the threats are discussed. A wide range of systems from simple to complex, including the emerging Internet of Things is covered. *(Abstract)

Keywords–Mobile Operating Systems, Access Control, Mobile Device Management, Internet of Things.

I. INTRODUCTION

Users are accustomed to a world full of choices and possibilities. Thus they are often surprised and frustrated by usage restrictions and constraints that computer and communication devices impose upon them [1]. The way that we manage devices is by policies/rules, that define access to, and the use of, the resources made available. There is a growing need, even an expectation, that these should cover the variety of ways in which people and organizations want to use the devices instead of reflecting the engineering limitations [2].

With good *authentication*, the system can validate the *bona fides* of the current operator of the device. However it cannot offer control over the way the assets available on those devices are being used, either deliberately or inadvertently. These include hardware components like the camera or microphone on a mobile phone, and software elements such as files, applications or network resource access. This paper elaborates on the problem of policy for the *authorization* of requests that arise in the everyday use of the device. It also proposes an implementation that solves that issue.

In a world where we can software-define almost anything, policies must be programmable. It is not important whether the computing asset is a mobile device, an Internet of Things (IoT) element, a Personal Computer (PC), a server, a Virtual Machine (VM) or a newly instantiated virtual network function (VN). A real-time, dynamic, contextually-aware policy model with a definable policy enforcement action (not just "allowed" or "not allowed") is more effective and more useful than the traditional static approach, as will be shown. The solution presented creates a hierarchy of policies, where multiple parties can author rules, which are then prioritized. This paper contributes to the field in several ways:

- It identifies lack of authorization mechanisms suited for mobile and distributed information systems.
- It presents a framework and an architecture that has general application to the management of many device types. This solution will find wide application with spread of Internet of Things approach.
- It shows how much more extensive and complex systems can be subject to powerful and flexible policy control of arbitrary granularity.

The rest of the paper is organized as follows. Section II reviews work that precedes our research. Section III presents an overview of the Framework architecture, explaining the general approach that was taken. Section IV discusses the components of the event centric portion of the Framework, while Section V discusses policy and policy administration for it. The work is summarized in Section VI.

II. RELATED WORK

On a more abstract level, the work done by van Thanh et al., presents the concept of Device Management Service (DMS) [3]. This is a "virtual terminal" that can be used to manage multiple phones, both mobile and stationary. Unlike the Universal Personal Telecommunications (UPT), DMS allows for parallel registration, collaboration and work. This work from 2001 can be seen as the precursor of actual Mobile Device Management (MDM) solutions.

Another attempt to solve the problem of device management is presented in [4]. Here, the authors present a secure device management framework that aims to securely deliver services to user devices, manage credentials and interact with service providers. However, it mostly focuses on access control within a single unit, usually for the security purposes and secret handling.

Mei et al. discuss a design of a remote device management framework that is crafted for personal devices in [5]. This tool allows access to information about operational status, maintenance functionalities and potential security issues of the device. The advantage of their work is that it includes an open

 ISBN: 978-1-61208-451-0

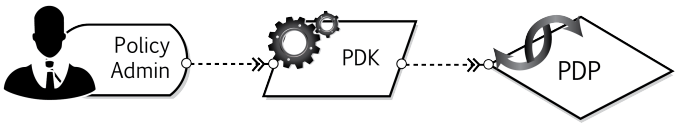

Figure 1. Framework Overview. The PDK captures the user's design intent at Level 1, and generates an instance of the PDP at Level 0.

implementation based on the SyncML Device Management (DM) specification.

Song et al. identify the problem of managing and controlling machine-to-machine (M2M) devices in [6]. This mostly state-of the-art work shows several architectures for M2M devices that are subject to standardization, with the focus on key functionalities that are used to manage and control them. On the other hand, Internet Protocol (IP) Multimedia Subsystem device management issues are presented in [7]. The paper also suggests an architecture for management by separating functions and providing service provisioning and tracing functionalities.

On the subject of security level analysis, Landman provides an in-depth analysis of the types and nature of threats to an organization from the use of smartphones as well as controls, available security software and tools, in [8]. He also shows the state of corporate security programs.

Another extensive work, where the authors suggest a new modeling methodology and present threats that MDM systems face is presented in [9]. They achieve this by an analysis of the agents, assets and ad-verse actions.

III. Framework

This section provides details of a novel, context-aware policy management solution that is applicable to a wide variety of systems, as will be shown. On a high level it comprises two layers:

a) Level 0:: Includes the operational components that communicate together to implement the policy authorization process in a live system. This level also includes compilers[1] that synthesize policy specifications in the Policy Object Language (POL) language into executable Policy Decision Point (PDP) instances for use in a live system.

b) Level 1:: Contains the Policy Design Kit (PDK), a GUI-based policy design environment that simplifies the authoring of complex policy sets. It also manages the end-to-end process of creating an instance of a Level 0 framework, ensuring its integrity, authenticity and correctness.

The data flow from the user's design intent, through Level 1, and Level 0 to produce an instance of the framework is shown in Figure 1.

IV. Framework Level 0 Architecture

As depicted in Figure 2, the operational architecture of the framework has four components: Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Information Point (PIP) and Fabric.

[1]The discussion of POL compilers is deferred to the section on Level 1.

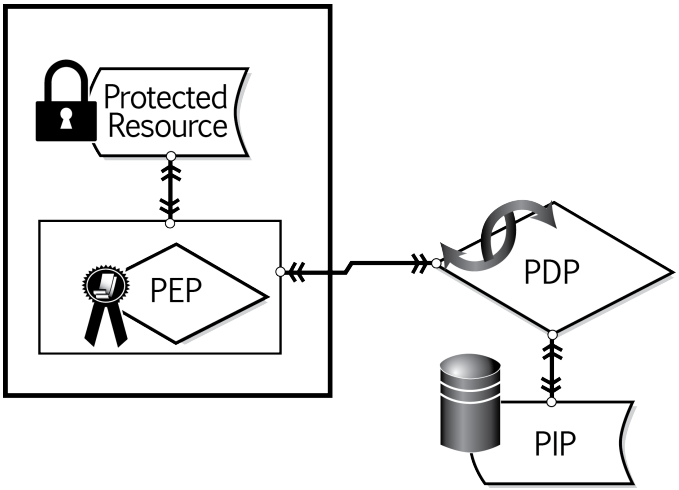

Figure 2. Framework Level 0 Architecture. The Level 0 architecture consists of four components: Policy Enforcement Point, Policy Decision Point, Policy Information Point and the Fabric (denoted by the double headed arrow).

A. Operational Elements

A PEP is an agent located on a device, such as a mobile phone. It monitors events on the device that represent requests to access resources under policy control. For each such request, the PEP creates a *query* message that contains details of the request and of the state of the device at the time of the request and transmits it to a PDP; it then waits for a response. When the PEP receives a response, it enforces the decision contained therein.

A PDP is a server that may be located remotely or, sometimes, co-located with the PEP in a device. The PDP listens for queries from PEPs and, using the data within the query and policies specific to the instance of the PDP, transmits a response to the requesting PEP.

A PIP is a data source, external to the PDP, containing information needed to evaluate policies. PIPs are typically directory services. The PDP has special features that exploit access to PIPs efficiently.

Depending upon the nature and disposition of the PEPs and PDPs and the application for which they provide an authorization service, PEP queries and PDP responses may be transmitted through the Fabric that may be any of a wide variety of external media. As will be detailed further, these may be a complex as User Datagram Protocol (UDP) over Universal Mobile Telecommunications System (UMTS) [10] or as simple as shared memory. The Fabric is depicted as a double headed arrow in Figure 2.

B. Structure of a query

From an abstract point of view, *queries* and *responses* consist of an array of elements and a fabric dependent header. Each array element represents a PEP state attribute or data pertinent to the request. For responses, each array element contains an attribute of the response, such as a verdict or information qualifying the verdict, for example a stipulation. Array items are identified by their indices, as agreed in a dictionary shared by the PEP and the PDP. The array is referred

 ISBN: 978-1-61208-451-0

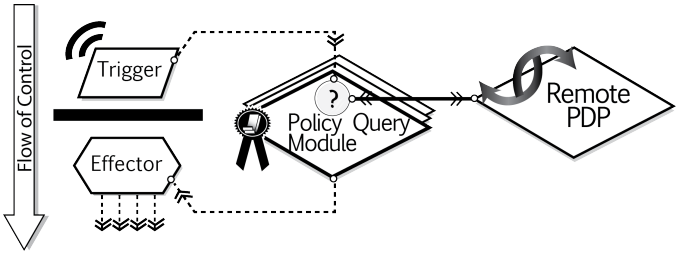

Figure 3. Policy enforcement. The Policy Enforcement Points (PEPs) are remote agents located on the managed device that are triggered by access requests. PEPs query the PDP for authorization and enforce the response.

to as the *dynamic array*, and each of its elements is referred to a *dynamic*.

C. Policy Enforcement Point (PEP)

Depending on the type of resource, the PEP can be located at the driver level or implemented as an application. The PEP consists of a Trigger, a Policy Query Module (PQM) and an Effector, as depicted in Figure 3.

The Trigger, interrupting the normal flow of an event, gathers the relevant dynamic content and sends it to the PQM.

The purpose of the PQM is to create queries and enforce the intent of responses on the operation of the device. Depending on the Trigger and any included dynamic attributes it may look for additional environmental variables to form a query. It then sends the query to a designated PDP (local or remote). Finally, it obtains a policy decision and provides the response to the appropriate Effector for that Trigger.

The process followed by the PQM varies according to the nature of the PEP. This includes consideration of the state of the device (online or offline), whether or not the device policy cache may be used and which PDPs may be consulted for a policy response.

The Effector enforces verdicts returned by the PQM. It provides a meaningful control path for each possible verdict/stipulation combination and ensures a sensible user experience for all outcomes.

Caching is shared by all PEPs on a given device because the wireless network can present an inconsistent and sporadic connection to all back-end services. Policies can direct caching of responses for a specified period of time. This reduces fabric bandwidth demand on the network and allows for near-instant response (to queries that match the same conditions) without recourse to an external connection. This is useful for devices that have a high frequency request cycle, such as a camera that may query as frequently as 60 times a second.

In the event that there is no cached decision and the PDP is not reachable, a simple set of decisions is stored on the device to provide a fail-safe response for all PEPs. If the network does not allow for a real-time decision to be received, the response is equivalent to that provided by traditional MDM solutions.

Policy decisions may include stipulations that require logging. There is a common service which collects log data and delivers it efficiently. For example, if multiple control points are logging and the network is unavailable, the common service will hold the data and deliver it as a bulk payload once the connection is restored. This relieves individual control points from tracking and performing these functions.

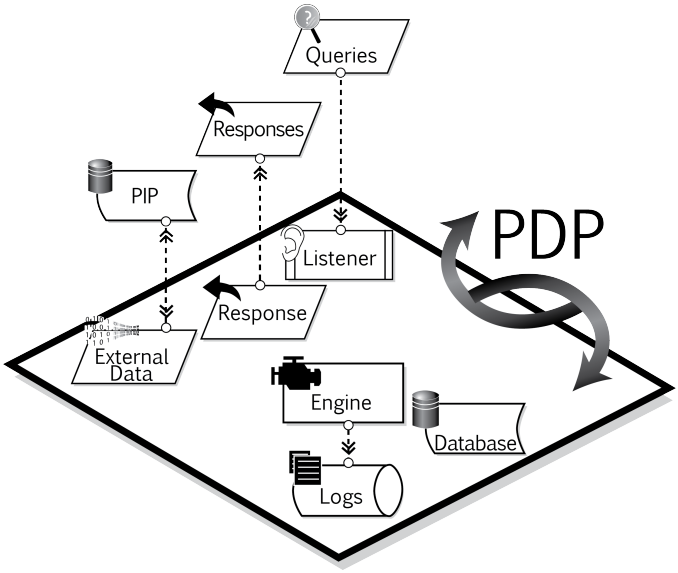

Figure 4. Policy Decision Point. The PDP is a server that issues that issues responses to queries from PEPs. Its responses are based on PDP-specific policies. PEPs and PDPs communicate through a two-way channel.

The heartbeat function maintains periodic communication between the PEP and the PDP confirming PEP presence and state. The PDP may respond to heartbeats with control messages.

The installation of PEPs to devices depends on the nature of the implementation and the protected resource. PEPs protecting hardware resources are implemented at the hardware resource driver level. For example, a camera resource needs to be implemented in the camera driver. PEPs protecting data files (for example), may be implemented as part of a file system driver or as part of an application.

The PEP ensures a sensible outcome for the end user or application when an access attempt is denied. Devices do not appear broken; applications do not crash.

D. Policy Decision Point (PDP)

A PDP is a server that issues responses to queries received from PEPs. The responses are based on policies that are specific to the PDP and that make use of the dynamic values in the query. Refer to Figure 4 for more detail.

1) Characteristics of a PDP: In order to ensure availability, a PDP always returns a response to a PEP even if no relevant policy is found. A PDP is defined as *permissive* or *restrictive* depending on whether it sends an ”allow” or ”deny”. A PDP can handle large collections of policies (thousands or more) without significant performance degradation. The PDP is stateless. This improves performance and simplifies interactions with PEPs and management of PDP farms. For audit and forensic analysis purposes, a PDP generates a log of all query transactions. A PDP may respond to PEP heartbeat signals with commands that affect the state of the PEP or host device. One of the operations that may be requested is maintenance for a PEP instance.

 ISBN: 978-1-61208-451-0

The decision engine of the PDP interfaces at the level of abstract query objects. This allows the use of various harnesses to adapt the PDP to a variety communication fabrics. Furthermore, this provides for the creation of systems with multiple, concurrent PDP instances for higher bandwidth needs.

2) Security Design of a PDP: PDP architecture minimizes the opportunities for malicious intervention. The code of the PDP itself, as well as the representation of policies is attestable (signed and authenticated) through a chain of trust. The process of creating PDPs and their policies and delivering them can be made verifiably secure.

The PDP is synthesized by a compiler which directly generates its executable, including the embedding harness, from policy *specifications* in the proprietary POL language. The PDP offers no programming API or facility for source code tampering or modification and the executable may be signed.

To limit the attack surface, the PDP has only one I/O channel on which encrypted packets are received (queries) and transmitted (responses). It has an output channel on which logging information is emitted. Moreover, in order to prevent unauthorized access and modification of the policy database and rules, the database is password-protected. Only the compiled PDP and the data-base server have the key, which is generated at compile time. Essential database fields may also be encrypted.

Lastly, when it comes to network fabric, queries and responses are transmitted using a protocol that is resilient to man-in-the-middle attacks and spoofing. The PDP drops incoming packets as soon as protocol discrepancies are detected.

E. Policy Information Point (PIP)

Corporate data, particularly personnel data, is often stored in active directories and databases which deliver fast data retrieval and consistency. Policy formulation often requires precisely this kind of data. The PDP has been designed to connect to directories. The POL language provides directory specification and search functionality to make the data available for policy evaluation by the PDP.

V. Policy and Policy Management

From the forgoing description it is clear that the behavior of devices governed by the system is dependent upon the policies embedded in the PDP. A policy is a rule that dictates what actions should be taken for a particular event under a given set of conditions. Individual policies are gathered together into policy sets which together address all of the events and circumstances of interest to the policy authors.

A. Characteristics of Policy Management

The objectives of policy management include the following:

- The method of expressing policies is rich enough to express policy author intent under a wide range of circumstances, some of which cannot be foreseen. It is succinct so that the resulting policy sets remain manageable in size and complexity.
- Policies are written only by those who have the required authority for the resource being governed.
- Policy sets created by many authors are combined with a clear order of precedence, with any conflicts or logical problems detected and addressed.
- Information referenced by the policies to arrive at decisions, in PIPs or otherwise, are only writable by properly authorized authors.
- Compilation and deployment of PDPs employs a controlled process and is done by authorized individuals.

B. Chain of Trust

A result of a stringent implementation is a chain of trust for the policy data, beginning with the creation of policy and extending through to the deployment of PDP. This chain of trust ensures that the integrity of policy intent is maintained.

The establishment of the chain of trust requires two types of policies: Level 0 policies which specify how PDPs respond to device requests, and Level 1 policies which govern access control for data used to create Level 0 policies. These also secure custody of the data.

1) Policy Specification, Level 0: Level 0 policies are specified in a formal policy language, the POL. POL is terse and declarative to facilitate synthesis and static verification of policy sets. It follows a pattern of: Subject, Agent, Object, Action and Environment (SAOAE). Each component has a corresponding clause in the model which consists of an arbitrary expression that can reference dynamic data from the query, static data from a PIP or data from the policy set itself. The clauses organize policy considerations along the following lines:

- Subject: the identity of the entity making the request, e.g. the user.
- Agent: the means by which the request will be carried out, e.g. the program that will make the access.
- Object: the elements and items affected by the request and being acted upon by the Agent.
- Action: the specific function that the Agent applies to the Object.
- Environment: information in the request that would be observable at the PEP but independent of any given event, such as time or location.

The POL language provides a mapping feature that allows query and PIP data to be tagged along arbitrary lines. Tags can be tagged themselves, forming chains which begin with query or PIP data, allowing large numbers of specific data elements to be aggregated into manageable categories for expressing policy intent. Support is provided for regular expression constructs, simple geolocation constructs and time intervals. For example, a tag chain could be constructed to map an identifier from a mobile device into an employee number, and then into a role or a location or an authorization level. Clauses in SAOAE statements can reference tag chains in their expressions. The number of tags and length of tag chains is not dictated by the language. It can be used to construct both policies with a wide scope and general application, or policies for specific situations with very fine granularity.

 ISBN: 978-1-61208-451-0

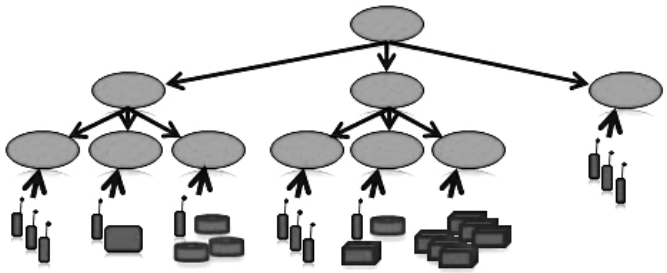

Figure 5. Policy Ownership Tree. Policy stakeholders are arranged in a hierarchy, each responsible for their own domain and allowed to delegate authority to those beneath them.

2) Policy Application, Hierarchy and Delegation: Designated authorities are permitted to write the policies for those areas within their domain. As shown in Figure 5, Policy stakeholders are arranged in a hierarchy and are allowed to delegate authority to those beneath them. This permits higher-ranking authorities to reserve for themselves the rights they need and to delegate policy decisions to others as appropriate. Policies may be marked as default, to permit policy set closure if no lower-ranking policy authors provide an applicable policy.

This policy ownership tree model can be applied in various ways to simplify the management of complex policy sets. For example, one approach might use an organization chart to map the hierarchical tree such that levels of policy authority correspond to levels of organizational authority. Another example might use a high-ranking policy set to specify some coarse-grained generic default policies and one or more lower-ranking policy sets to create categories of specific exceptions to the generic rules.

3) Administrative Policies, Level 1: The POL language and its constructs define Level 0 policies which address the manner in which the system responds to requests from devices. In order to implement the chain of trust as outlined above, another layer of administrative, or Level 1 policies is required to govern the authentication and authorization of stakeholders as they perform their duties within the system. These duties can consist of policy and data entry, configuring policy information points, compiling and deploying PDPs, introspection and debugging of policy sets, and administration and management of the administrative policies themselves. Administrative policies are defined and enforced in the PDK.

C. Policy Design Kit (PDK)

The PDK is a general purpose authentication and authorization platform that provides controlled access to the data and tools and a policy life-cycle framework. Its work is presented in Figure 6. After authenticating to the PDK, users receive a fine-grained set of abilities dictating allow/deny access to various resources. Data objects in the PDK have a strict chain of ownership back to a user. These abilities combine with the object ownership to enforce Level 1 policy. Relationships between users are established in the PDK to enforce rules for sharing data, and to establish relative positions in the policy ownership tree.

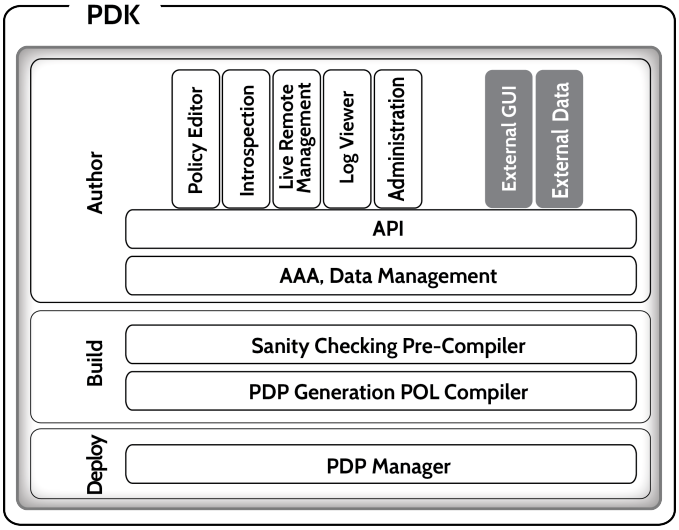

Figure 6. Policy Design Kit. The PDK is a policy life-cycle management framework with a set of tools for the policy administrator to conduct policy authoring, introspection of policy sets, PDP generation, sanity checking and deployment.

1) Policy Capture, Level 0: The PDK provides policy models that allow users to express their Level 0 policy intent at a level of abstraction suitable for a given application. For example, a policy model that governs a resource according to a time interval and a location might present an interface that captures just the interval and the location. No knowledge of POL is required to use the models. Data to populate a policy model's tags is also captured by the model at an appropriate level of abstraction. The PDK provides a number of policy models; however, a given user only has access to those for which they are authorized by Level 1 policy.

2) PIPs and Data Sharing: While PIPs usually contain common organization data, often additional data is required for policy authoring. For this reason, the PDK provides models that allow users to capture extra data and explicitly share the information with other selected policy authors by using the PDK's authorization mechanism. The shared data is available to a second user's policy models but cannot be viewed or changed. As an example, an employee in HR may be responsible for maintaining employee data, while an IT employee may dictate network policies using a role-based scheme bound to that employee data.

3) Configuration: The PDK provides a set of user administration functions that allow users authorized by Level 1 policy to manage other users in the PDK. These functions include the typical user life-cycle operations, as well as the assignment of abilities to users which grant them access to various functions, policy models and shared data in the PDK. Users and their policy sets can also be assigned to their positions in the policy ownership tree.

4) PDP Generation, Sanity Checking and Deployment: The PDK provides tools for the generation and compilation of PDPs from policy set trees. The first step in the process is to synthesize POL code from the policy model and captured data. The POL code is then compiled to produce C++ code and any associated database. The compiler carries out simple semantic

 ISBN: 978-1-61208-451-0

checks as well as a sanity check, which verifies that the policy set possesses certain properties. For example, it verifies that there is no ambiguity where two policies might apply to the same query, and it verifies that there is a set of query data capable of activating each policy in the set. The last step is to compile the C++ code to produce the executable PDP. The entire flow can be executed stepwise or automatically as a single step.

The PDK provides functionality to define server elements, to transfer the executable PDP to them and to set them running. Server elements can be designated as database servers, PDP servers or both. This allows flexibility in the number of PDPs deployed, and in load balancing the number of executing PDPs to the number of corresponding database servers.

D. Debugging and Introspection

The executable PDP produced by the compiler has minimal I/O. I/O is limited to receiving requests, emitting decisions, and providing a log file. The log file provides an indication of which policy was used to arrive at a decision; however, it doesn't provide any information on how that policy was selected. As policy set complexity increases, so does the need for tools to analyze and debug them, and examine them forensically. To address this, the PDK provides features for log file analysis and management as well as policy set introspection.

Authorized users of the PDK can collect PDP log files from various servers, combine and analyze them. The log entries for any given device may be distributed amongst a number of PDPs in a farm. Combining them allows the entire history for any one device to be examined. The combined log is elaborated to allow any sequence of events to be found quickly.

Introspection refers to the activity of studying policy set behavior by simulating the PDP look-up process, while displaying all intermediate results. For a given simulated request, each of the policies considered are displayed in rank order, along with the values of all policy clauses. Intermediate terms and tag expressions in the clauses are also evaluated and displayed, providing detailed information showing why any given policy was selected, rejected or not evaluated. This capability allows authors to evaluate different scenarios against policy sets and groups of policy sets to determine if desired outcomes are being produced.

VI. Conclusions

The paper postulates that as computing assets become commonplace, the need to manage data on an event-centric and context-aware basis is becoming urgent. Mobile Device Management has been adopted as the industry standard for mobile security. However, current MDM solutions are hard to manage, are inflexible and apply the rule "define once, run always". When it comes to mobile devices, Personal Computers, servers, health trackers, Virtual Machines or Internet of Thing devices, more power and flexibility is needed.

This paper presents a novel, context-driven policy definition and enforcement framework which addresses these shortcomings. Instead of deciding whether access is granted to a resource at configuration time, this solution takes into account the state of the system, time, location and any other definable factors at the time of the event. The solution addresses the entire policy life-cycle: formulation, management, verification, debug and analysis of policy behavior at time of definition as well as execution, policy server generation and secure delivery of the implementation.

The solution can be applied to many fields, not only those with a focus on the enterprise. The Framework elements that are common to all foreseeable applications are discussed: the Policy Design Kit, the Policy Decision Point, the Policy Enforcement Point and the Policy Information Point. The application of the solution to MDM is presented. It shows how extensive and complex systems can be subject to powerful and flexible policy control with appropriate granularity, while still remaining manageable. Evaluation is based on comparison with existing solutions.

The context-aware policy framework provides generalized resource access control based on a wide set of conditions. The wide range of factors that may be considered in policy, the unrestricted specification of policy, the hierarchical and delegation features, the real-time assessment of events and conditions, and applicability to other domains, such as the Internet of Things, are unavailable in current solutions. Most importantly, the combination of these features represents a significant step forward in the field of access control.

References

[1] B. J. Myers, "Student perceptions of computer anxiety: The relationship of computer attitude, computer experience, age, gender, and socioeconomic status," Ph.D. dissertation, Vermillion, SD, USA, 2006, aAI3255100.

[2] S. M. AlHaj, "Context-aware policy management platform: Based multi-agent systems mas," in *Proceedings of the 2011 Developments in E-systems Engineering*, ser. DESE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 490–495. [Online]. Available: http://dx.doi.org/10.1109/DeSE.2011.89

[3] D. V. Thanh, T. Jonvik, E. Vanem, D. van Tran, and J. Audestad, "The device management service," in *Intelligent Network Workshop, 2001 IEEE*, May 2001, pp. 199–211.

[4] A. Leung and C. Mitchell, "A device management framework for secure ubiquitous service delivery," in *Information Assurance and Security, 2008. ISIAS '08. Fourth International Conference on*, Sept 2008, pp. 267–274.

[5] H. Mei and J. Lukkien, "A remote personal device management framework based on syncml dm specifications," in *Proceedings of the 6th International Conference on Mobile Data Management*, ser. MDM '05. New York, NY, USA: ACM, 2005, pp. 185–191. [Online]. Available: http://doi.acm.org/10.1145/1071246.1071275

[6] J. Song, A. Kunz, M. Schmidt, and P. Szczytowski, "Connecting and managing m2m devices in the future internet," *Mob. Netw. Appl.*, vol. 19, no. 1, pp. 4–17, Feb. 2014. [Online]. Available: http://dx.doi.org/10.1007/s11036-013-0480-9

[7] J. Ma, J. Liao, and X. Zhu, "Device management in the ims," *J. Netw. Syst. Manage.*, vol. 16, no. 1, pp. 46–62, Mar. 2008. [Online]. Available: http://dx.doi.org/10.1007/s10922-007-9092-7

[8] M. Landman, "Managing smart phone security risks," in *2010 Information Security Curriculum Development Conference*, ser. InfoSecCD '10. New York, NY, USA: ACM, 2010, pp. 145–155. [Online]. Available: http://doi.acm.org/10.1145/1940941.1940971

[9] K. Rhee, D. Won, S.-W. Jang, S. Chae, and S. Park, "Threat modeling of a mobile device management system for secure smart work," *Electronic Commerce Research*, vol. 13, no. 3, pp. 243–256, September 2013. [Online]. Available: http://dx.doi.org/10.1007/s10660-013-9121-4

[10] M. Sauter, *Communication Systems for the Mobile Information Society*. John Wiley, 2006.

 ISBN: 978-1-61208-451-0