Using Hierarchal Change Mining to Manage Network Security Policy Evolution

Gabriel A. Weaver, Nick Foti, Sergey Bratus, Dan Rockmore, Sean W. Smith
Deparment of Computer Science Dartmouth College, Hanover, New Hampshire USA

Abstract Managing the security of complex cloud and networked computing environments requires crafting *security policy*—ranging from natural-language text to highly-structured configuration rules, sometimes multi-layered—specifying correct system behavior in an adversarial environment. Since environments change and evolve, managing security requires managing *evolution* of policies, which adds another layer, the change log. However, evolution increases complexity, and the more complex a policy, the harder it is to manage and update, and the more prone it is to be incorrect.

This paper proposes *hierarchical change mining*, drawing upon the tools of software engineering and data mining, to help practitioners introduce fewer errors when they update policy. We discuss our approach and initial findings based on two longitudinal real-world datasets: low-level router configurations from Dartmouth College and high-level *Public Key Infrastructure (PKI)* certificate policies from the *International Grid Trust Federation (IGTF)*.

1 Introduction

The paper considers the *security-policy evolution problem*. Security policies must evolve to be useful: if practitioners don't update their policy, their networks are vulnerable to new threats. However, updating policy creates management challenges: due to complexity, practitioners may introduce error or make insufficient changes, Additional dependencies due to the multilayered nature of policy make maintenance even more complex. The more complex a policy, the harder it is to manage and the more prone it is to be incorrect. Indeed, recent research provides quantitative evidence that evolution actually increases the complexity of Layer 3 network configuration [1].

However, security policies, even low-level, are forms of text, and the fields of software engineering and data mining give us building blocks for automated tools to deal with multiple levels of evolving text.

We believe that change detection will help practitioners to update policy in a way that reduces errors. We propose *hierarchical change mining* to detect changes and patterns of change across multiple layers of policies—whether low-level router configurations or high-level natural-language policies, perhaps with change logs.

Our problem is important to both real-world practitioners and security researchers. In our fieldwork, we have seen firsthand that practitioners struggle to understand how changes that must be made to policy will affect the enterprise mission. Often improper or incomplete changes increase as a result of policy complexity. Both low-level and high-level policy changes have major consequences if they are done properly or improperly.

Security policies are axiomatic to the *discipline* of security. In the literature, the phrase "security policy" may include mechanisms to ensure quality-of-service, to configure a network, or to authenticate and authorize users to a system. Traditional Orange-Book security methods formalize *Discretionary Access Control (DAC)* and *Mandatory Access Control (MAC)* policies with access-control matrices and lattices respectively [17]. However, these Orange Book policy formalizations often play less of a role in security in actual practice: according to a leading European industrial security consultant, well-structured security policies are becoming less important and 'risk assessments' are becoming more important [13]; according to a study in our lab, when users think deeply about a Facebook-like privacy policy, they actually leak *more* personal information than when they don't think so carefully [33]; according to Benson et al [1], configuration errors cause a large number of network outages and the misconfiguration risk is a function of complexity. Managing real-world security policy for large real-world systems is a pressing problem.

Furthermore, real-world security policies must be defined iteratively since policy correctness evolves over time. Software engineering denotes such systems as *E-type*: systems *embedded* in the real-world and whose correctness depends upon the "usability and relevance of its output in a changing world" [18]. Consequentially, network security policies must be continually re-defined and re-evaluated to accommodate their changing environment. This sentiment is echoed by other network security researchers who have noted that "top down security policy models are too rigid to cope with changes in dynamic operational environments" [20].

This Paper Section 2 discusses two real-world instances of the security policy evolution problem and how practitioners currently address it. Section 3 proposes hierarchical change mining and Section 4 evaluates our results against our use cases. Section 5 orients our research within the fields of software engineering, data mining, and network management. Section 6 concludes.

2 Real-World Use Cases

The security-policy evolution problem emerged from our collaborations with practitioners in network adminis-

tration and identity management and recent findings in software engineering. Properly done, changing a router configuration allows emergency operators to resolve calls from VOIP phones at Dartmouth to a physical location. Improperly done, changing the policy can crash the router that provides phone service when that router reboots. If the change is not made, emergency workers may not know how to get to the scene of an incident. Properly done, changing a security policy allows a university to access massive datasets and computational power via the grid. Improperly done, the university increases the vulnerability of the entire grid. If not done, the university is denied research opportunities.

We now discuss some observed limitations of the currently-practiced state-of-the-art.

Router Configuration in Enterprise Networks Collaborating with real-world practitioners from Dartmouth Computing Service, we analyzed four years (2005-2009) of Cisco router configuration files, which specify access lists and VLANs may be used to group users into a single logical subnet with uniform routing rules.

One example of requirements evolution these practitioners faced during this period was the change of college telephony to *Voice Over IP (VOIP)*, and the resulting directive that every VOIP phone be resolvable to a physical location for 911 emergencies. Network administrators had no choice but to evolve policy; specifically, they had to set the MAC address of every VOIP phone on the network with a *switchport* command. However, if they didn't also update the router's *flash*, then the router would be in a bad state upon reboot. Therefore, we see that due to dependency rules, a change to a configuration file may be necessary but not sufficient to keep the router in a good state; the literature attests to the relation between router misconfiguration and network service failures [23].

To aid with this evolution, the current practice is to use the *Really Awesome New Cisco configuration Differ (RANCID)* tool [27]. However, practitioners told us of several limitations. First, RANCID does not report changes with respect to dependencies; practitioners must manually check to make sure that if a MAC address is changed then the flash on that router is updated.

Second, RANCID may report meaningless changes that add noise to the change results. For example, if one permutes five lines in a block of the configuration file, then RANCID will report it as 5 deletions and 5 insertions even though the behavior of the configuration is unchanged.

Identity Management in Federations Collaborating with real-world practitioners from the *International Grid Trust Federation (IGTF)* [14], we analyzed their member organization's *certificate policies* and *certification practices statements (CP/CPSs)*. In this context, a *grid* is a distributed computing system that provides researchers access to massive amounts of computing power, storage, and data. The IGTF accredits organizations against a common set of standards for grid authentication. An organization that is IGTF-accredited can authenticate to several large-scale computational grids via a PKI certificate. Anyone not accredited cannot authenticate. Natural-language texts, these CP/CPSs ensure that "certificate generation, publication, renewal, re-key, usage, and revocation is done in a secure manner" [10]. The IGTF base policy determines accreditation guidelines.

As an example of policy evolution, when the IGTF changes their base accreditation guidlines, member organizations have 6 months to comply. If members don't comply, they lose accreditation. If member organizations comply poorly and still get accredited, then they expose themselves to legal risk. Moreover, they risk losing institutional access to grid services. Either way, changes to policy may result in increased exposure to risk.

Current practice for the IGTF to manage compliance relies on manual inspection of CP/CPSs combined with changelogs, where member organizations have recorded changes that affect their level of assurance [28]. Analysts will look at the previous and current versions of the CP/CPS to try to detect changes; analysts may also use change detection software such as Adobe redlining tools or Microsoft Word's Track Changes.

However, real-world data from the current, manual practice of documenting and reviewing changes revealed limitations. For example, when we compared the number of changes reported in the changelogs to the number of actual changes, we found changes reported between two versions of policy whose corresponding passages were identical—and other places where the polices had far more changes made than reported. Current practice does not suffice for effective management.

Software Engineering Recent findings from software engineering validate our concerns about this insufficiency. In 2010, Israeli and Feitelson [15] looked at the evolution of the Linux kernel and argued for code-based measurements for software versus surveys and logs. They cite a study by Chen et al. that compares change logs for three software products and their corresponding changed source code; this study showed that 80% of the changes were not logged [8]. Another example comes from a 2007 study by Fluri et al. which looked at three open-source systems and described how comments and code co-evolved. They found that newly-added code barely gets considered despite its considerable growth rate.

3 Hierarchical Change Mining

Our proposed approach models security policies as hierarchically-structured texts and mines changes within these structures over time. We use the Cisco router and IGTF polices discussed above as *longitudinal*[1] datasets. When practitioners modify a security policy, they create a new version of that policy. This sequence of policies, ordered by time, is a rich stream of historical data.

Policy as Hierarchically-Structured Text Security policy languages are typically hierarchically structured, and practitioners often group these policies hierarchically. Our approach accommodates both these senses. Commands within Cisco IOS are hierarchically structured and the *show* command for a router's start configuration or running configuration serializes this hierarchy. Furthermore, network-administrators group router configurations hierarchically; admins may decompose a network into *edge* and *core* routers. IGTF security policies are written as hierarchically-structured documents with sections, subsections, and subsubsections. IGTF policy analysts group member organization's policies into sub-federations such as *The Americas Grid Policy Managment Policy Authority (TAGPMA)*, *The European Union Grid Policy Management Authority (EuGridPMA)*, and the *The Asia-Pacific Policy Management Authority (APGRID)*.

We model the hierarchical structure of a policy as a tree. Vertices within a policy tree represent syntactic blocks of text; children correspond to blocks nested within that block. Each vertex has a unique reference string, persisting across policy versions, that allows us to compare change based upon policy *structure* rather than line numbers (which vary). Tables 1 and 2 list sample reference strings.

Quantifying Hierarchical Evolution We quantify change in two steps: we parse policy trees into change tables, then we mine these change tables. A row of a change table associates two versions of a syntactic block with a set of change features like word or tree edit-distance. Tables 1 and 2 list rows from change tables for IGTF PKI and Cisco IOS policies respectively. Currently, we compute word and tree edit-distance with a postorder traversal of the policy tree, summing features as we move up the tree. If desired, we can also sum features across multiple versions of a single policy or groupings of multiple policies. For example, we analyzed all VOIP routers in Dartmouth's Nugget Theater. In ongoing work, we are mining these change tables in order to detect changing and frozen policy structures, association rules, and trends.

Reference	Description	wordED	treeED
SDG. 1_5_1:6.1.1	In Sec 6.1.1, added more description	12	0
AIST.1_1:1.4.3	Added Section 1.4.3	21	1
IUCC.1_5:4.6.1	Changed 4.6.1 to add logging of login, logout, and reboots	0	0

Table 1: Our PKI-policy change table associates change features with paths to syntactic blocks for sections, subsections, and sub-subsections mentioned in changelogs. A path takes the form of *organization.edition:section*. These rows illustrate a content change, an added section, and a change that never occurred respectively.

4 Evaluation

We now discuss how hierarchical change mining improves on existing research and addresses the problems we introduced in Section 2.

Router Configuration in Enterprise Networks Our methods to quantify policy evolution build upon and improve state-of-the-art in studies of router configurations using longitudinal data [9, 25, 29, 30]. Previous studies have not considered hierarchical change, nor have they looked at considered commonly-used variable names as opposed to commonly-used tokens. Sung et al. [30] define blocks and superblocks to study correlated changes across router configurations. While superblocks allow one to see commonly occurring tokens, they do not allow one to see how these tokens change with respect to the hierarchical structure of the configuration.

Plonka et al studied the evolution of router configurations using the stanza [25]. Their approach, like Sung's, does not allow one to count how often a *particular interface* such as *FastEthernet0/8* is used, only the total number of times the command *interface* occurs. In addition, our hierarchical model of Cisco IOS is more general than stanza-type analysis, which only counts revisions for level-0 syntactic blocks such as *global* and level-1 syntactic blocks such as *interface*, *vlan*, *logging*, *ip*, and *bridge* [25]. In contrast, our approach considers paths to syntactic blocks at any level in the Cisco IOS command hierarchy. This rich syntactic structure allows practitioners to drill-down within change reports for more detail than previously provided.

In Table 2, we count level-1 change hits and level-2 change hits separately to avoid double counting a single change. For example, if we add 5 *switchport* commands to an *interface*, then the level-2 change count would be 5, but the level-1 change count would be 1. If we summed change counts across levels of the tree, then we would double-count one of the level-2 changes. Also, the total tree edit-distance for the */root/interface** category at level-1 (1542) is greater than the total tree edit-distance for the same category at level-2 (628). A possible explanation that contributes to this is the set of edits required to insert and delete changed, level-2

[1] A longitudinal dataset is a sequence of versions of a policy that change over time [30].

Reference	Total treeED	Hits	Reference	Total treeED	Hits	Reference	Total treeED	Hits
/root/interface*	1542	80	/root/logging*	0	0	/root/interface_FastEthernet0_8/switchport*	17	17
global	304	278	/root/bridge*	0	0	/root/interface_FastEthernet0_8/switchport_port-security*	13	13
/root/vlan*	28	25	/root/interface*	(1542/628)	(80/628)	/root/interface_FastEthernet0_8/switchport_voice*	2	2
/root/ip*	18	18	/root/interface*/switchport*	247	247	/root/interface_FastEthernet0_8/switchport_mode*	2	2

Table 2: Our change table for all VOIP routers in the Nugget Theater allows for an analysis of level-1 commands that resembles Plonka and Tack's stanza-type analysis. We differ, however, in that our change feature is tree edit-distance, and our *global* category refers to all unindented lines that are not any of the others in the list. More importantly, our hierarchical model allows us to navigate the Cisco IOS command hierarchy and get increasingly more specific change information. Where appropriate, total tree edit-distances and total hits are reported in terms of level-1 and level-2 structures using the format *(level-1 score/level-2 score)*.

components as children of level-1 vertices.

In ongoing work, we are mining associations between *categories* of syntactic blocks as well as between *particular instances* of syntactic blocks. *Generalization* replaces arguments to commands with a token for their production rule. For example, MAC addresses are replaced with the token "MACADDRESS" [1]. An example of a *generalized* association rule comes from Sung's approach [30]. Using Sung's approach, practitioners can detect associations between the *access-list*, *interface*, and *policy-map* categories. However, using our approach, practitioners can also drill-down to see the particular *access-lists* and *policy-maps* with which *interface FastEthernet 0/8* is associated.

Identity Management in Federations Our initial and ongoing work in hierarchical change mining also improves upon research in identity management in federations and addresses real-world problems that policy analysts face during accreditation.

Our *hierarchical* model of policy is based on over 20 years of experience to model, reference, and retrieve Classical texts [11, 12, 36, 37]. Others have done work in high-level policy formalization. Authorization and authentication policies have been formalized using various XML vocabularies including SAML [4] and XACML [22]. Previous work in certificate policy formalization focuses less on human-readable, machine-actionable representation than our prior research [35, 38] and our current work. Several have explored using ASN.1 to model properties based on a CP's source text [2, 26]. Others like Casola [5, 6] have experimented with data-centric XML representations of policy. and Jensen have experimented with data-centric XML representations of policy. Contemporary to our previous work, Jensen [16] encoded policies using DocBook [34].

Real-World Impact Administrators recognize that tools like RANCID are unaware of policy dependencies and that change reports may contain false changes because RANCID is not aware of the syntactic structure of Cisco IOS. Even our initial work improves upon this state-of-the-art; a practitioner has lauded how the hierarchical structure of policies gives an intuitive framework against which to organize and interpret change.

Rather than reporting line numbers and forcing administrators to peruse the entire configuration file, we express change in terms of structural features—and can generate queries to see the evolution of a *particular interface* or *access-list* over a range of time across an entire network or within a single router. In ongoing work, we plan to try to correlate syntactic changes within a single router configuration file and between multiple layers of network administration. For example, one practitioner thought it might be interesting to look at correlations between *ip helper-addresses*, servers that state which DHCP addresses to use, and snapshots of active DHCP leases. One could also look at correlations between router configuration files and network snapshots of active MAC addresses. At a higher level, it would also be interesting to try to correlate changes to router configurations with bug reports [24].

Administrators also must manually sift through meaningless changes that have no effect on the behavior of routers. Our change tables are syntax-aware, and so can enable filtering out meaningless changes according to Cisco IOS semantics. We are the first, to the best of our knowledge, to identify and address this problem.

Our initial work also improves the state of practice in our other application domain. The IGTF's current accreditation process uses change logs as well as manual review; the initial change tables we generated, however, revealed problems. Our initial study of 13 IGTF member organizations revealed 5 organizations with at least one *reported* change for which there was no *actual* change in the policy. Out of a total of 178 reported changes, 9 of those changes corresponded to no actual change. Of the 94 of 178 that claimed major change, we found 5 that were logged but never performed.

To detect the the second type of discrepancy, we are building a change feature classifier that decides whether or not to add a section to a changelog. This problem is more challenging since not all policy changes affect level-of assurance.

5 Related Work

To the best of our knowledge, we are the first to observe that practitioners express security policy in multiple layers ranging from high-level natural-language texts

to low-level configuration files and that these policies must be changed and synchronized in order to maintain security. We are also the first to propose a unified methodology to understand change in many of these policies through the lens of hierarchically-structured text. Our work improves upon and leverages the state-of-the-art in security, network service management, data-mining, and software-engineering.

A few other security researchers have introduced the term *security policy evolution* in the context of autonomous security. A 2004 proposal [21] proposed a mechanism for systems to dynamically change access rights in authorization policies, collecting data, using machine-learning to find patterns, and then using planning and optimization to construct a new policy on the fly. Unfortunately, this proposal did not appear to generate any follow-on work. The proposal was cited in later papers [20, 31] discussing how to use genetic programming to automatically generate new policies. Our approach differs from this group in that our goal is to produce tools that produce useful information for practitioners. Our approach ensures that humans remain accountable for security decisions.

Our work is also unique in that it considers the necessity, implications, and properties of security-policy evolution through the lens of structured text, whether a high-level or low-level policy, in contrast to some prior work studying the evolution of network policies through data mining and software engineering.

Hierarchical Change Mining The various components of our *hierarchical change mining* leverage and improve upon the state-of-the-art in data mining and software engineering. Computing a change table between two consecutive versions of a policy is an instance of Chawathe's [7] *hierarchical change detection problem*. Our approach is unique, however, in that we extract multiple features from policy node content; most others focus on changes to the tree structure but not the lower level features discussed above. Furthermore, a comprehensive 2009 review of XML similarity notes that a future research direction in the field would be to explore similarity methods that compare "not only the skeletons of XML documents...but also their information content" [32]. Our work improves upon the state-of-the-art in change detection.

Our work also applies the Zhao et al.'s 2005 best student paper *XML Structural Delta Mining*, which proposed a vision for mining change patterns in XML documents [39]—although our approach works with any tree (not just an XML DOM). More recently, Leskovec et al. studied how phrases evolve in the blogosphere [19]; Bottcher et al. introduce the new paradigm of *Change Mining* as "data mining over a volatile, evolving world with the objective of understanding change" [3]. Our work also improves upon the state of the art in data mining.

6 Conclusion

We have introduced and motivated the security-policy evolution problem with anecdotal evidence from real-world practitioners in router configuration and PKI policy accreditation. We have argued the importance and necessity of studying policy evolution and its relation to policy complexity in network services. To our knowledge, we are the first to consider a unified approach to manage change in multiple layers of network-security policies ranging from high-level natural-language policies to low-level router configurations. We accomplish this by modeling policy as a hierarchically-structured text and have proposed general methods to manage change that build upon state-of-the-art research in network management, data-mining, and software-engineering.

We have several ideas for future work. We noted before that Benson et al. observe that evolution increases the complexity of network configuration and that quantifying complexity can help one to evaluate network design decisions [1]. Just as this paper describes work in progress on general methods to quantify changes to policy, we would like to investigate how well Benson's referential complexity metrics generalize to other security policies. Our current work looks at two longitudinal datasets that correspond to two different layers of two distinct network security policies. We would like to study multiple layers of the same network security policy in the future. Furthermore, we would like to also apply *hierarchical change mining* to detect changes to other policies such as SELinux.

References

[1] T. Benson, A. Akella, and D. Maltz. Unraveling the Complexity of Network Management. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pages 335–348. USENIX Association, 2009.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[3] M. Bottcher, F. Hoppner, and M. Spiliopoulou. On Exploiting the Power of Time in Data Mining. *ACM SIGKDD Explorations Newsletter*, 10(2):3–11, 2008.

[4] S. Cantor, J. Kemp, R. Philpott, and E. Maler. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.* 2005.

[5] V. Casola, A. Mazzeo, N. Mazzocca, and M. Rak. An Innovative Policy-Based Cross Certification Methodology for Public Key Infrastructures. In *EuroPKI*, 2005.

[6] V. Casola, A. Mazzeo, N. Mazzocca, and V. Vittorini. Policy Formalization to Combine Separate Systems

into Larger Connected Network of Trust. In *Net-Con*, page 425, 2002.

[7] S.S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change Detection in Hierarchically Structured Information. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 493–504. ACM, 1996.

[8] K. Chen, S.R. Schach, L. Yu, J. Offutt, and G.Z. Heller. Open-Source Change Logs. *Empirical Software Engineering*, 9(3):197–210, 2004.

[9] X. Chen, Z.M. Mao, and J. Van der Merwe. Towards Automated Network Management: Network Operations Using Dynamic Views. In *Proceedings of the 2007 SIGCOMM Workshop on Internet Network Management*, pages 242–247. ACM, 2007.

[10] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. RFC3647: Internet X. 509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. 2003.

[11] Gregory Crane. The Perseus Digital Library. Retrieved May 29, 2009 from `http://www.perseus.tufts.edu/hopper/`.

[12] C. Dué, M. Ebbott, C. Blackwell, and D. Smith. The Homer Multitext Project, 2007. Retrieved May 29, 2009 from `http://chs.harvard.edu/chs/homer_multitext`.

[13] E. Rey, November 2010. Conversations on Security Policy and Risk Assessment.

[14] International Grid Trust Federation Charter. Retrieved May 29, 2009 from `http://www.igtf.net/new-doc/IGTF-Federation-20051005-1-igtf.html/`.

[15] Israeli, A. and Feitelson, D.G. The Linux Kernel as a Case Study in Software Evolution. *J. Syst. Softw.*, 83:485–501, March 2010.

[16] J. Jensen. Presentation for the CAOPS-IGTF session at OGF25, March 2009.

[17] D.C. Latham. Department of Defense Trusted Computer System Evaluation Criteria. *Department of Defense*, 1986.

[18] M.M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE*, 68(9), 1980.

[19] Leskovec, J. and Backstrom, L. and Kleinberg, J. Meme-Tracking and the Dynamics of the News Cycle. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

[20] Y.T. Lim, P.C. Cheng, P. Rohatgi, and J.A. Clark. Dynamic Security Policy Learning. In *Proceedings of the First ACM Workshop on Information Security Governance*, pages 39–48. ACM, 2009.

[21] P.D. McDaniel. Policy Evolution: Autonomic Environmental Security. *Software Engineering Research Center Showcase*, 2004.

[22] T. Moses. *eXtensible Access Control Markup Language (XACML) Version 2.0.* 2005.

[23] D. Oppenheimer, A. Ganapathi, and D.A. Patterson. Why do Internet services fail, and what can be done about it? In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems*, volume 4, page 1. USENIX Association, 2003.

[24] P. Schmidt, November 2010. Conversations on Cisco Router Configuration.

[25] D. Plonka and A.J. Tack. An Analysis of Network Configuration Artifacts. In *Proceedings of the 23rd Conference on Large Installation System Administration (LISA)*, page 6. USENIX Association, 2009.

[26] R. Grimm and T. Hetschold. Security Policies in OSI-Management Experiences from the DeTeBerkom Project BMSec. *Computer Networks and ISDN Systems*, 28:499, 1996.

[27] RANCID - Really Awesome New Cisco Config Differ. Retrieved December 1, 2010 from `http://www.shrubbery.net/rancid/`.

[28] S. Rea, November 2010. Conversations on PKI Policy Accreditation.

[29] X. Sun, Y.W. Sung, S.D. Krothapalli, and S.G. Rao. A Systematic Approach for Evolving VLAN Designs. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[30] Y.W. Sung, S. Rao, S. Sen, and S. Leggett. Extracting Network-Wide Correlated Changes from Longitudinal Configuration Data. *Passive and Active Network Measurement*, pages 111–121, 2009.

[31] J.E. Tapiador and J.A. Clark. Learning Autonomic Security Reconfiguration Policies. In *2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)*, pages 902–909. IEEE, 2010.

[32] J. Tekli, R. Chbeir, and K. Yetongnon. An Overview on XML Similarity: Background, Current Trends and Future Directions. *Computer Science Review*, 3(3):151–173, 2009.

[33] S.A. Trudeau, S. Sinclair, and S.W. Smith. The Effects of Introspection on Creating Privacy Policy. In *Workshop on Privacy in the Electronic Society*, 2009.

[34] N. Walsh and L. Muellner. *DocBook: The Definitive Guide*. July 1999.

[35] G. Weaver, S. Rea, and S. Smith. A Computational Framework for Certificate Policy Operations. In *Public Key Infrastructure: EuroPKI 2009.* Springer-Verlag LNCS., 2009.

[36] G. Weaver and D. Smith. Canonical Text Services (CTS). Retrieved May 29, 2009 from `http://cts3.sourceforge.net/`.

[37] G. Weaver and D. Smith. Applying Domain Knowledge from Structured Citation Formats to Text and Data Mining: Examples Using the CITE Architecture. In *Text Mining Services*, page 129, 2009.

[38] G.A. Weaver, S. Rea, and S.W. Smith. Computational Techniques for Increasing PKI Policy Comprehension by Human Analysts. In *Proceedings of the 9th Symposium on Identity and Trust on the Internet*, pages 51–62. ACM, 2010.

[39] Q. Zhao, L. Chen, S.S. Bhowmick, and S. Madria. XML Structural Delta Mining: Issues and challenges. *Data & Knowledge Engineering*, 59:627–651, 2006.