Secure Business Intelligence Markup Language (secBIML) for the Cloud

Aspen Olmsted
Fisher College
Department of Computer Science, Boston, MA 02116
e-mail: aolmsted@fisher.edu

***Abstract*— Enterprise organizations have relied on correct data in business intelligence visualization and analytics for years. Before the adoption of the cloud, most data visualizations were executed and displayed inside enterprise applications. As application architectures have moved to the cloud, many cloud services now provide business intelligence functionality. The services are delivered in a way that is more accessible for end-users using web browsers, mobile devices, data feeds, and email attachments. Unfortunately, along with all the benefits of the cloud business intelligence services comes complexity. The complexity can lead to slow response times, errors, and integrity issues. An information technology department or service provider must get ahead of the problems by automating the execution of reports to know when availability or integrity issues exist and dealing with those issues before they turn into end-user trouble tickets. In this paper, we develop an Extensible Markup Language programming language that allows execution against many cloud documents and business intelligence services. The language enables issues to be proactively discovered before end-users experience the problems.**

Keywords-Business Intelligence; Cloud Computing; Heterogeneous Data

I. INTRODUCTION

Forrester Research defines business intelligence as "a set of methodologies, processes, architectures, and technologies that transform raw data into meaningful and useful information used to enable more effective strategic, tactical, and operational insights and decision-making [1]. For today's businesses, this mainly takes shape through data visualization (tabular and charts), business documents, data mining, customer interaction automation, and email marketing.

Data visualizations have been developed by enterprises for decades to allow users to analyze their data in tabular or chart format. The visualizations change based on runtime prompts that filter the data displayed in the visualization. Data from separate Online Transaction Processing (OTP) systems are often aggregated into data warehouses to allow visualizations that span data from multiple source systems. Unfortunately, little tooling was provided to ensure the visualizations guaranteed the required availability and integrity. This paper describes our work in developing a programming language to help an organization with these issues. We call our programming language, Secure Business Intelligence Markup Language (secBIML). Our programming language secBIML allows an organization to script the correctness requirements and receive proactive notification of security issues.

Data mining allows an enterprise to discover new knowledge from their OTP data using data science algorithms. Unfortunately, the integrity of the source data is often ignored, leading to new knowledge derived from bad information. Utilizing secBIML, an organization can script the correctness requirements into comparison tables and receive proactive notification of integrity issues in the source data.

Many cloud application providers sell customer relationship management (CRM) and email marketing solutions and advertise their ability to automate interactions with customers based on changes in the data. Unfortunately, little attention is provided to how the data is aggregated and the availability and integrity of the information that is used as the source of the automation or email marketing. Our programming language secBIML can alert an organization of issues so they can proactively solve the problems with the correctness of the data used in the process.

The organization of the paper is as follows. Section II describes the related work and the limitations of current methods. In Section III, we describe the elements in the secBIML programming language. Section IV provides the motivating example behind our work. Section V describes how we developed our runtime engine. Section VI drills into the data we gather in our experimentation with data visualizations. Section VII investigates the tests we used in our experimentation with business document integrity. Section VIII describes the test implementation used in our experimentation with business email integrity. We conclude in Section IX and discuss future work.

II. RELATED WORK

The large corporate cloud providers such as Microsoft, Google, Amazon, and IBM hold many patents in the domain of recognizing application availability. The patents are designed for business to consumer websites where there is less control than we have in our enterprise BI environment. The lower level of control stems from the client machines in business to consumer architectures are unknown to the provider. One example of such a patent is from Letca et al.[7]. In the patent, Microsoft inserts a stub between the calling client and the web application. The stub gathers performance data as the user is using the web application. Unfortunately, with such a solution, a flaw in the stub can reduce the availability of the service. In our work, we utilize the network during off-hours for the enterprise to gather application data. The information gathered informs the information technology staff of priorities to proactively solve problems before they are filed as end-user trouble tickets.

Codd [1] describes integrity constraints in his original work on relational databases. Codd's original work assumed the data sources are two-dimensional tables that are

 ISBN: 978-1-61208-778-8

normalized to eliminate redundancy. Codd's ideas made it into most online transaction processing (OTP) databases but never made it to the BI or document level. The data layer behind most BI architectures often increases availability by allowing dirty data through the use of database hints. In our work, we are looking for integrity errors by defining constraints in the document testing language itself and not in the data layer behind the documents.

TABLE I. secBIML TAGS.

Tag	*Type*	*Parent*
Credential	*Statement*	
Report	*Statement*	*Credential*
Execution	*Statement*	*Report*
Parameter	*Statement*	*Execution*
Alert	*Statement*	*Comparison or Execution*
RestAction	*Statement*	*Alert*
DBAction	*Statement*	*Alert*
LogAction	*Statement*	*Alert*
Reference	*Expression*	*Comparison*
Comparison	*Expression*	*Comparison*
Literal	*Expression*	*Comparison*
ActionValue	*Expression*	*RESTAction, DBAction or LogAction*

Many security software vendors offer a web application security scanner. These scanners try to break a web application to find common vulnerabilities such as cross-site scripting and SQL injection. Khoury et al. [8] evaluate the state of art black-box scanners that support detecting stored SQL injection vulnerabilities. Our work utilizes white box testing to find vulnerabilities in access control on both the docuememt or data element level.

III. LANGUAGE ELEMENTS

The programming language secBIML is defined in Extensible Markup Language (XML) with elements expressing the statements and expressions. Attributes or child elements express the parameters to the statements and expressions. Elements are identified in a SecBIML program as a start-tag, which gives the element name and attributes, followed by the content, followed by the end tag. Start-tags are delimited by `<, ' and `>'; end tags are delimited by `</' and `>'. TABLE I shows a breakdown of the tags available in the secBIML language.

```
<report name="eventbyhour"
server=https://logireports.fi.edu?rdName=Reports.Admissio
ns.Event_ByHour credential=”bilogin”/>

<execution name="eventbyhourjuly"
report="eventbyhour"/>

<parameter execution="eventbyhourjuly"
name="BeginDate" value="07/01/2019"/>

<parameter execution="eventbyhourjuly"
name="EndDate" value="07/31/2019"/>
```

Figure 1. Example Report, Execution and Parameter Declaration Elements

A. Statement Tags

secBIML syntax is made up of declarative statements that define one of eight statement entities: credential, report, execution, parameter, alert, RESTaction, DBaction, and LOGaction. Figure 1 shows an example set of declarations to define a single implementation of a report with two runtime parameters. The parameters are set for a date range of the entire month of July 2019. The following is the set of language elements currently supported by secBIML:

- Credentials – The credential tag declare
- Reports – The report tag states the details on the server and the name of a specific report that is tested.
- Executions – The execution tag declares a specific test case for a report.
- Parameters – The parameter tag declares the runtime values used in the test of a specific execution.
- Alerts – The alert tag defines the data that is tested specify actions to take on failures. Actions can add tuples to a datastore, send emails, or call web-services. Parent tags for Alerts can either be comparison entities or execution entities.
- RESTactions – The RESTaction tag defines actions that call to web-services. The web-services call has the key-value pairs in the delivery.
- DBActions – The DB actions tag defines tuples written to a database table. The key in the key-value pair returned from the ActionValue entity matches with a table column, and the value is inserted in the tuple.
- Logactions – The "Logaction" tag is used to define values written to a log file.

B. Expression Tags

Expressions in the secBIML are entities where the syntax returns one of five different data types: list, boolean, numbers, text, or key-value pairs. Expressions are used to find a specific value in the report output, aggregate a set of values in the report output, express literal values, or define what data is sent to actions. Operators can combine expressions to be used in complex relational comparisons. There are four expression elements that return values in the secBIML language. The four elements are reference, literal, comparison, and ActionValue. We document the four elements below:

- References – The reference tag allows for the retrieval of a value from a report. The values are

specified in the output by the hypertext markup language (HTML) id or a position in an HTML table. The type attribute allows values to be accumulated, counted, or averaged. The selector attribute is used to aggregate the values in a row or column within an HTML table. Selectors are patterns that match against elements in a tree and are the primary method used to select nodes in an XML document. secBIML supports CSS Level 3 selectors [9].

- Literals - The literal tag allows the expression of a constant value. Literal tags are used when comparing a value in a report to a static value defined at the time the test is created.
- Comparisons – The comparison tag allows values to be compared. A comparison tag returns a Boolean value based on the results of the comparison. The comparison tag requires an operator attribute to specify the comparison operation type. There are six supported comparison operator abbreviations: equal (EQ), not equal (NE), greater than (GT), less than (LT), great than or equal to (GE), less than or equal to (LE). The value in the parenthesis is the abbreviated version of the comparison operator. Figure 2 shows the declaration of a reference to a cell within the last row of a table in the report output. A comparison of a literal value of 23,201 is made to the value on the report, and if the data is different, a REST web service call is made to save the data. By default, actions include the data used in the comparison, the name, the compared values, and a timestamp marking the comparison evaluation time.
- ActionValues – The ActionValue tag allows the delivery and storage of key-value pairs in response to the alert. The type attribute defaults to a comparison name but can be a comparison, reference, literal, or execution result. There are two available values from the execution results; the HTTP status and the duration of the execution.

C. *Attributes & Child Elements*

In both the statement and expression tags, white space and attributes are allowed between the element name and the closing delimiter. An attribute specification consists of an attribute name, an equal sign, and a value. A child element is a tag fully enclosed between the open tag of another statement or expression and the matching closing tag. White space is allowed around the equal sign. Attributes and child elements in the secBIML syntax specify the parameters in the statements or the expressions. It is possible to express any parameter either through an attribute or through a child element. The expression of a child element allows for more complicated parameters including collections of values. Figure 2 shows how the RestAction and ActionValue entities can be rolled up into attributes. Attribute parameters are similar to read but do not allow for more than one value of the same attribute type.

```
<reference name="attendancetotal"
execution="eventbyhourjuly" type="sum"
selector=”#attendance”/>

<comparison name="totalattendance"
reference="attendancetotal" literal="23201"/>

<alert comparison="totalattendance"
action="writeerror"/>

<action name="writeerror" restaction="
http://https.logireports.fi.edu/saveerror"/
actionvalue=”totalattendance”>
```

Figure 2. Example Alert and Supporting Elements

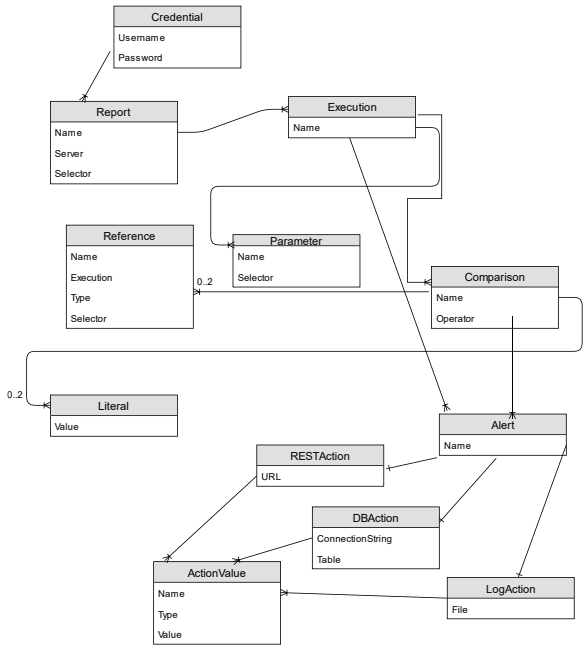

Figure 3. secBISQL ER Diagram

IV. SECBISQL& MOTIVATING EXAMPLE

To facilitate the usage of the programming language by non-programmers, we developed a version of the language that has the tags stored in a SQL database. The SQL version is called secBISQL. The semantics of the two versions secBIML and secBISQL are identical. The difference is in how the programming language is stored in the source format. Figure 3 shows an entity-relationship diagram (ER) for secBISQL.

secBISQL was developed for The Franklin Institute (TFI) in Philadelphia, PA [10] to allow them to identiy availability and integrity errors in their business intelligence operations. In their business intelligence operations, TFI had one hundred and twenty custom reports that ran in the cloud using a business intelligence tool name Logi Analytics [11]. The custom reports were developed over many years by several

different developers. Unfortunately, the end-users were experiencing errors and timeouts throughout the day.

In our first iteration, we used secBISQL to measure the security of data visualizations. We followed this iteration up by experimenting with other generated business documents and communications. The documents we experimented with can be categorized into three primary categories; word processing, presentation, and spreadsheet documents. Each document we looked at had aggregation of values or references to data from business intelligence reports. We also looked at automated emails sent to patrons after activities with the patrons, along with mass emails that were sent for marketing future events to patrons.

For the word processing, presentation, and spreadsheet documents, we utilized Microsoft™ Office 365 [8]. Office 365 is a cloud-based software as a service (SAAS) solution for word processing. To programmatically reference the word processing document, the URL of the office 365 document is added in the entity object as a "report" entity. Comparisons can be defined to compare individual values in the document to other values or aggregated values in the same document or a data visualization. For example, an invoice document laid out in Microsoft™ Word can be verified to ensure that the columns for quantity and amount are equal to the total column. A spreadsheet document has the functionality to aggregate values but a word processing document is often used for the end printed business document because of layout concerns. Integrity checks can be established in secBIML to ensure the word processing data is correct. Values in a business document could also be compared to a source business visualization. Often data is pulled from a data visualization and placed in a flyer or presentation, but that data may change in the source system. secBIML can ensure that data remains correct. This same technique can be used with documents stored in competitive cloud SAAS word processing solution providers such as Google™ GSuite [9].

After tackling the business documents, we looked at emails generated from back-end business transactional data. We were able to retrieve emails from an email service provider (ESP) through the Representational state transfer (REST) application programmer interface (API)s. REST is a software architectural style that defines a set of constraints for Web services creation. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet. The "report" entity was used to specify a REST front end URL, and the parameters were used to call out to the web-service for the specific REST data. The data was then compared to a report that listed the source data consumed in the generation of the email marketing or business automation.

V. RUNTIME ENGINE

The language compiler and execution engine were built using the C Sharp programming language on the .NET Core runtime engine [12]. .NET Core is an open-source, managed execution framework that allows execution on the Microsoft Windows, Linux, and macOS operating systems. The framework is a cross-platform successor to the .NET Framework. The framework allows the implementation of secBIML on any modern operating system.

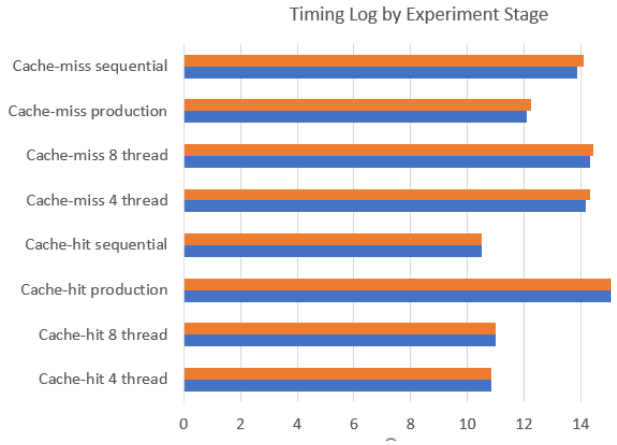

Figure 4. Average Timing

secBIML links to a .NET library named Puppeteer Sharp [13]. Puppeteer Sharp is a .NET port of the Node.JS Puppeteer API [14]. Puppeteer is a Node programming language library that provides a high-level API to control the Chrome browser. Puppeteer allows a program to run the browser headless so that the browser interface is not exposed to the console. This layer of browser execution is critical in the execution of the business intelligence reports to ensure proper execution of JavaScript rendered HTML reports.

VI. EMPIRICAL DATA – DATA VISUALIZATION

In this section, we look at the empirical data we gathered to support our hypothesis that the usage of the secBIML language could increase the security of business intelligence reports and visualizations. To measure the availability of the business intelligence reports, we scheduled one hundred and twenty reports to run overnight in six modes. The six modes were sequential with a cache and without a cache, four

TABLE II SECBIML PRE-TESTING DATA

Data Point	Timing	Executions
Cache-miss sequential	17652	120
Cache-hit sequential	1464	120
Cache-miss 4 thread	20556	120
Cache-hit 4 thread	1824	120
Cache-miss 8 thread	22380	120
Cache-hit 8 thread	2016	120
Cache-hit production	145873	910
Cache-miss production	4864	320

 ISBN: 978-1-61208-778-8

TABLE IV SECBIML BUSINESS DOCUMENT TESTS

Document Type	Count	Internal	External	Initial Integrity	Continuous Integrity
Word Documents	102	24	82	82%	94%
PowerPoint Documents	55	2	53	86%	92%
Excel Documents	1	0	1	100%	100%

concurrent threads with a cache and without a cache, and eight current threads with a cache and without a cache. The tests were run over thirty days, and the average execution is shown in TABLE III. Also, include in the table is the average production data for the same period. The production data was gathered by parsing the web server logs for calls to the business intelligence report.

The reports that exhibit slow behavior were optimized based on the data gathered in the first phase and were optimized, and the experiment was run again for thirty days. TABLE III shows the average timing data collected in the post-optimization period. Figure 4 shows the comparison of the average per report timing for both pre-optimization and post-optimization timing experiments. The data clearly shows that the availability was increased in every mode of data gathering based on the knowledge gathered from the secBIML executions.

VII. EMPIRICAL DATA – BUSINESS DOCUMENTS

In this section, we look at the empirical data we gathered to support our hypothesis that the usage of the secBIML language could increase the security of business documents. We sampled fifty-seven business documents stored as Microsoft™ Office 365 documents. The data was either stored in Word, PowerPoint, or Excel applications. TABLE IV shows the documents used in our tests. The internal and external columns represent the number of tests that we established in each category. The internal tests compare values within the document, and the external tests compare values across documents. The initial integrity column displays the percentage of the correctness of the numbers returned from the first execution of the test. The continuous integrity displays the rate of accuracy over 12 weeks. After the initial test, corrections were applied to the documents, and continuous integrity tests ran nightly. The test demonstrates how often the data changed in the source data. We only found one excel document that had external budget data, and the data was correct and did not change over the 12-week test period. Discovery and setup of tests for business documents was a tedious process. In our future work, we plan to develop a Chrome web browser plugin to allow the automation of the test creation within the document. Nightly executions of the tests for business documents helped to improve the integrity, but trigger-based test execution would be a better solution. Both Microsoft Office 365 and Google GSuite offer API hooks that can be used to launch the test when a document is saved. The test could then run and immediately notify the user of the error. We would also plan to add web browser notifications immediately when an integrity error occurs.

VIII. EMPIRICAL DATA –EMAILS

In this section, we look at the empirical data we gathered to support our hypothesis that the usage of the secBIML language could increase the security of email marketing and business automation. Many CRM and email marketing vendors claim functionality to allow artificial intelligence with email marketing and continuous communication with customers based on business automation. We believe this is a more difficult process than vendors imply. The difficulty comes from the fact that the data used to generate these emails and automation must be accurate and current. So, we wanted to test the correctness of data used in a production system. To measure the integrity of the data, we used an email services provider (ESP) Mailgun [13]. An ESP is a cloud service provider that manages the delivery of email messages. Some vendors provide analytic data on email delivery, such as the number of messages delivered, suppressed, and dropped. Data about the email clients, click-throughs, and unsubscribe data is also maintained. An added benefit of the provider we chose is that a free version is available through the GitHub Student Developer Pack [14].

A Standard Query Language (SQL) Server Common Language Runtime (CLR) extension was developed to send

TABLE III SECBIML POST-TESTING DATA

Data Point	Timing	Executions
Cache-miss sequential	14808	120
Cache-hit sequential	1452	120
Cache-miss 4 thread	18324	120
Cache-hit 4 thread	1812	120
Cache-miss 8 thread	20556	120
Cache-hit 8 thread	2016	120
Cache-hit production	139647	989
Cache-miss production	4393	289

 ISBN: 978-1-61208-778-8

TABLE V SECBIML EMAIL AND AUTOMATION TESTS

Type	Count	Errors
Visitation Email Automation	18,114	13
Email Merge Errors	756,123	1,243

the emails with proper tagging and retrieve the sent email data through the APIs. Database triggers were used to send automation responses based on the visitation of patrons. For example, an email was sent before a visitation that included details on arrival, directions to the venue, and the group's itinerary. Surveys were also sent to the patrons the day after visitation. Using the APIs from Mailgun, we were able to retrieve the data about the sent emails and check the integrity of the merged fields, appropriateness of the content in the email, and problems with delivery. TABLE V shows the errors found over a month of tests. The errors fell into two categories; data errors with the automated emails and data merge errors where data was truncated or displayed improperly in the final layout. The automation errors originated from data entry errors from operators entering transaction data and poor design in the transactional systems to allow the data inconsistencies to exist. The merge errors originated from live data that did not look like the data used in the testing of the email templates. In both cases, the percentage of error is small, but if an organization works hard to acquire a customer, these types of errors can negate that hard work.

IX. CONCLUSIONS AND FUTURE WORK

Based on our research, we demonstrate that the availability and integrity of business visualizations, documents, and communications increase using the secBIML programming language. This work demonstrates the successful implementation of the tests written in secBIML for an actual organization utilizeing their production environment. Our future work will develop tooling to make it easier to create business document tests while doing layout in the document. The tooling will make it more likely that an end-user will specify the correctness of a document. We will also create trigger-based executions of our testing programs. The triggers will enable on the fly verification instead of a point in time testing.

REFERENCES

[1] B. Evelson, "Topic Overview: Business Intelligence," Forrester, 2018.

[2] I. Letca et al., "Measuring Actual End User Performance And Availability Of Web Applications". Patent US 8,938,721 B2, 2015.

[3] C. E. F., "A relational model of data for large shared data banks," *Communications of the ACM,* vol. 13, no. 6, pp. 377-387 , 1970.

[4] N. Khoury, P. Zavarsky, D. Lindskog and R. Ruhl, "An Analysis of Black-Box Web Application Security Scanners against Stored SQL Injection," in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, Boston, MA, 2011, pp 1095-1101.

[5] World Wide Web Consortium (W3C), "Selectors Level 3," 18 November 2018. [Online]. Available: https://www.w3.org/TR/selectors-3/. [Accessed 16 October 2019].

[6] The Franklin Institute, "The Franklin Institute," 2019. [Online]. Available: http://www.fi.edu. [Accessed 16 October 2019].

[7] Logi Analytics, "Business Intelligence is Dead," 2019, [Online]. Available: https://www.logianalytics.com. [Accessed 16 October 2019].

[8] Microsoft Corporation, "What is Office 365," [Online]. Available: https://www.office.com/. [Accessed 12 November 2019].

[9] Google, Inc., "About Google Docs," [Online]. Available: https://www.google.com/docs/about/. [Accessed 12 November 2019].

[10] Microsoft, ".NET Core Guide," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/core/. [Accessed 23 October 2019].

[11] D. Kondratiuk, "Puppeteer Sharp," [Online]. Available: https://www.puppeteersharp.com/. [Accessed 23 October 2019].

[12] Google, "Puppeteer," [Online]. Available: https://developers.google.com/web/tools/puppeteer. [Accessed 23 October 2019].

[13] Mailgun Technologies, Inc, "The Email Service for Developers," [Online]. Available: https://www.mailgun.com/. [Accessed 3 December 2019].

[14] GitHub, "GitHub Education," [Online]. Available: https://education.github.com/pack. [Accessed 3 December 2019].

 ISBN: 978-1-61208-778-8