AI-Assisted Controls Change Management for Cybersecurity in the Cloud

Harshal Tupsamudre[1], Arun Kumar[1], Vikas Agarwal[1], Nisha Gupta[1], Sneha Mondal[2*]

[1] IBM Research, India
[2] Google, India
Harshal.Tupsamudre@ibm.com,{kkarun, avikas, nisgup7}@in.ibm.com, sneha.12392@gmail.com

Abstract

Webscale services dealing with sensitive content are increasingly being deployed in public and hybrid cloud environments. At the same time, the impact of security breaches have also increased manifold averaging at USD 3.86M per data breach To tackle such increasing risks, regulations and security frameworks are defined that an organization must comply with. Most of these frameworks are published in natural language text that run into hundreds of pages resulting into thousands of requirements and controls. When these frameworks undergo revisions, understanding the changes, and interpreting their impact consumes huge amount of time, effort and resources.

In this paper, we propose a change management system that supports SMEs with AI-assisted automation of this extremely manual and time consuming activity. Specifically, we introduce the concept of live crosswalks – a framework that models complex relationships among security and compliance documents along with associated operations to manage the change. It uses natural language processing (NLP) and algorithmic techniques to transform the current document-driven, highly manual process into a data-driven interactive intelligent system. We present the overall design and demonstrate its efficacy over several hundreds of diversified controls through experimental evaluation.

Introduction

Security and compliance requirements are typically specified in natural language documents and published by regulatory bodies either in PDF or WORD format. For instance, PCI-DSS (PCI 2018) is a regulation in financial industry dealing with payment cards, and HIPAA (HIPAA 1996) is a regulation in medical domain dealing with health insurance. These documents typically run into hundreds of pages of text that need to be read, understood and interpreted by compliance Subject Matter Experts (SMEs). In some cases, the publishers also make a structured format such as Excel, XML, YAML or JSON available.

To assist organizations in meeting cybersecurity requirements of these regulations, common security controls are defined by organizations such as NIST and CIS. They have defined and published control definition baselines such as the NIST 800-53 (NIST 2013) and the CIS Security Controls (CISC 2019). The cybersecurity requirements in regulations are mapped to these common security controls. This mapping is called a *crosswalk*.

As an example, Figure 1 shows a crosswalk from NIST Cybersecurity Framework's (CISA 2020) data security controls to a bunch of requirements from regulations including PCI-DSS, COBIT, etc. More concretely, PCI-DSS 3.2.1 requirement 10.6 states: *"Review logs and security events for all system components to identify anomalies or suspicious activity"*. This maps to PR.DS-5 in the NIST CSF (i.e. *"Protection against data leaks are implemented"*) as shown in the figure. The crosswalk helps organizations understand what needs to be implemented to meet those regulatory requirements. Second, they act as a harmonizer across regulations so that an organization does not need to implement the same control again.

SUBCATEGORY	INFORMATIVE REFERENCES[3]
PR.DS-5: Protections against data leaks are implemented.	• **CIS CSC** 13 • **COBIT 5** APO01.06, DSS05.04, DSS05.07, DS • **ISA 62443-3-3:2013** SR 5.2 • **ISO/IEC 27001:2013** A.6.1.2, A.7.1.1, A.7.1.2, A.9.1.1, A.9.1.2, A.9.2.3, A.9.4.1, A.9.4.4, A.9.4 A.11.1.5, A.11.2.1, A.13.1.1, A.13.1.3, A.13.2.1 A.14.1.2, A.14.1.3 • **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-8, SC-13, SC-31, SI-4 • **PCI DSS v3.2.1** 10.6
PR.DS-6: Integrity checking mechanisms are used to verify software, firmware, and information integrity.	• **CIS CSC** 2.3 • **COBIT 5** APO01.06, BAI06.01, DSS06.02 • **ISA 62443-3-3:2013** SR 3.1, SR 3.3, SR 3.4, S • **ISO/IEC 27001:2013** A.12.2.1, A.12.5.1, A.14.1 • **NIST SP 800-53 Rev. 4** SC-16, SI-7 • **PCI DSS v3.2.1** 11.5

Figure 1: A crosswalk example

A recent report from Thomson Reuters (Hammond and English 2019) indicates that the number of regulatory bodies tracked by them has continued to grow over the years, surpassing the count of 1000 in 2019, with an average of 220 regulatory updates per day across industry sectors. On the other hand, the current state-of-the-art at most organiza-

*The work was done when the author was an employee at IBM

tions is to have legal SMEs employed in-house or hired from consulting firms. They read the regulatory documents, identify additions or changes, and then interpret their impact for the organization. The information is exchanged through textual documents. Given the scale of such regulatory changes and often constrained availability of SMEs and Compliance officers, it has become a necessity to automate this process to the extent possible.

To address the scaling problem, we propose *live crosswalks* construct which extends the notion of a crosswalk such that the mapping is not just across a set of pairs of regulatory documents but across multiple layers. Extending the example shared earlier, through the crosswalk in Fig 1), PR.DS-5 can be further mapped to CIS Control 13 (CISC 2019) (shown as CIS CSC 13 in Fig 1). CIS CSC 13 deals with Data Protection and has several sub-controls to ensure the privacy and integrity of sensitive information. One of them is 13.4: *"Monitor and Block Unauthorized Network Traffic"* for detection and another one 13.5: *"Only Allow Access to Authorized Cloud Storage or Email Providers"* for protection. Both of these together provide coverage to satisfy PR.DS-5. To enforce such requirement, various software and systems implement technical rules, such as those specified in CIS Benchmarks (CISB 2021). These can be implemented using scripting languages and frameworks such as Ansible[1]. This end-to-end mapping makes it possible to determine which piece(s) of code needs to execute on a device or service to meet a specific regulatory requirement.

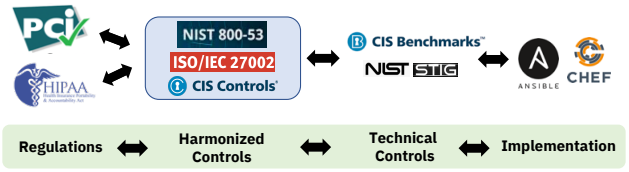

Figure 2: Crosswalk representation spanning across multiple layers of security and compliance documents

Figure 2 depicts a high level view of how two regulatory documents connect to implementation via harmonized controls. The mappings in a live crosswalk stay 'live' i.e., can be updated semi-automatically when any document in the crosswalk evolves to a new version. While we use the context of IT security and compliance, the overall approach is applicable to general internal controls compliance as well.

Problem Statement

In this section, we formalize the concept of *live crosswalks* and define the problem of realizing it and maintaining it.

Let's say a first-layer security and compliance document $\mathcal{X}_1$ maps into a second-layer document $\mathcal{X}_2$, and over a few more mappings across documents such as $\mathcal{X}_2 \rightarrow \mathcal{X}_3 \rightarrow \ldots \rightarrow \mathcal{X}_n$ maps onto code at $\mathcal{X}_n$, then we say that there is an end-to-end crosswalk $\mathcal{C}$ from $\mathcal{X}_1$ to $\mathcal{X}_n$. Note that mapping the elements of one security and compliance document ($\mathcal{X}_i$) into that of another ($\mathcal{X}_j$), leads to a *pair-level crosswalk*.

[1]https://www.ansible.com/

A security and compliance document can be thought of as a set of control elements, each element embodies some technical requirement. If $\mathcal{X} = (x_1, x_2, \ldots x_m)$ denotes the set of control elements in document $\mathcal{X}$, and $\mathcal{Y} = (y_1, y_2, \ldots y_n)$ denotes the set of control elements in document $\mathcal{Y}$, then a pair-level crosswalk between source document $\mathcal{X}$ and target document $\mathcal{Y}$ is a mapping $\mathcal{M} : \mathcal{X} \rightarrow \mathcal{Y}$, $\mathcal{M} = (m_1, m_2, \ldots m_n)$ such that $m_l = (x_k, \{y_1, y_2, \ldots, y_n\})$ where $x_k \in \{\mathcal{X} \cup \phi\}$ and $y_{1..n} \in \{\mathcal{Y} \cup \phi\}$. Intuitively, a mapping from one document to another consists of a set of one-to-many relations that are uni-directional wherein each element of the source could potentially map into several elements of target. These mappings, i.e. pair-level crosswalks, can either be created manually by human experts or created semi-automatically using AI. Techniques for semi-automated mapping creation have been demonstrated in (Agarwal et al. 2021) for regulatory requirements and in (Adam et al. 2019) for compliance technical checks.

Note that, over a few hops of the crosswalk, a single first-level regulatory requirement (*i.e.* at $\mathcal{X}_1$) can fan-out to a large number of target requirements/codes (say, at $\mathcal{X}_n$). At the scale of a few tens of security and compliance documents over multiple hops, manual maintenance of the crosswalk becomes challenging. To tackle this we need to detect changes and assess their impact on the mappings across hops. In this paper, we propose a system that enables automation of change detection.

Essentially, change detection exploits the semantic and structural information in these documents to generate a log of changes across two versions. Formally, given a document $\mathcal{X}_v$ and its subsequent version $\mathcal{X}_{v+1}$, the change detection module outputs a 4-tuple $(\mathcal{A}, \mathcal{R}, \mathcal{I}, \mathcal{M})$, where $\mathcal{A}$ is the set of added elements ($\mathcal{A} \in \mathcal{X}_{v+1}$, $\mathcal{A} \notin \mathcal{X}_v$), $\mathcal{R}$ is the set of removed elements ($\mathcal{R} \notin \mathcal{X}_{v+1}$, $\mathcal{R} \in \mathcal{X}_v$), $\mathcal{I}$ is the set of identical elements. $\mathcal{M} = \{(e_1, e'_1), (e_2, e'_2), \ldots, (e_n, e'_n)\}$, $e_i \in \mathcal{X}_v$ and $e'_i \in \mathcal{X}_{v+1}$, is the set of modified elements. Intuitively, this set contains pairs of elements from the two documents that correspond to each other but are not identical, i.e., $e_i \mathrel{\widehat{=}} e'_i; e_i \not\equiv e'_i$.

Since a crosswalk stitches together pair-level mappings across multiple layers, this method is applied across those layers iteratively in a pair-wise fashion. Though the updates need a review by an SME yet the semi-automation brings substantial savings of time and effort.

Solution Design

In this section, we explain the core design for semi-automatic maintenance of multi-layered crosswalks. Detection of change requires understanding security and regulatory documents at multiple levels - lexical, syntactic as well as semantic.

A security framework or regulatory document is effectively a collection of security controls. Each control element encapsulates a discrete security requirement that a compliant organization must fulfill. The statement of the control is often organized into parts and sub-parts, providing a fine-grained view of individual requirements. As an example, Figure 3 juxtaposes a control, **AC-1**, from NIST 800-53 version 4, and its analogue in the publicly available draft of

version 5. Differences in the two versions are marked by SMEs – equivalent sub-parts are highlighted in identical colors, sub-parts newly introduced in version 5 are tagged. The two documents combined contain more than 1000 controls and manual annotation of changes entails a significant cognitive overhead. The task of the change detection operator is to automatically generate a change-log that captures these incremental differences.

Change Detection

Given a list of m controls that constitutes an older version of a regulatory document, i.e, $X = \{x_1, \ldots, x_m\}$ and a revised list of n controls that constitutes a newer version of the same document, i.e., $Y = \{y_1, \ldots, y_n\}$, the change detection operator detects if a control has been **added**, **removed**, **modified**, or remained **identical** across the two versions. To detect changes, we need a textual similarity metric that assigns similarity scores to pairs of control texts. Jaccard index is one of the most popular metric in NLP that measures the degree of equivalence between a pair of texts. To compute Jaccard index, we first preprocess the text from each control element and extract bag of words by – 1) tokenizing the text into words, 2) converting to lowercase, 3) removing stop words, and 4) applying Wordnet Lemmatizer to transform each word into its base form. We denote a control element using a lower-case letter (x_i) and the corresponding bag of words with an upper-case letter (X_i). Once we extract bag of words X_i from a control element x_i and bag of words Y_j from a control element y_j, we compute the Jaccard index between two sets X_i and Y_j, which is defined as the size of their intersection divided by the size of their union.

$$Jaccard(X_i, Y_j) = \frac{|X_i \cap Y_j|}{|X_i \cup Y_j|} \quad (1)$$

Intuitively, Jaccard index measures the extent of overlap between a pair of control elements. The higher the Jaccard index, more similar are the control elements.

We propose three algorithms for change detection. The first algorithm is greedy that matches a set of control elements in the newer version of a document with a set of elements in an older version. The second algorithm is based on dynamic programming (DP) that takes into consideration the order of control elements in two versions of a document and aligns a sequence of control elements in the newer version with an older version. However, DP algorithm fails to align (or identify) the reordered (out-of-sequence) control elements. Therefore, we give a hybrid algorithm, which first aligns two sequences using the DP algorithm and then employs the Greedy algorithm on the output of the DP algorithm (i.e., set of added and the set of deleted control elements) to identify reordering if any.

Greedy Alignment Algorithm

The Greedy algorithm to identify changes between two versions of a regulatory document is given in Algorithm 1. It matches a control element $y_j \in Y$ with a control element $x_i \in X$ if the Jaccard index between the corresponding bags of words Y_j and X_i is greater than a predefined threshold t (where $0 < t \leq 1$), and maximal among all m control elements of X, i.e., $Jaccard(X_i, Y_j) \geq Jaccard(X_k, Y_j)$, $k \in \{1 \ldots m\}$. The resulting mapping is stored in a dictionary M with the index of a newer control element as key and the index of an older control element as value.

Algorithm 1: Greedy Sequence Alignment Algorithm

Input: Two sets of control elements, $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$, and threshold t

Output: A dictionary M which maps a control element in Y to the maximally similar control element in X based on a threshold value t

```
Initialize a dictionary M
for j=1 to n do
    max_overlap ← t
    k ← −1
    for i=1 to m do
        overlap ← Jaccard(X_i, Y_j)
        if overlap > max_overlap then
            max_overlap ← overlap
            k ← i
        end if
    end for
    if k > −1 then
        M[j] ← k
    end if
end for
return M
```

1. If $y_j \in Y$ is mapped to $x_i \in X$ and $Jaccard(X_i, Y_j) = 1$, then y_j is **identical** to x_i
2. If $y_j \in Y$ is mapped to $x_i \in X$ and $1 > Jaccard(X_i, Y_j) > t$, then x_i is **modified** to y_j
3. If $y_j \in Y$ is not mapped to any $x_i \in X$, then y_j is considered to be a newly **added** element in Y
4. If $x_i \in X$ is not mapped from any $y_j \in Y$, then x_i is considered to be **deleted** from Y

Figure 3 shows the four control elements of AC-1 from NIST 800-53 version 4 (left) and six control elements of AC-1 from NIST 800-53 version 5 (right). The matching computed by the greedy algorithm (t=0.5) is depicted in Figure 4. The four control elements of AC-1 in version 5 are matched with some control element in version 4. The remaining two control elements AC-1.a.1.(b) and AC-1.b are considered to be added to the newer version.

Dynamic Programming Alignment Algorithm

Greedy algorithm can reliably recognize identical, slightly modified and reordered control elements in the newer version of a regulatory document. However, most often the newer version retains a subset of control elements from an older version in the same order, i.e., $(y_{j_1}, \ldots y_{j_q}) \sim (x_{i_1}, \ldots x_{i_q})$, where y_{j_l} is either identical to or a modification of x_{i_l}, $1 \leq l \leq q \leq min(m, n)$. One important limitation of the Greedy algorithm is that it performs matching between the newer control elements and older controls elements without taking into account the order of control elements within the two versions of a document. This could potentially result in incorrect matches. Consider the following two scenarios:

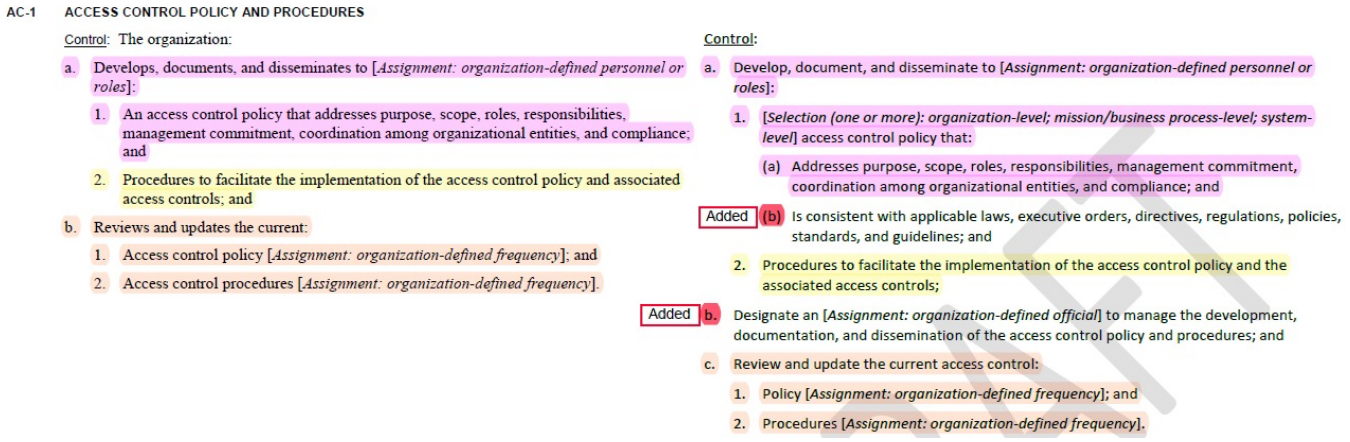

Figure 3: Control AC-1 from NIST 800-53 version 4 (left) and version 5 (right).

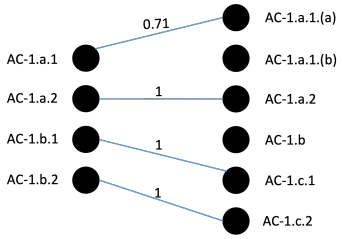

Figure 4: Matching computed by Greedy algorithm (t=0.5) between AC-1 from NIST 800-53 v4 (left) and v5 (right).

- Suppose that $(x_{i_1}, x_{i_2}, x_{i_3})$ are three consecutive control elements in an older version of a document X and $(y_{j_1}, y_{j_2}, y_{j_3})$ are three consecutive control elements in a newer version of the document Y such that $Jaccard(X_{i_1}, Y_{j_1}) \sim 1$, $t > Jaccard(X_{i_2}, Y_{j_2})$ and $Jaccard(X_{i_3}, Y_{j_3}) \sim 1$. Then, it is more likely that y_{j_2} is a modification of x_{i_2} since their neighbouring control elements are similar. However, if there is another control element $x_{i_q} \in X$ such that $Jaccard(X_{i_q}, Y_{j_2}) > t > Jaccard(X_{i_2}, Y_{j_2})$, then the greedy algorithm matches y_{j_2} with x_{i_q}.
- Suppose that (x_{i_1}, x_{i_2}) are two consecutive control elements in X and $(y_{j_1}, y_{j_2}, y_{j_3})$ are three consecutive control elements in Y such that $Jaccard(X_{i_1}, Y_{j_1}) \sim 1$ and $Jaccard(X_{i_2}, Y_{j_3}) \sim 1$. Then, it is more likely that y_{j_2} is a new control element that is added to Y since its neighbouring control elements y_{j_1} and y_{j_3} are similar to x_{i_1} and x_{i_2} respectively. However, if there is another control element $x_{i_q} \in X$ such that $Jaccard(X_{i_q}, Y_{j_2}) > t$ then the greedy algorithm will match y_{j_2} with x_{i_q}.

To address this limitation, we propose a dynamic programming based algorithm that aligns a sequence of n control elements within a newer version of a regulatory document i.e., $Y = (y_1, \ldots, y_n)$, with a sequence of m elements within an older version i.e., $X = (x_1, \ldots, x_m)$. The algorithm computes an optimal alignment based on a cost model (w_a and w_r), with w_a representing the cost of adding a control element in a newer version of a document and w_r representing the cost of removing a control element from an older version of a document. The algorithm computes an alignment score $D[i][j]$ between the first i control elements of X and first j control elements of Y using the recurrence relation defined below:

$$D[i][j] = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0 \\ D[i-1][j] + w_r & \text{if } i > 0 \text{ and } j = 0 \\ D[i][j-1] + w_a & \text{if } i = 0 \text{ and } j > 0 \\ \min \begin{cases} D[i-1][j] + w_r \\ D[i][j-1] + w_a \\ D[i-1][j-1] + (1 - Jaccard(X_{i-1}, Y_{j-1})) \end{cases} & \text{if } i > 0 \text{ and } j > 0 \end{cases} \quad (2)$$

Specifically, the alignment score $D[i][j]$ depends on the following three values :

1. $D[i-1][j] + w_r$: where $D[i-1][j]$ is the score obtained by aligning the first $i-1$ control elements of X, i.e., $(x_1, \ldots, x_{i-1})$ and the first j control elements of Y, i.e., $(y_1, \ldots, y_j)$, and w_r is the cost of **removing** x_i from Y.
2. $D[i][j-1] + w_a$: where $D[i][j-1]$ is the score obtained by aligning the first i control elements of X, i.e., $(x_1, \ldots, x_i)$ and the first $j-1$ control elements of Y, and w_a is the cost of **adding** y_j to Y.
3. $D[i-1][j-1] + (1 - Jaccard(X_i, Y_j))$: where $D[i-1][j-1]$ is the score obtained by aligning the first $i-1$ control elements of X, i.e., $(x_1, \ldots, x_{i-1})$ and the first $j-1$ control elements of Y, i.e., $(y_1, \ldots, y_{j-1})$, and $(1 - Jaccard(X_i, Y_j))$ indicates dissimilarity between the control elements y_j and x_i. Note that when $Jaccard(X_i, Y_j) = 1$, then $(1 - Jaccard(X_i, Y_j)) = 0$ and the control element y_j is **identical** to x_i, otherwise y_j is considered to be a **modification** of x_i.

The last cell in the matrix $D[m][n]$ represents the optimal alignment score between the m control elements of X and n control elements of Y. Once matrix D is computed, one can back trace from $D[m][n]$ to $D[0][0]$ to identify a sequence of operations used to transform an older version of a document to a newer version.

Older AC-1 \ Newer AC-1	AC-1.a.1.(a)	AC-1.a.1.(b)	AC-1.a.2	AC-1.b	AC-1.c.1	AC-1.c.2
0	1	2	3	4	5	6
AC-1.a.1 1	0.29	1.29	2.29	3.29	4.29	5.29
AC-1.a.2 2	1.29	0.94	1.29	2.29	3.29	4.29
AC-1.b.1 3	2.29	1.94	1.68	1.99	2.29	3.29
AC-1.b.2 4	3.29	2.94	2.68	2.46	2.19	2.29

Figure 5: The score matrix D computed by DP algorithm with $w_a = w_r = 1$. The arrows starting at $D[4][6]$ and ending at $D[0][0]$ indicate the optimal sequence of operations required to transform AC-1 from NIST 800-53 v4 to v5.

The alignment matrix D computed by dynamic programming algorithm between four control elements of AC-1 from NIST 800-53 version 4 (left) and six control elements of AC-1 from NIST 800-53 version 5 (right) is shown in Figure 5. For the purpose of demonstration, both costs w_a and w_r are set to 1. A sequence of operations used to transform AC-1 from version 4 to AC-1 from version 5 can be identified by starting at $D[m][n] = D[4][6]$ and back tracing to $D[0][0]$.

We note that the algorithm used in the diff utilities (GNU and Git) is also based on dynamic programming [2]. However, there are two primary differences between our approach and the diff algorithm.

1. Our approach computes optimal alignment between two sequences of lines where each line is represented as a bag of words. The diff algorithm also aligns two sequences of lines, however it treats each line as a sequence of words.
2. Our approach uses Jaccard index to compute similarity between two lines. Hence, the similarity score varies from 0 to 1. The similarity score within the diff algorithm is either 0 or 1.

In case of regulatory documents, most of the controls in a newer version are modification of controls from an older version (in some cases modification is slight) which makes our proposed algorithm more suitable for change detection.

Hybrid Algorithm

The DP based algorithm fails to align (or identify) the reordered (out-of-sequence) control elements. The Greedy and DP algorithms can be combined to identify reordered control elements in a newer version of a document (Hybrid Algorithm 2). The intuition is that if some control element is reordered, then the DP algorithm would label it as added in the newer document (since it would not be aligned to any control element in the older document) and removed in the older document (since it would not be aligned with any control element in the newer document). Therefore, if the Greedy algorithm is executed on the sets of added and removed control elements, it can help us identify such reordered pairs.

Evaluation

In this section, we report the performance of the change detection operator by (i) comparing the results against human annotations on a few security and compliance documents and (ii) evaluating our algorithm across different kinds of security and compliance documents. Overall, we found that both DP and Hybrid (DP+Greedy) algorithms outperform Greedy algorithm in terms of recall and precision. As the number of identical controls across all regulatory documents was small (less than 10% of total controls), and both recall and precision for all algorithms were almost 1, we do not further discuss results pertaining to the identical controls.

[2] gnu.org/software/diffutils/manual/html_node/Overview.html

Algorithm 2: Hybrid Algorithm (DP+Greedy)

Input: Two sequences of control elements, $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$, and threshold t, addition cost w_a and removal cost w_r

Output: A label assigned to each control element in X and Y indicating if the control element has been added, deleted, modified or remained identical

1: Run the DP algorithm and compute the optimal alignment between two versions of the document X and Y, according to the cost model (w_a and w_r)
2: Extract the sets of added, removed, modified and identical controls as identified by the DP Algorithm
3: Run the Greedy algorithm (Algorithm 1) on the sets of added and removed controls extracted in the previous step to identify **reordered** controls if any

NIST 800-53 First, we compare versions 4 and 5 of NIST 800-53 To obtain a complete list of changes , we rely on a changelog authored by the MITRE corporation, also available at NIST's website[3]. The precision and recall of the proposed algorithms for 1061 coarse-grained control elements is provided in Table 1. The threshold t required for the Greedy algorithm is set to 0.5 and the costs w_a and w_r are set to 0.4. We chose these values as they were found to be yielding better results empirically (Figure 6).

The overall recall and precision of the DP algorithm (0.935 and 0.886) exceeds the overall recall and precision of the Greedy algorithm (0.877 and 0.805). We observed that 49 controls in the version 5 were reordered which were labelled incorrectly by the DP algorithm. The Hybrid algorithm improves the recall of modified controls and the precision of added and removed controls, which in turn improves the overall recall to 0.957 and overall precision to 0.941.

Regulation	Total	Added	Removed	Modified
NIST 800-53	1061	237	52	761
Greedy ($t = 0.5$)				
Recall	**0.877**	0.937	0.808	0.862
Precision	**0.805**	0.696	0.284	0.966
DP (w_a=w_r=0.4)				
Recall	**0.935**	0.992	0.923	0.917
Precision	**0.886**	0.799	0.432	0.991
Hybrid				
Recall	**0.957**	0.966	0.808	0.963
Precision	**0.941**	0.909	0.6	0.983

Table 1: Recall and Precision of Greedy, DP, and Hybrid algorithms for two versions of NIST 800-53 document.

[3] https://csrc.nist.gov/CSRC/media/Publications/sp/800-53/rev-5/draft/documents/sp800-53r5-draft-fpd-comparison-with-rev4.xlsx

Regulation	Total	Added	Removed	Modified
PCI-DSS	745	244	118	353
Greedy ($t = 0.5$)				
Recall	**0.737**	0.881	0.889	0.572
Precision	**0.624**	0.63	0.415	0.789
DP (w_a=w_r=0.4)				
Recall	**0.91**	0.943	0.898	0.887
Precision	**0.886**	0.874	0.768	0.934
Hybrid				
Recall	**0.894**	0.902	0.788	0.915
Precision	**0.899**	0.913	0.816	0.907

Table 2: Recall and Precision of Greedy, DP, and Hybrid algorithms for two versions of PCI-DSS regulatory document.

PCI-DSS Next, we used our change detection algorithms on two versions of PCI-DSS regulatory document (v3.2.1 and the latest draft) which contains 15 requirements (including 3 appendices). Each requirement has sub-requirements, each containing description, test procedures and supplementary guidance. There are over 1500 elements. We manually created the list of changes for 745 elements to serve as ground truth. The overall recall and precision of the DP algorithm exceeds that of the Greedy algorithm (Table 2). Also, as the number of reordering were few, the recall of the DP algorithm is better than the Hybrid algorithm.

CFR 12 Part-25 Finally, we evaluated our change detection algorithms on 12 CFR Part 25[4], a U.S. Government regulation around Community Reinvestment Act in Banking industry. We took the help of an SME to manually generate the list of changes between two releases to serve as ground truth for our recall and precision computation. The overall recall of the Greedy and DP algorithm is similar because more than 50% of the elements were newly added to the latest version of the document (Table 3). Overall, the Hybrid algorithm performs slightly better among all algorithms.

Regulation	Total	Added	Removed	Modified
CFR 12 Part-25	811	479	163	91
Greedy ($t = 0.5$)				
Recall	**0.921**	0.956	0.945	0.67
Precision	**0.895**	0.944	0.823	0.772
DP (w_a=w_r=0.3)				
Recall	**0.93**	0.994	0.988	0.615
Precision	**0.879**	0.903	0.767	0.933
Hybrid				
Recall	**0.94**	0.96	0.945	0.77
Precision	**0.93**	0.958	0.895	0.816

Table 3: Recall and Precision of Greedy, DP, and Hybrid algorithms for two versions of CFR 12 Part-25 document.

Selecting Parameters The recall of the Greedy algorithm for different values of threshold t and the recall of the Hybrid algorithm for different values of costs w_a and w_r are depicted in Figure 6. Empirically, we observed that the Greedy algorithm performs better across all documents when the threshold t is around 0.5, and the DP algorithm performs better when the costs w_a and w_r are between 0.3 and 0.4. The figure also show that the Hybrid algorithm with parameters $t = 0.5$ and w_a and w_r between 0.3 and 0.4 always performs better than the Greedy algorithm irrespective of the value of t used within the Greedy algorithm.

Intuitively, when both costs w_a and w_r are greater than 0.5, then the DP algorithm tries hard to match unrelated control pairs (x_i, y_j) even if there is no significant overlap between the two, i.e., $Jaccard(X_i, Y_j) \sim 0$. This is because, the total cost of adding y_j and removing x_i is greater than 1 ($w_a + w_r > 1$), whereas the cost of matching (x_i, y_j) is close to 1 since $(1 - Jaccard(X_i, Y_j)) \sim 1$. On the other extreme, if costs w_a and w_r are close to 0, then the DP algorithm labels every element in the newer document as added and every element in the older element as removed even if there is a significant overlap between the two. Therefore, setting w_a and w_r between 0.3 and 0.4 results in a better recall.

Related Work

The survey paper (Papanikolaou, Pearson, and Mont 2011) covers existing work around processing and understanding of regulatory documents. The authors have grouped past research based on various stages in the pipeline like parsing and extraction, semantic representation, implementation/enforcement of regulations etc. The survey also mentions that none of the existing work is able to capture the entire pipeline. While the notion of pair-level regulatory *crosswalks* has existed in several industry domains, the process of creating these is mostly manual relying on human expertise and domain knowledge.

Authors in (Adam et al. 2019) describe a system that uses machine learning to automatically map regulatory requirements across documents. However, they demonstrate this in the context of NIST STiGs and CIS Benchmarks which have overlapping fine grained requirements as compared to a regulatory document such as HIPAA. Mapping across a semantic ontology created from regulations and an existing business process ontology is described in (Sapkota et al. 2016). Given the tough nature of this problem of mapping text across different kinds of documents, this problem is still largely unsolved.

Regulatory change analysis has also been attempted earlier. This includes predicting which portions of regulations are likely to change and which are more likely to be stable (Maxwell, Anton, and Swire 2012), ensuring whether a change proposed has been incorporated in new version (May, Gunter, and Lee 2006) or mapping change impact to business processes (Rudzajs and Buksa 2011). However, the first two appear to mostly employ a manual process, and while third does seem to have automation, it relies on version control system for change analysis and does not expand much on change propagation. In contrast, our system assumes the presence of initial mappings across regulatory documents (usually created by SME's) and automates the task of keeping these up-to-date in the light of changes and provides assistance in determining impact of those changes.

[4]https://www.ecfr.gov/cgi-bin/text-idx?tpl=/ecfrbrowse/Title12/12cfr25_main_02.tpl

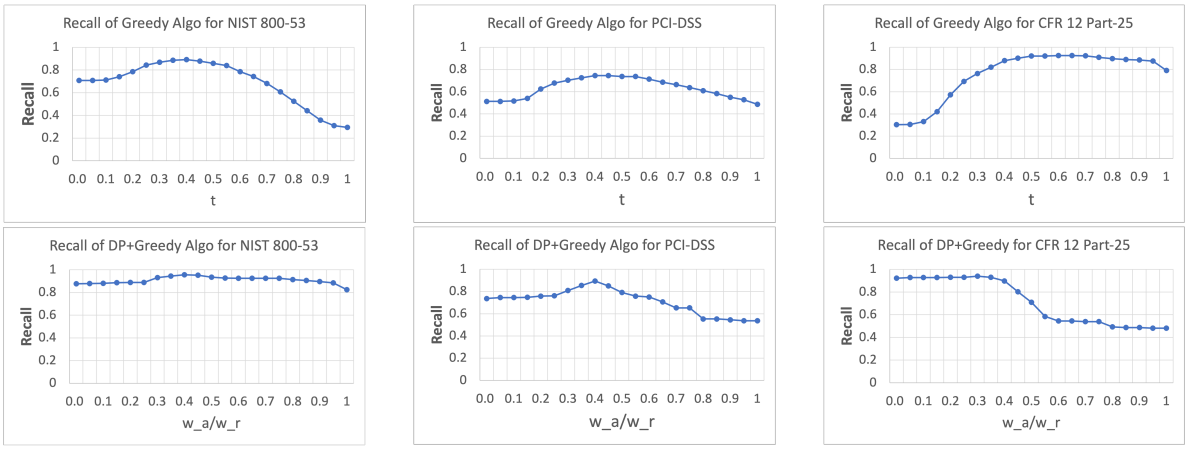

Figure 6: Recall of Greedy algorithm for varying value t and Hybrid algorithm for varying costs w_a and w_r across documents.

Conclusion

We introduced the concept of live crosswalks - a framework to link hundreds of regulatory requirements to thousands of actual control implementations. We focused on identification of change in regulatory and security control documents. We demonstrated that our system performs comparable to human generated change logs available today. Hence, it can assist compliance experts saving their time and effort.

As future work, we intend to provide even better assistance to the experts by incorporating semantic analysis of the identified changes. Further, the value of this work gets amplified when these changes can be connected to mappings (Agarwal et al. 2021) to advise and alert the compliance experts and engineers on impacted policies/controls.

Acknowledgements

The authors would like to thank Kuntal Dey, Aditya Dwivedi, Milton Hernandez, and Sagar Gahalod for their inputs on early parts of this work.

References

Adam, C.; Bulut, M. F.; Hernandez, M.; and Vukovic, M. 2019. Cognitive Compliance: Analyze, Monitor and Enforce Compliance in the Cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 234–242.

Agarwal, V.; Bar-Haim, R.; Eden, L.; Gupta, N.; Kantor, Y.; and Kumar, A. 2021. AI-Assisted Security Controls Mapping for CloudsBuilt for Regulated Workloads. In *IEEE International Conference on Cloud Computing (CLOUD)*.

CISA. 2020. Cyber Resilience Review (CRR): NIST Cybersecurity Framework Crosswalks. https://us-cert.cisa.gov/sites/default/files/c3vp/csc-crr-nist-framework-crosswalk.pdf. Accessed: 2022-01-03.

CISB. 2021. CIS Benchmarks. https://www.cisecurity.org/. Accessed: 2022-01-03.

CISC. 2019. CIS Critical Security Controls. https://workbench.cisecurity.org/files/2312/download/2608. Accessed: 2022-01-03.

Hammond, S.; and English, S. 2019. Report: Cost of Compliance 2019—after 10 years of regulatory change, expect more change. https://blogs.thomsonreuters.com/legal-uk/2019/07/23/report-cost-of-compliance-2019-after-10-years-of-regulatory-change-expect-more-change/. Accessed: 2022-01-03.

HIPAA. 1996. HEALTH INSURANCE PORTABILITY AND ACCOUNTABILITY ACT OF 1996. www.govinfo.gov/content/pkg/CRPT-104hrpt736/pdf/CRPT-104hrpt736.pdf. Accessed: 2022-01-03.

Maxwell, J. C.; Anton, A. I.; and Swire, P. 2012. Managing changing compliance requirements by predicting regulatory evolution. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, 101–110.

May, M.; Gunter, C.; and Lee, I. 2006. Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, volume 2006, 85–97.

NIST. 2013. Security and Privacy Controls for Federal Information Systems and Organizations, NIST Special Publication 800-53, Revision 4. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf. Accessed: 2022-01-03.

Papanikolaou, N.; Pearson, S.; and Mont, M. 2011. Towards Natural-Language Understanding and Automated Enforcement of Privacy Rules and Regulations in the Cloud: Survey and Bibliography. In *Secure and Trust Computing, Data Management, and Applications*, volume 187, 166–173.

PCI. 2018. Payment Card Industry (PCI) Data Security Standard version 3.2.1. https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf. Accessed: 2022-01-03.

Rudzajs, P.; and Buksa, I. 2011. Business Process and Regulations: Approach to Linkage and Change Management. In *Perspectives in Business Informatics Research*, volume 90, 96–109.

Sapkota, K.; Aldea, A.; Younas, M.; Duce, D.; and Banares-Alcantara, R. 2016. Automating the semantic mapping between regulatory guidelines and organizational processes. *Service Oriented Computing and Applications*, 10.