Submitted for publication. Author Copy - do not redistribute.

Re-Engineering Grep and Diff for NERC CIP

Gabriel A. Weaver and Sean W. Smith
Department of Computer Science
Dartmouth College
Hanover, New Hampshire 03784
Email: (gweave01,sws)@cs.dartmouth.edu

Rakesh B. Bobba and Edmond J. Rogers
Information Trust Institute
University of Illinois at Urbana-Champaign
Illinois, USA
Email: (rbobba,ejrogers)@illinois.edu

***Abstract*—The present and future smart grid has a large number of devices that produce an avalanche of data. Our research explores whether we can channel this avalanche to drive more efficient NERC CIP audits. The power industry has both high-level, natural-language NERC CIP policies and low-level data sources such as router configuration files, Windows registries, and PMU data. Utilities must use low-level data to demonstrate compliance at a high level and each utility does this in their own way. In this paper, we propose our two tools, Context-Free Grep and Hierarchical Diff, to help utilities and auditors generate reliable and reproducible evidence to support or refute NERC CIP compliance. Our tools are currently in the design phase, although we have prototyped Hierarchical Diff.**

I. Introduction

The smart grid will increase the stability and reliability of the grid overall with vast numbers of cyber components. Many of these devices will generate data.

Power system control networks must comply with the North American Electric Reliability Corporation's Critical Infrastructure Protection (NERC CIP) regulations. The consequences of failing to fulfill these provisions are severe. According to one industry expert who has performed audits at a major utility, fines scale up to *1.5 million dollars per day of violation* retroactive to the beginning of the offense. In addition to financial consequences, failure to comply implies a lack of basic and sound security controls, making the system vulnerable to cyber attacks and their consequences, including power outages.

Despite the importance of NERC CIP compliance audits, there are a few drawbacks to current audit process. First, the audit process can be arbitrary and subjective. Secondly the audit process is costly. At Investor-Owned Utilities (IOUs), a conservative estimate suggests that audits consume 30 man days per day of audit. This estimate does not include ramp up time for the auditors. Furthermore, audits cost large IOUs from hundreds of thousands to millions of dollars. Currently utilities are on a three-year audit cycle. However, the Federal Energy Regulatory Commission (FERC) would like annual audits.

This Paper This paper makes two contributions to try to address the drawbacks of the current NERC CIP audit process. First, in Section II, we observe that network and system configuration files form a dataset that utilities and auditors may analyze for a more consistent, reliable, data-driven NERC CIP audit. Given the estimated number of devices on the smarter grid of the future, we expect this dataset to only grow in size over time.

Second, in Section III, we propose two tools that emerged from our broader work on security policy management but which we argue will help with CIP audit. Our Context-Free Grep and Hierarchical Diff tools are designed to extract and measure changes to network and system configuration files. We argue the **utility** of our tools; that our tools provide the functionality needed to evaluate and document compliance in several CIP provisions. These CIP provisions include CIP-003-4 (Cyber Security - Security Management Controls), CIP-005-4a (Electronic Security Perimeter(s)), and CIP 010-1 (Configuration Management and Vulnerability Assessments).

In Section IV we describe in detail our plan to evaluate the correctness, performance, and usability of our tools. Section V discusses our second contribution in the context of prior work done in industry and academia. Finally, Section VI concludes.

II. Datasets for CIP Audit

A. Cisco IOS

Utilities must manage their computer networks. In our fieldwork, we have found that administrators must interpret and implement security policies, sift through network configuration data, and track changes to a network's security configuration through time. Although there are several options to managing a network, we focus on Cisco IOS, a configuration languge for Cisco routers and switches.

Cisco IOS configuration files contain information that both utilities and auditors may use to understand how a cyber control network evolves over time. NERC CIP-005-4a requires utilities to update documentation to reflect changes to a network within 90 days of the change. In fact, network operators consider router configuration files to be the "most accurate source of records of changes" [1], [2].

Although Cisco IOS has commands such as `banner` that make it context-sensitive [3]. We argue that a meaningful subset of Cisco IOS is nontheless context-free. Intuitively, context-sensitive grammars are more general than context-free languages and most programming languages are, in theory, context-free. Although Cisco IOS lacks a formal grammar, the language still represents constructs of interest through its syntax. Many of these structures are either blocks (such as the `interface` command) or span multiple lines.

B. Windows Registry

A majority of power control systems use Windows and NERC CIP was written for Windows-based control systems. We therefore focus on Windows.

We argue that utilities and auditors may use the information in a Windows registry to define a baseline configuration of a system; baseline configurations are required by NERC CIP-010-1. The Windows Registry stores configuration information in a hierarchically-structured set of key/value pairs. For example, NERC CIP 010-1 R1.1 requires utilities to "develop a baseline configuration of the BES Cyber System, which shall include: physical location, OS(s) and versions, any commercially available application software (and version), any custom software and scripts installed, logical network accessible ports, and security-patch levels." Figure I shows that most of the elements of a baseline configuration are present in the Windows registry.

Currently, there is no general best practice for utilities to craft a baseline configuration. There are change management products such as ChangeGear and Remedy, but generally, many utilities will use spreadsheets to track revisions and system levels.

There are other components of the Windows registry that utilities could use to define a baseline configuration. For example, the registry also specifies how a system handles file extensions. Utilities could treat entries in this part of the registry as a whitelist of accepted file extensions and have all other file extensions be opened through a special application (or not opened at all). On a production machine, for example, extensions such as `.cpp` may be unnecessary since source files are usually unnecessary in production.

III. Our Tools

A. Context-Free Grep

Although grep is a traditional UNIX workhorse for sifting data, it was not designed to extract information in terms of the syntactic and/or logical structure of modern configuration and programming languages. Specifically, grep can only recognize regular languages and regular expressions are not powerful enough to extract blocks of text nested arbitrarily deep. For a general-purpose tool, we need to be able to extract structures at arbitrary nesting depth.

For example, Cisco IOS, although it lacks a formal grammar, represents meaningful constructs of interest through its syntax. Some of Cisco IOS' data structures are reflected in its syntax. These structures, such as the `interface`, are expressed as blocks. Other structures, such as the `access list` span multiple lines but are logically grouped together by identifier rather than by a block.

Our Context-Free Grep tool aligns the *unit of extraction* with the block structures found in modern programming and configuration languages. Given a grammar that generates the syntactic structures of interest, and an input file, our proposed tool finds matching strings in the language of the subgrammar, and reports those matches within *meaningful regions* of text.

Utilities and auditors will be able to use our Context-Free Grep tool to directly query network and system configuration files in terms of a language's structure. As we will see, this capability will be useful in the context of NERC CIP-005-4a.

Furthermore, once we have the ability to parse a context-free subset of a language like Cisco IOS or Windows registry, practitioners can grep out this structure as an abstract syntax tree. Different kinds of syntax tree consumers could then generate reports in a variety of formats and one of these formats could be a controlled natural-language description of contents. This would be similar in flavor to the work done by Inglesant, Sasse, and Chadwick in which they used a controlled natural language to create RBAC policy [4].

B. Hierarchical Diff

Diff is a time-honored UNIX shell tool whose *unit of comparison* is the line. TkDiff recognized that other units of comparison are useful and so began to color the differing portion of the string. Many modern programming and configuration languages, however, specify structures (meaningful units of text) that span multiple lines. For example, Cisco IOS' `interface` blocks, C functions, and subtrees in the logical structure of a Windows registry.

Our Hierarchical Diff tool aligns the *unit of comparison* with context-free substructures found in programming and configuration languages. In its most general form, Hierarchical Diff takes two versions of a file written in the same configuration language or syntax. These files are parsed into an intermediate tree representation. The current intermediate representation is XML, however, XML is a bloated format and the final version of our tool may use an alternative representation. Once in an intermediate tree representation, we compare both versions in terms of the structure of the file and its content. Finally, we output a change report that describes how to transform the first version of the file into the second.

Our Hierarchical Diff tool will let utilities compare network and system configuration files in terms of their syntactic structure rather than by line number. We anticipate this tool will be useful in NERC CIP compliance audit.

C. Our Tools and NERC CIP

We now discuss the utility of our proposed tools in the context of NERC CIP compliance audit. Table II quotes the provisions that we discuss. Table III summarizes the requirements that emerge from our discussion.

CIP-005-4a NERC CIP-005-4a R5.2 requires responsible entities to update documentation within 90 days of a change to the network [5]. In this manner, documentation reflects the actual configuration of the power system network.

Our Context-Free Grep and Hierarchical Diff will enable utilities and auditors to *extract* and *compare* meaningful regions of network configuration files. We hypothesize that both of these capabilities will be time-efficient ways to demonstrate compliance with CIP-005-4a R5.2.

Utilities and auditors will benefit from the capability to *extract meaningful regions* of text from system and network

CIP 010-1 R1 Baseline Configuration Component	Corresponding Windows Registry Subtree
1.1.1. Physical location	not available
1.1.2. Operating system(s) (including version)	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
1.1.3. Any installed application software (including version)	HKEY_LOCAL_MACHINE\Software InstallDir, Revision, and Version are contained in this subtree
1.1.4. Custom software and scripts	Uncertain, perhaps in HKEY_LOCAL_MACHINE\Software
1.1.5. Logical network accessible ports	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip
1.1.6. Security-patch levels	HKEY_LOCAL_MACHINE\Software\Microsoft\Updates

TABLE I
MOST OF THE ELEMENTS OF A BASELINE CONFIGURATION ARE PRESENT IN THE WINDOWS REGISTRY.

		Description
CIP 003-4	R6	Change Control and Configuration Management — The Responsible Entity shall establish and document a process of change control and configuration management for adding, modifying, replacing, or removing Critical Cyber Asset hardware or software, and implement supporting configuration management activities to identify, control and document all entity or vendor- related changes to hardware and software components of Critical Cyber Assets pursuant to the change control process.
CIP 010-1	R1.1	Develop a baseline configuration of the BES Cyber System, which shall include the following for each BES Cyber Asset identified, individually or by specified grouping: 1.1.1. Physical location; 1.1.2. Operating system(s) (including version); 1.1.3. Any commercially available application software (including version) intentionally installed on the BES Cyber Asset; 1.1.4. Any custom software and scripts developed for the entity; 1.1.5. Any logical network accessible ports; and 1.1.6. Any security-patch levels.
	R1.2	Authorization, by the CIP Senior Manager or delegate, and document changes to the BES Cyber System that deviate from the existing baseline configuration.
	R1.5	For each change that deviates from the existing baseline configuration for Control Centers...
	R2.1	Where technically feasible, monitor for changes to the baseline configuration (as defined per CIP-010 R1, Part 1.1) and document and investigate the detection of any unauthorized changes.
CIP 005-4a	R5.2	The Responsible Entity shall update the documentation to reflect the modification of the network or controls within ninety calendar days of the change.

TABLE II
THE NERC CIP PROVISIONS DISCUSSED IN THIS PAPER.

configuration files. In CIP-005-4a R5.2, responsible entities must update documentation within 90 days of a change to the network.

Context-Free Grep would provide the mechanism for utilities and auditors to extract meaningful units of configuration files for comparison. Given a grammar and an input file, Context-Free Grep will find matching strings in the language of the grammar and report those matches within meaningful units of text. For example, utilities could extract all interfaces to which a certain access list is applied and have matches reported as interfaces. Figure 1 illustrates the usage scenario for our design.

Our approach is different from available tools. In traditional grep, one can approximate this effect by reporting lines of context surrounding a match. Cisco IOS's `show include` command similarly allows one to extract matching strings from a configuration. Our approach, however, allows one to grep out the interface block that contains the match. Once extracted, meaningful units of configuration files may be compared using our Hierarchical Diff.

Our Hierarchical Diff will provide utilities and auditors the capability to *compare meaningful regions* of system and

	CIP Provisions	Context-Free Grep	Hierarchical Diff
Software	CIP 003-4	1) extract Windows registry subtrees to monitor for changes.	1) compare versions of Windows registry subtrees. 2) Nightly change report generation.
Software	CIP 010-1	1) extract Windows registry subtrees relevant to a baseline configuration	1) compare a baseline configuration registry to a system registry. 2) Nightly change report generation
Network	CIP 005-4	1) locate strings generated by Cisco IOS sub grammar. 2) return matches in meaningful unit of text, such as an interface	1) compare Cisco IOS interface blocks 2) network configuration changelog generation

TABLE III
THIS PAPER MOTIVATES SEVERAL REQUIREMENTS FOR OUR CONTEXT-FREE GREP AND HIERARCHICAL DIFF IN TERMS OF NERC CIP REQUIREMENTS.

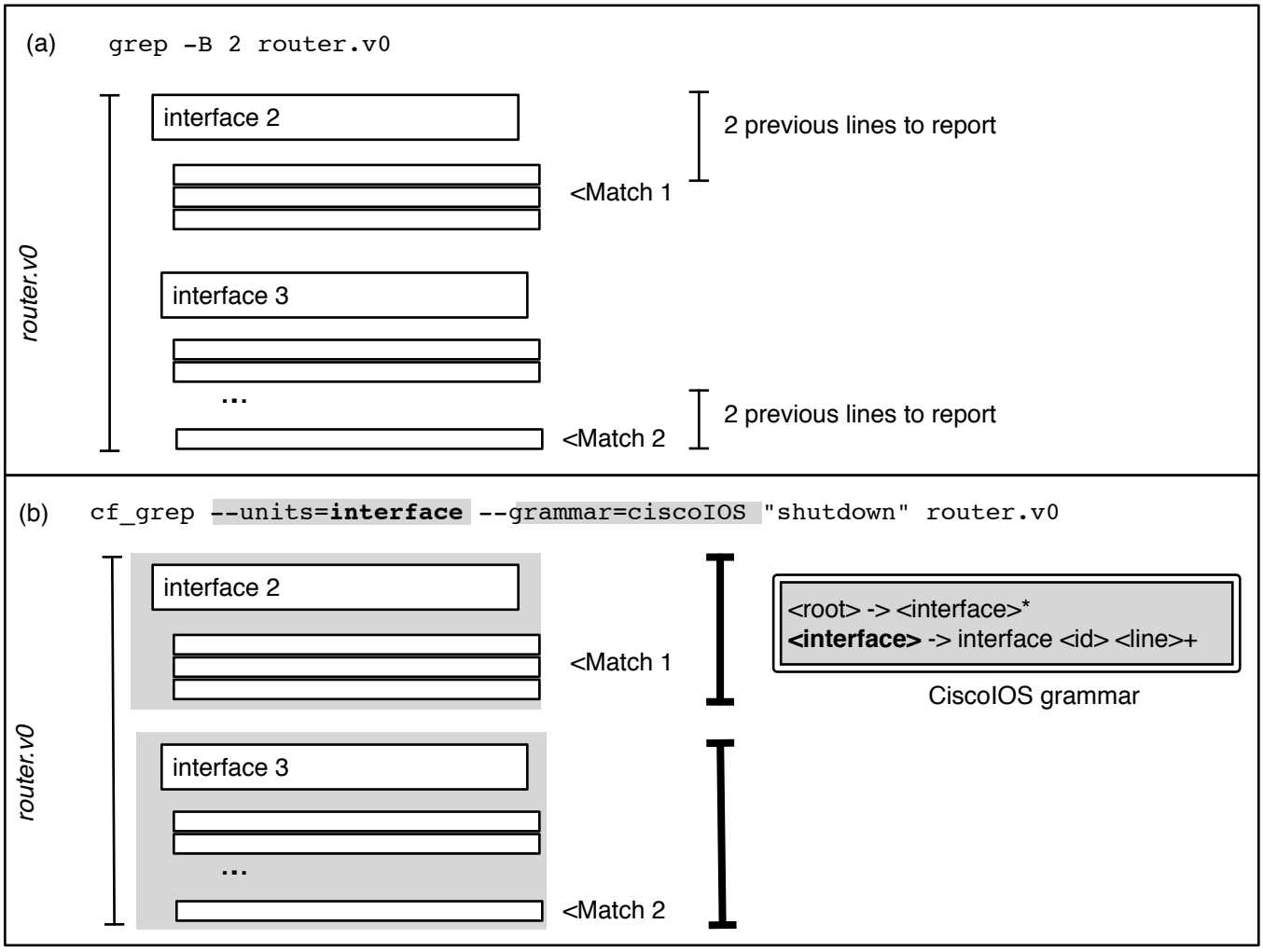

Fig. 1. (a) The unit of extraction in traditional grep is the line, but even with previous context, via the -B flag, matches may be missing useful information such as the interface name in which the match occurs. (b) In contrast, the unit of extraction in our Context-Free Grep corresponds to meaningful regions of text. Practitioners know where matches occur. The grammar need only specify production rules for interfaces and lines.

network configuration files. We hypothesize that this capability will be a time-efficient mechanism to demonstrate compliance with CIP-005-4a R5.2.

Auditors could use Hierarchical Diff to identify meaningful changes to network components. After Context-Free Grep extracts meaningful units of text, Hierarchical Diff could be used to compare them. For example, suppose we have two versions of a set of network interfaces that are on edge routers and use a particular access list. Hierarchical Diff compares these subtrees both in terms of tree structure and node values and then generates an edit script to transform one version into the other. The change operations defined by the edit script, report the meaningful changes to the system between those two versions.

The output of our Hierarchical Diff could also be used to help utilities to quickly document meaningful changes to both software and network components. Since change deltas are expressed in terms of a configuration language's syntactic structure, we could generate natural-language changelogs. These logs could then be used by utilities comply with requirements to document changes to network configurations.

Both our own work in other domains (such as PKI), as well as literature from software engineering suggests that traditional change logs, although used in audit, may not accurately reflect changes. In the domain of PKI we did an initial study of the changelogs for security policies of 13 organizations. We found that out of 178 reported changes, 9 of those changes corresponded to no actual change [6]. If these observations on changelogs generalize to power system networks, then automatically-generated, meaningful changelogs would both save time and increase accuracy of a compliance audit on a control network.

NERC CIP-003-4 and CIP-010-1 We now discuss how our tools will give utilities and auditors the capability to *efficiently and consistently* satisfy several provisions within NERC CIP-003-4 and NERC CIP-010-1.

NERC CIP-003-4 R6, and CIP-010-1 R1.2, R1.5, and R2.1 motivate the need for utilities to have the fundamental capability to *compare* system configurations. In NERC CIP-003-4, utilities must be able to "identify, control, and document" *meaningful changes* to software components of Critical Cyber Assets [5]. In NERC CIP-010-1, utilities must be able to compare a system configuration against a baseline configuration.

Both Context-Free Grep and Hierarchical Diff are our proposed tools for utilities and auditors to achieve the capabilitity of *meaningful configuration comparison* in Windows systems. We contend that this capability is useful to satisfy both NERC CIP-003-4 R6 as well as several provisions in NERC CIP-010-1.

Context-Free Grep would allow utilities and auditors to extract meaningful units of system configuration files to compare. For example, auditors could extract all references to Adobe within the software subtree of the logical view of a Windows registry.

Hierarchical diff would help utilities to "identify, control, and document" meaningful changes to software and network components as required by CIP-003-4 R6. In this case, we would compare the meaningful regions of text extracted by Context-Free Grep. For example, suppose we extract the subtrees that correspond to installed software.

To address CIP-010-1, our Hierarchical Diff will enable utilities and auditors to *measure* how a system changes from a baseline configuration directly from the Windows registry. In this case we would compare a baseline configuration to the current registry of a system.

Given two versions of a Windows registry subtree, our Hierarchical Diff will return an edit script that describes how to transform one registry into the other using operations such as `insert, delete, update, or move`. Unlike traditional diff, however, the edit script will be in terms of operations on the registry's tree structure. Practitioners may then either document changes to the cyber system by saving this edit script, or writing a summary of the changes. As mentioned before, it may also be possible to generate human-readable changelogs directly from the edit script.

Our Hierarchical Diff could help utilities and auditors to monitor changes to Windows Registries. If the baseline configuration was expressed as the logical view of the Windows registry, then a nightly batch process could either extract *interesting* registry subtrees and diff them or diff the current configuration against the baseline configuration. A change script or machine-generated change log could then be emailed to the administrator.

IV. Evaluation

In this section, we give a detailed plan of how we shall evaluate the correctness, performance, and usability of our Context-Free Grep and Hierarchical Diff tools.

A. Correctness

Table III, lists the functional requirements for our tools in order to help utilities comply with NERC CIP-003-4, CIP 010-1, and CIP 005-4a. These requirements, combined with use cases such as those shown in Figures 1 and 2 form the basis for evaluating correctness. We now describe our proposed evaluation of correctness for both of our tools.

Context-Free Grep: We want to verify two statements based on the usage diagram in Figure 1 to ensure the correctness of our implementation of Context-Free Grep. First, if we give `cf_grep` a pattern, a set of units in which to report found instances of that pattern, a grammar that specifies productions for those units, and a file to query, then all instances of that pattern that occur within the desired unit are reported. Note that there may be instances of the pattern that do not occur within a specified reporting unit. In that case, that instance is not output.

In order to verify these two statements, we will develop a test suite for our future implementation of Context-Free Grep. The use cases for our tests will be derived from each of the requirements outlined in Table III. This means that we will build up test cases for Cisco IOS and a logical view of Windows Registries.

Hierarchical Diff: We now describe how we intend to evaluate the correctness of Hierarchical Diff.

Based upon the usage diagram in Figure 2, we need to verify two statements. First, if we give `hier_diff` a set of units of comparison, a grammar that specifies productions for those units, and two files to compare, then our tool generates an edit script that describes how to transform the first file into the second. Note that the changes should be reported in the largest unit of comparison possible so as to reduce the length of the change report. For example, we note in Figure 2 that the move of *interface 2* is reported as a delete and an insert of one interface, rather than as deleting and inserting four lines.

Again, we can verify our notion of correct behavior for Hierarchical Diff with a test suite. The test suite will be based upon the requirements outlined in Table III. If we design our tests carefully, our *blackbox* tests will be able to evaluate any implementation of our Hierarchical Diff specification.

1) Performance: In order to estimate how many man hours our tools can save during a NERC CIP compliance audit, we need to evaluate the performance of our tools on realistic datasets. As mentioned before, current NERC CIP audits at major IOUs consume 30 man days per day of audit and cost large IOUs from hundreds of thousands to millions of dollars.

Towards this end, we have already begun to collect real-world datasets. For example, we have four years of Cisco IOS router configuration data provided by Dartmouth Computing Services. We need to build up a similar dataset for Windows Registries.

We will use both of these datasets to benchmark our Hierarchical Diff and Context-Free Grep to understand their memory usage and execution time relative to the size of the files input. In this manner, we will understand the context in which our tools can be used by utilities and auditors.

2) Usability: Finally, we want to evaluate our tools with usability studies based upon the requirements in Table III. We now outline some of the aspects of our tools that our usability studies should address.

Since both Context-Free Grep and Hierarchical Diff are redesigns of two traditional UNIX workhorses, we should evaluate both of our tools against traditional grep and traditional diff. In particular, one aspect of our tools that may make them harder to learn are the input grammars. Depending upon how difficult these grammars are for test participants to write, we may decide to have precanned grammars for utilities and auditors to use during a NERC CIP audit.

We want to evaluate the interpretability of our tools' output. For Context-Free Grep, how much do the meaningful units of extraction help test participants to locate matches? Is the additional structural information provided by these units actually useful or does the traditional grep suffice? In Section III we provided evidence for why we think our tool would improve traditional grep.

Additionally, we want to understand Hierarchical Diff and whether the meaningful units of comparison actually help here as well. Does expressing edit scripts in different units of comparison (such as interfaces) help or confuse users? How much savings in edit script length do we get by varying the unit of comparison. Figure 2 suggests there may be a benefit to our approach over traditional diff.

Finally, we want to evaluate the overall user satisfaction with our tool. In other words, we will make our tool available to utilities and auditors to try and collect anecdotal feedback. Already, we have some interested industrial partners.

V. Related Work

We now discuss our proposed tools in the context of prior work done in both industry and academia.

A. Context-Free Grep

Industry: Currently, there are a variety of tools available to extract regions of text based upon its structure. The closest tool we have found to our design of Context-Free Grep is sgrep [7]. SGrep is suitable for querying structured document formats like mail, RTF, LaTeX, HTML, or SGML. Currently, an SGML/XML/HTML scanner is available but it does not produce a parse tree. A parse tree library might be useful for our Hierarchical Diff. Nonetheless, the querying model of sgrep is worth paying attention to. If one is processing XML, XSLT may be used to transform and extract information based upon the structure of the XML.

Windows Powershell has a `Where-Object` Cmdlet that allows people to issue queries on the properties of an object at the command line. An object may be created from a source file by casting it as a type (such as xml) [8].

Pike's structural regular expressions allow users to write a program to refine matches based on successive applications of regular expressions [9]. Our approach is different because we extract matches based upon whether a string is in the language of the supplied grammar.

Academia Although Grunschlag has built a context-free grep [10], this classroom tool only extracts matches with respect to individual lines. Coccinelle [11] is a semantic grep for C. In contrast, we want our tool to have a general architecture for several languages used by system administrators.

Our Context-Free Grep complements research in network configuration management. For example, Sun et al. argue that the block is the right level of abstraction for making sense of network configurations across multiple languages. Despite this, however, they only look at correlated changes in network configurations in Cisco [1]. Similarly, Plonka et al. look at stanzas in their work [12].

B. Hierarchical Diff

In this paper, we argued that our Hierarchical Diff can help utilities and auditors to satisfy NERC CIP 010-1 because much of the information in a baseline policy is available in a Windows registry.

The goal of our Hierarchical Diff is to produce a general-purpose tool to compare multiple versions of a file with the same syntactic structure. Our tool is unique because we seek to build a tool that can do a structural comparison of files in

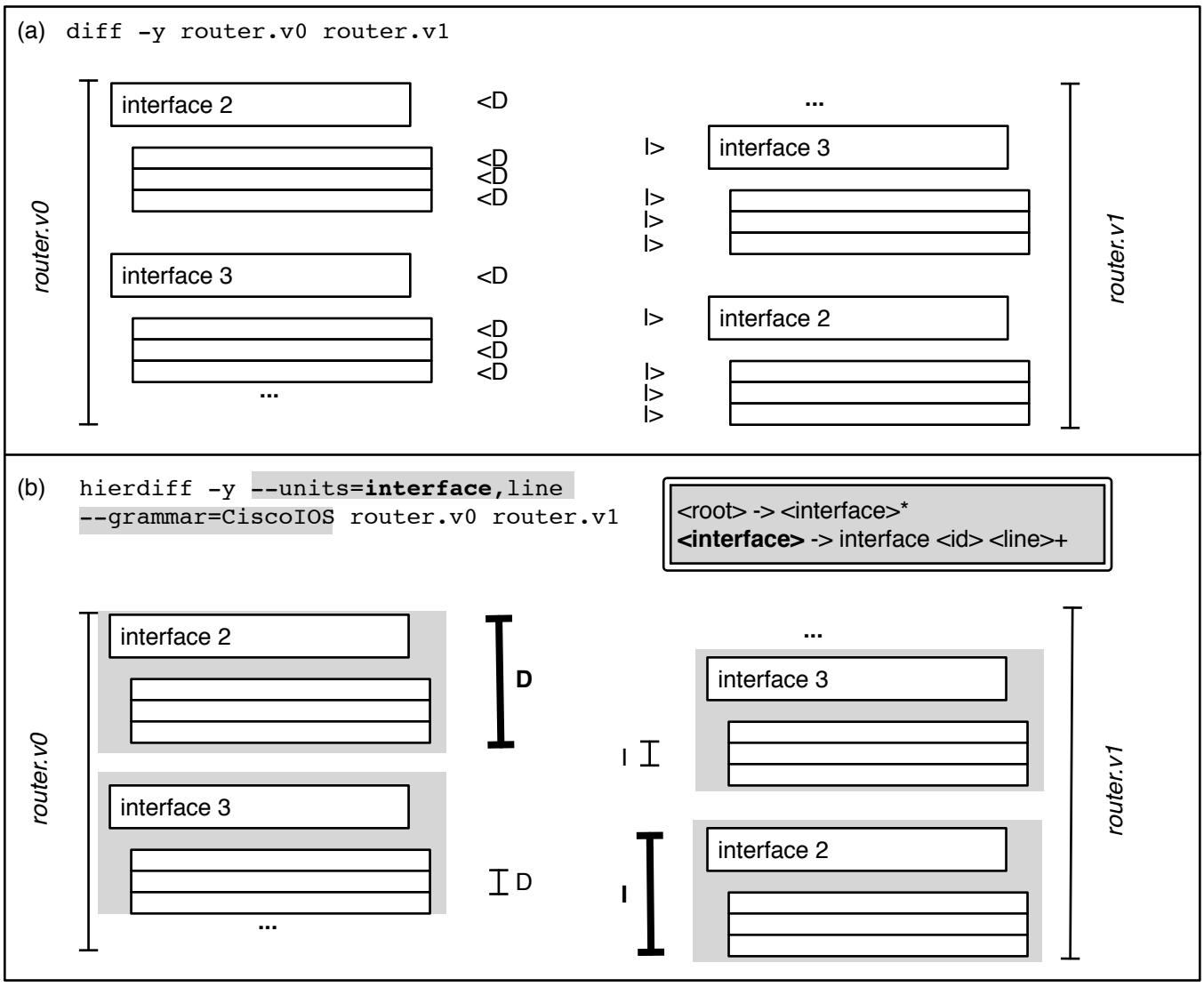

Fig. 2. (a) The unit of comparison in traditional diff is the line. (b) In contrast, the units of comparison in our Hierarchical Diff directly corresponds to meaningful, nested regions of text. This reduces the length of a change report. The grammar need only specify productions for interfaces and lines.

general. In fact, we could use our Hierarchical Diff to compare Cisco IOS configuration files as well as Windows Registries.

Industry: Several commercial products can already be used to help utilities and auditors with NERC CIP-010 compliance. In general CIP-010 R2.1 requires utilities to identify, monitor, and document changes to an existing baseline configuration. Open Source Tripwire [13] could be used to monitor changes to the files that form the Windows registry. The files could be hashed and a change to the hash would indicate a change to the registry. However, this technology would only inform utilities that a change occurred, not what the change was.

The commercial TripWire product monitors changes to a more general set of file attributes and couples this with a line-based diff tool that also could be used to monitor the files that represent the Windows registry [14]. We note, however, that CIP-010 concerns itself not with changes to the physical registry file, but rather to changes to the baseline configuration. Many components of the baseline configuration, are found in the logical structure of the Windows registry, not in the physical attributes of the registry files.

Change management products such as ChangeGear [15] and Remedy [16] provide a ticketing system to document changes. They do not provide a mechanism to compare the contents of files, to automatically monitor whether a file on a system has been changed, or to automatically document *how* the file was changed.

There are several commercial or open source products that utilities and auditors could use to compare two versions of a Windows registry. WinDiff [17] and the more recent WinMerge [18] are both line-based, graphical file comparison tools that one could use to compare two registry files. However, as both tools are line-based, change reports will not be expressed in terms of edits to the logical, tree-structure of a registry. Furthermore, since the tools are graphical in nature, it the

change reports may not be processed by subsequent services. Finally, TkDiff [19], available for Windows, improves upon line-based units of comparison by highlighting character differences within a changed line.

In contrast to the above commerical products, we designed our Hierarchical Diff to express changes in terms of the logical structure of the Windows registry. This means that changes are reported in terms of inserted, deleted, updated, or moved registry subtrees as well as changes to key values. Since registry subtrees correspond to required components of a baseline configuration, we expect our tool to help practitioners more quickly isolate changes relevant to CIP-010.

Academia: The various components of our Hierarchical Diff tool use and improve upon the state-of-the-art in computer science. Computing changes between two trees is an instance of the tree diffing problem and has been studied by theoretical computer science [20]. Depending upon the number of moves required to transform one tree into another, some instances of tree diff are NP-hard [21]. Nonetheless, researchers have investigated heuristics such subtree hashing, and even using XML IDs to align subtrees between two versions of a structured document and generate an edit script [22], [21].

Furthermore Tekli et al. in a comprehensive 2009 review of XML similarity note that a future research direction in the field woiuld be to explore similarity methods that compare "not only the skeletons of XML documents ... but also their information content" [23]. Our Hierarchical Diff tool, expresses edit scripts both in terms of structural changes (registry subtrees) as well as changes to the content within tree nodes (registry key data values). Other researchers have looked at techniques to compare XML trees for version control [24], and to compare Puppet network configuration files based upon their abstract syntax trees [25].

VI. Conclusion

In this paper, we have proposed two tools to help utilities and auditors use network and software configurations to generate evidence for a NERC CIP compliance audit. Specifically, our Context-Free Grep and Hierarchical Diff tools, will help practitioners to demonstrate compliance with NERC CIP-003-4, CIP-005-4a, and CIP-010-1. Our planned evaluation of correctness, performance, and usability will help to ensure that these tools, once built, actually help reduce the man hours and costs involved in CIP compliance audit.

Acknowledgment

The authors would like to thank Sergey Bratus, and Jun Ho Huh for their discussions of these tools and their applicability to power system networks. This work was supported in part by the TCIPG project from the DOE (under grant DE-OE0000097). Views are the authors' alone.

References

[1] X. Sun, Y. W. Sung, S. Krothapalli, and S. Rao, "A systematic approach for evolving VLAN designs," in *In Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM 2010)*. IEEE Computer Society, March 2010, pp. 1–9.

[2] Y.-w. E. Sung, S. Rao, S. Sen, and S. Leggett, "Extracting network-wide correlated changes from longitudinal configuration data," in *In Proceedings of the 10th Passive and Active Measurement Conference (PAM 2009)*. unknown, April 2009, pp. 111–121.

[3] D. Caldwell, S. Lee, and Y. Mandelbaum, "Adaptive parsing of router configuration languages," in *Internet Network Management Workshop, 2008. (INM 2008)*. IEEE, October 2008, pp. 1–6.

[4] P. Inglesant, M. A. Sasse, D. Chadwick, and L. L. Shi, "Expressions of expertness: The virtuous circle of natural language for access control policy specification," in *In Proceedings of the 4th Symposium on Usable Privacy and Security (SOUPS '08)*. ACM, July 2008, pp. 77–88.

[5] "NERC CIP reliability standards," 2011, retrieved November 11, 2011 from http://www.nerc.com/page.php?cid=2%7C20.

[6] G. A. Weaver, N. Foti, S. Bratus, D. Rockmore, and S. W. Smith, "Using hierarchical change mining to manage network security policy evolution," in *In Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (HotICE 2011)*. USENIX Association, March–April 2011, p. unknown.

[7] J. Jaakkola and P. Kilpelainen, "Using sgrep for querying structured text files," in *In Proceedings of SGML Finland 1996*. unknown, October 1996, p. unknown.

[8] "Windows PowerShell," retrieved February 3, 2012 from http://technet.microsoft.com/en-us/library/bb978526.aspx.

[9] R. Pike, "Structural regular expressions."

[10] Z. Grunschlag, "cfgrep - context free grammar egrep variant," 2011, retrieved November 11, 2011 from http://www.cs.columbia.edu/ zeph/software/cfgrep/.

[11] "Coccinelle: A program matching and transformation tool for systems code," 2011, retrieved November 11, 2011 from http://coccinelle.lip6.fr/.

[12] D. Plonka and A. J. Tack, "An analysis of network configuration artifacts," in *The 23rd Conference on Large Installation System Administration (LISA '09)*. USENIX Association, November 2009, p. unknown.

[13] "Open Source Tripwire," retrieved February 3, 2012 from http://sourceforge.net/projects/tripwire/.

[14] "TripWire," retrieved February 3, 2012 from http://www.tripwire.com/.

[15] "SunView Software, ChangeGear," retrieved February 3, 2012 from http://www.sunviewsoftware.com/.

[16] "BMC Remedy IT Service Management," retrieved February 3, 2012 from http://www.bmc.com/solutions/itsm/it-service-management.html.

[17] "Download WinDiff," retrieved February 3, 2012 from http://www.grigsoft.com/download-windiff.htm.

[18] "WinMerge," retrieved February 3, 2012 from http://winmerge.org/.

[19] "TkDiff," retrieved February 3, 2012 from http://tkdiff.sourceforge.net/.

[20] P. Bille, "A survey on tree edit distance and related problems," *Theoretical Computer Science*, vol. 337, no. unknown, p. unknown, June 2005.

[21] G. Cobéna, S. Abiteboul, and A. Marian, "Detecting changes in XML documents," in *In Proceedings of the 18th International Conference on Data Engineering*. IEEE, February and March 2002, pp. 41–52.

[22] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change detection in hierarchically structured information," in *In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD '96)*. ACM, June 1996, pp. 493–504.

[23] J. Tekli, R. Chbeir, and K. Yetongnon, "An overview on XML similarity: Background, current trends and future directions," *Computer Science Review*, vol. 3, no. 3, pp. 151–173, August 2009.

[24] S. Apel, J. Liebig, C. Lengauer, C. Kastner, and W. R. Cook, "Semistructured merge in revision control systems," in *In Proceedings of the Fourth International Workshop on Variability Modeling of Software Intensive Systems (VaMoS 2010)*. University of Duisburg-Essen, January 2010, pp. 13–20.

[25] B. Vanbrabant, P. Joris, and J. Wouter, "Integrated management of network and security devices in it infrastructures," in *The 25th Conference on Large Installation System Administration (LISA '11)*. USENIX Association, December 2011, p. unknown.