Metrinome – Continuous Monitoring and Security Validation of Distributed Systems

By Michael Atighetchi, Vatche Ishakian, Joseph Loyall, Partha Pal, Asher Sinclair, Robert Grant

Distributed enterprise systems consist of a collection of interlinked services and components that exchange information to collectively implement functionality in support of (sometimes mission critical) workflows. Systematic experimental testing and continuous runtime monitoring of these large scale distributed systems, including event interpretation and aggregation, are key to ensuring that the system's implementation functions as expected and that its security is not compromised.

To illustrate the need, consider an example Information Management System (IMS) that enables sharing of sensitive information between information publishing and consuming clients. Problems associated with configuration management can easily lead to situations in which the IMS allows unauthenticated clients to participate in information exchanges or allows unauthorized information to be disseminated to consumers. Furthermore, the loose coupling between subscribers and the IMS can lead to situations in which the IMS is unavailable and consumers believe that no new information is being published, causing significant misunderstandings across information sharing relationships. Finally, remnant vulnerabilities in the IMS can cause failures to happen at any time and cause significant damage to mission execution if not dealt with in a real-time manner. Unavailability of information sharing directly reduces situational awareness, loss of integrity can give adversaries control over mission execution, and loss of confidentiality can be detrimental to the reputation of actors and/or mission goals in general.

Monitoring and validation of IMS and client operations can aid in detection, diagnosis, and correction of situations like this. This is particularly important since 92% of reported vulnerabilities are located at the applications layer [1]. Despite the importance of experimental validation and continuous monitoring, and the increased support to adopt security assessment as part of the software development life cycle, current approaches suffer from a number of shortcomings that limit their application in continuous monitoring situations and their use in the validation of assurance claims.

First, current test practices favor unit tests over integrated tests for establishing correct functionality. Unit testing, e.g., performed via Junit [2], checks program functionality piece-by-piece but provides little to assess the overall information assurance claims of a system under test. Various tools exist for actively assessing the security of distributed systems, e.g., Nessus [3] and HP Fortify [4] to name a few, but their functionality is achieved by running specialized unit tests for security properties against either the code or the running system. In contrast, integrated end-to-end testing tools, such as YourKit [5] or Grinder [6], focus on performance and scalability. These tools enable operators to find bottlenecks or provision computing resources, but lack metrics associated with assessing security and correct functionality.

Second, integrated and end-to-end testing and experimentation is often postponed until software artifacts have matured significantly. This is because integrated testing and experimentation can be time consuming and effort intensive and the perception is that the cost of manually performing experiments early on frequently outweighs the benefits.

Finally, common testing and metrics frameworks add additional dependencies to existing systems, in the form of additional libraries that need to be loaded into the system under test and lines of code being added in support of instrumentation. This not only increases software complexity but more importantly can cause version dependency issues. It can also have unintended side effects on certification and accreditation as the software now has additional code that must be certified but that is not part of the core functionality, i.e., it is part of the continuous monitoring.

Distribution A. Approved for public release; distribution unlimited (Case Number 88ABW-2013-4215). This work was sponsored by the Air Force Research LaboratoryAFRL.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAR 2014		**00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Metrinome - Continuous Monitoring and Security Validation of Distributed Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **BBN Technologies,10 Moulton Street,Cambridge,MA,02138**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**7**	

This article describes Metrinome, a metrics framework written in Java that is specifically designed to provide a platform for structured continuous security assessments throughout the software lifecycle. The novelty of Metrinome lies in its loose coupling with the system under test and its integration of end-to-end testing with continuous application-level remote monitoring. Specifically, Metrinome provides (1) runtime computation of a wide range of metrics from log messages generated by distributed components during system execution, (2) execution of assertions over the metrics to determine correct functionality while the system is operating, and (3) improved situational awareness via dashboard views and generation of experimentation reports. The outputs of Metrinome-based assessments can be used as input to Certification and Assessment (C&A) processes to precisely document the assertions that were previously checked to hold true in the system. Metrinome is available free of charge to government entities through AFRL.

II. Related Work

A. SNMP Dashboards

A number of management platforms exist that use the Simple Network Management Protocol (SNMP) for monitoring devices and nodes. Network Management Information System (NMIS) [7] operates at the networking level and enables monitoring, fault detection, and configuration management of large complex networks. Its main metrics deal with device reachability, availability, and performance. HP OpenView, IBM Tivoli, and Nagios provide similar functionality. Unlike these platforms, Metrinome specializes on monitoring at the application level and execution of fine-grained assertions.

B. Distributed Testing

Software Testing Automation Framework (STAF) [8] is an open source multi-platform, multi-language framework that enables a set of functionalities including logging, monitoring and process invocation for the main purpose of testing. STAF operates in a peer environment; a network of STAF-enabled machines is built by running STAF agents across a set of networked hosts. In contrast to STAF, the goal of Metrinome is more focused and hence no agents are required to be installed. Avoiding agents not only leads to reduced maintenance costs but also significantly reduces the attack surface across networked systems under test. Due to their complimentary nature, we have used Metrinome in conjunction with STAF for continuous testing and integration.

C. Application-level Metrics Frameworks

Several application-level metrics frameworks exist to monitor and measure the performance of applications. For example, Javasimon [9] exposes an API which can be placed into the code and allows inline computation of count metrics and measurement of durations. Metrics [10] is similar to Javasimon but allows data to be streamed to other reporting systems, e.g., Ganglia [11] and Graphite [12].

An important distinction between Metrinome and the above mentioned frameworks is Metrinome's use of log messages to provide the same monitoring functionality. This makes Metrinome loosely coupled with the system being monitored and makes it applicable to any application that generates log messages, e.g., using Log4j or Logback.

D. Reporting/Graphing Backends

Ganglia, Graphite, and Splunk [13] are examples of highly popular platforms that offer the ability to search, analyze, and visualize data in real-time. Typically these frameworks consist of a processing backend that collects and stores the data. They also use statistical methods that provide new insight and intelligence about the data. Metrinome provides functionalities that intersect with the above mentioned applications, such as dashboard views and experimentation reports. One difference is that Metrinome focuses less on scalability but rather on ensuring correct execution of a system under test through the validation of assertions.

E. SIEM Platforms

Security Information and Event Management platforms (SIEMs), e.g. ArcSight [14], adopt many of the technologies described above, such as SNMP dashboards and reporting backends, to provide users with the ability to query, and analyze security threats generated by both hardware and software applications. Unlike Metrinome, these platforms require the deployment of agents on networked hosts to collect and report events.

III. Design and Architecture

Metrinome is designed to achieve specific objectives in portability and ease of use.

- **Portability** – Metrinome can monitor a system independent of the implementation of the system.
- **Minimal coding overhead** – Rather than adding new instrumentation libraries to monitored processes (causing versioning conflicts and Java classpath pollution), Metrinome interfaces with existing logging and auditing frameworks, e.g., Logback [15].
- **Ease of use** – To be of immediate use to experimenters and administrators, it should be easy to specify metrics and assertions that must hold over the metrics in a systematic way. In addition, results of metric computation need to be readily accessible by humans or other programs through a well-defined Application Programming Interface (API) and Graphical User Interface (GUI).

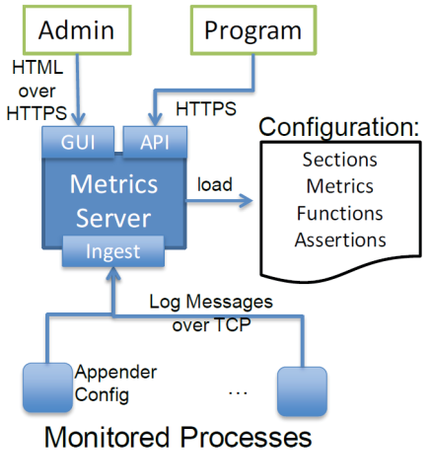


Figure 1: Metrinome High-Level Architecture

Figure 1 provides an overview of Metrinome high level architecture. Metrinome works with a set of monitored processes that have the ability to send log messages over TCP connections to the ingest API provided by the Metrics Server. Ingestion is performed via simple logging configuration changes on the monitored processes, e.g., by specifying the use of a SocketAppender in Logback to send certain log messages remotely to the Metrics Server over TCP connections in addition to or instead of sending those messages to the console or a local file.

Due to the fact that log messages issued by different processes may be similar, particularly if the processes are executing the same code base on different physical machines, the Metrics Server requires a descriptive unique process name associated with a specific logging instance as part of the log message. This requirement has already been built into most of the logging and auditing framework, enabling filtering of messages based on process names within Metrinome. The processing performed by Metrinome on received messages is defined using a XML-based Domain Specific Language (DSL), describing concepts such as sections, metrics, functions, and assertions. The Metrinome DSL allows administrators to specify processing logic in one file that can be dynamically loaded into the Metrics Server.

Finally, to ease access to information, Metrinome offers two interfaces: (1) a GUI, implemented in HTML and accessible through common web Browsers using HTTP(s), and (2) a RESTful [16] secure Web Services API for use by external programs.

IV. Modes of Use

The Metrinome framework supports a number of operational use cases and scenarios, including use during demonstrations, experiments, and continuous monitoring.

A. Runtime Visualization During Demonstrations

A major hurdle facing users during a demonstration is the ability to showcase a holistic view of the system operation while highlighting specific aspects that are being demonstrated, such as performance, load balancing, resistance to security attacks, etc. Metrinome's GUI equips the demonstrator not only with the ability to pinpoint the changes in the system as these events occur, but also to visualize these changes to the measurements graphically during runtime.

B. Experimentation

Metrinome seamlessly integrates with off-the-shelf continuous integration frameworks, such as Jenkins [17]. Users can easily specify assertions showcasing desired system behavior. Metrinome evaluates assertions at specific control points within an experiment or at the end of an experiment. Metrinome's HTTP interface also allows user controlled and on-demand evaluation of assertions at runtime. An HTTP response will indicate whether the assertion evaluation passed successfully or failed. In the case of failure, the HTTP response also includes information about the particular assertions that failed.

When an experiment is complete, Metrinome stores the state of all assertions along with metrics values, historical statistics, and definition of metrics. This process supports offline analysis and reproducibility of experiments, and can also generate inputs to C&A processes. Finally, Metrinome has the ability to export the metrics data into other programs using the Comma Separated Value (CSV) format which allows administrators to perform customized analysis over the data, using spreadsheet and visualization software of their choice.

C. Continuous Monitoring

Continuous monitoring is a desirable feature in enterprise environments because it decreases the time to react to occasional hardware and software failures and minimizes the time to mitigate security attacks such as Denial of Service attacks. While guidance for continuous monitoring is maturing [18], agencies have already started to struggle with compliance mainly due to implementation costs [19]. Metrinome reduces costs by virtue of integrating with existing logging and auditing frameworks. It also provides ready dashboard functionality that increases situational awareness at no additional implementation cost.

V. Interfaces

A. Metrinome Language

Metrinome processes receive messages based on user-specified processing logic, which is dynamically loaded into the Metrics Server. This processing logic echoes a user's perception of the desired system behavior and is declared in terms of metrics and assertions. Users are able to express such terms using a XML-based representation.

Figure 2 shows the XML schema for specifying the processing logic. The metrics element serves as an enclosing element to the entire document, while the section element serves not only to organize the metrics and assertions into different clusters, but also to limit the scope of assertions.

Figure 2: Metrinome's DSL Schema

Thus an assertion specified for a particular section will not be triggered against metrics in another section

The core of the language consists of two major elements: Metric and assert. Metric is used to specify a measurement evaluation while assert – associated with a metric or a set of metrics – is used to specify the expected system behavior.

A metric element has a unique name and a description to provide information about the Metric. An assert element has a unique name and a metricRef element which uses regular expressions to allow the referencing of a metric or a set of metrics that the assertion will be evaluated on. Both elements encompass a function which expresses a statistical calculation to perform. Functions allow the user to configure the actual operation to be performed, which in the case of metrics occurs over the incoming message (e.g., counting the number of exceptions occurred) or in the case of assertions, through the specification of a logic expression (e.g., zero number of errors).

A function specified as part of an assertion is triggered when an experiment is complete or by an external entity request. The main purpose of assertion functions is to validate metric values, thus they tend to be logical in nature.

A function specified in a metric can be triggered by a single event which is equivalent to specifying unary functions, such as count, or two separate events (denoted start_event and end_event) which is equivalent to specifying binary functions, such as time difference.

An event consists of two parts: component and regex. A component outlines the actual set of processes whose log messages can trigger such an event. All processes not specified via the component element will not trigger the specific event. The regex specifies the message string to be processed. The processing engine allows the use of regular expressions in both component and regex, thus enabling easy specification of processes and messages.

Finally, a function element can have several attributes:

- **round:** rounds a numeric value of the measurement to the nearest specified number beyond the decimal point.
- **roundhistory:** similar to round but applies over the statistical calculations rather than individual values.

Two special attributes epochs and colors are used to indicate the staleness of a measurement as observed by the Metrics Server and can be customized per metric. A user can specify a staleness threshold and an associated severity color which will be highlighted on the HTML GUI Interface.

Metrinome provides a set of predefined functions for computing metrics, including the following:

- **count:** counts the number of occurrences of an expression,
- **ratio:** provides the ratio of two expressions,
- **diff:** calculates the time difference between two events,
- **absdiff:** calculates absolute time difference between two events, and
- **sum:** calculates the sum of two expressions.

Examples of assertion functions are equals, greater than, less than, and greater than or equal. The library of functions can be easily extended to support additional functions, which currently requires changes to the Metrics Server but not to monitored processes.

```
<metric>
    <name>reqAuthz_pass</name>
    <description>Number of passed client requests</description>
    <function round="0" roundHistory="1">count</function>
    <event>
    <component>CoTToPubSvc|CoTToSubSvc</component>
        <regex>.*Request failed authorization.*</regex>
    </event>
</metric>
```

Figure 3: Example Metric: Count

```
<assert>
    <name>No_OutOfMemory_Errors</name>
    <metricRef>error_outOfMemoryErrors</metricRef>
    <function>equals</function>
    <value>0</value>
</assert>
```

Figure 4: Example Assertion: No Out Of Memory Error

```
<assert>
    <name>non_error_gt_0</name>
    <metricRef>^((?!.*(error|Processing)).*)</metricRef>
    <function>greaterThan</function>
    <value>0</value>
</assert>
```

Figure 5: Example Assertion: Non processing and error metrics should be greater than zero.

Figure 3 highlights an example of a security assessment metric called 'reqAuthz_pass' which provides the number of requests that failed during authorization generated by processes containing 'CoTToPubSvc' or 'CoTToSubSvc' in their descriptive names, based on which they are sending log messages to the Metrics Server. This metric is useful especially for testing the authorization process of an application during high load or automated attacks.

Figure 4 shows a simple assertion example over a metric called 'error_outOfMemoryErrors'. As the name indicates, this is a useful assertion for testing that a system has no out-of-memory exceptions.

Another example shown in Figure 5 highlights an assertion that showcases the correct functionality of the system under evaluation. The assertion uses regular expressions to state that all metrics except the ones containing error or processing in their names should have values greater than zero values.

B. HTML User Interface

Figure 6 displays a screenshot of the GUI. The first column highlights the name of the metric as specified in the configuration file. The next column highlights the latest measurement of the metric. By default, Metrinome provides statistical information such as the average, median, and standard deviation across historical values. The last column highlights the changes in the value of the metric over time graphically. This feature is useful to quickly pinpoint measurement anomalies. Users can view metrics without the graphs by clicking on the "Metrics" link.

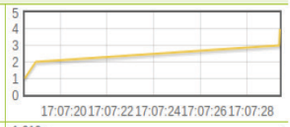
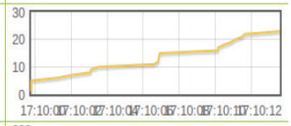
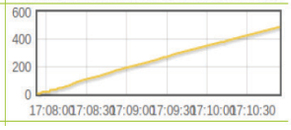
Metric	Value	mean	median	stddev	min	max	Graph
STEG Functionality:Internal Load							
subscription_count_t2e	5	2.2	2.0	1.3	1.0	4.0	
subscription_count_e2t	1	1.0	1.0	0.0	1.0	1.0	
publications_count_t2e	24	11.5	11.5	7.0	1.0	23.0	
publications_count_e2t	491	245.0	245.0	141.9	1.0	490.0	

Figure 6: Metrinome's Metrics with Graphs Interface

C. Metrinome API Interface

The service API consists of an Assertion and Metrics service accessible via HTTP.

The Assertion service offers the following functionality:

- HTTP GET http://localhost:8080/assertions
  - Triggers evaluation of assertions against the current status of the metrics, which either returns success in the form of a HTTP response code of 204, or a list of failed assertions, encoded as XML payload in the HTTP response.
- HTTP GET http://localhost:8080/assertions?SHOWDEFS
  - Displays a table of current assertion definitions.

The Metrics service offers the following API:

- HTTP GET http://localhost:8080/metrics?CSV
  - Returns the metrics values in a CSV format.
- HTTP GET http://localhost:8080/metrics?EVENTS
  - Returns the collected events that were used to generated the metrics.

VI. Use of Metrionome During Red Teaming

We have successfully used Metrinome during internal security testing of software artifacts developed under the Secure Tactical to Enterprise Gateway (STEG) [20] R&D effort. To evaluate the security benefits of STEG, we build an internal threat model

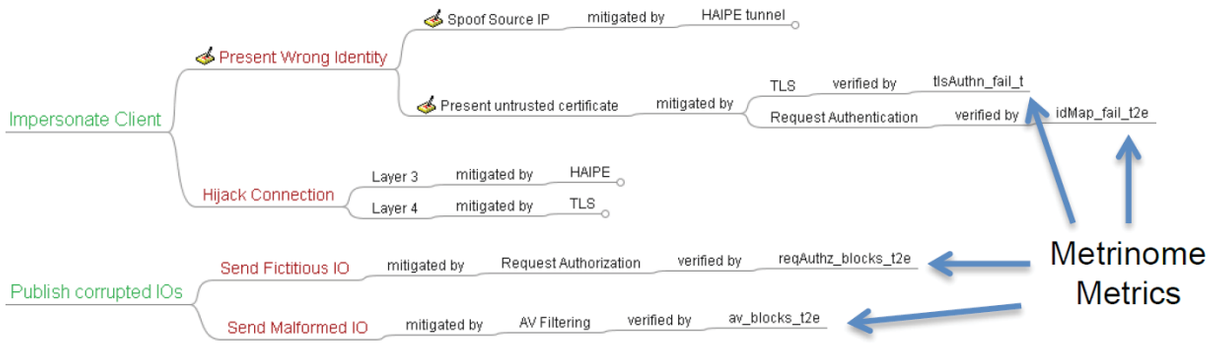


Figure 7. STEG Attack Tree for Loss of Integrity

that decomposes attacks into three main categories, namely, attacks that cause (1) loss of integrity, e.g., by corrupting service logic or changing data in transit, (2) loss of availability, e.g., by crashing critical components or exhausting shared resources, and (3) loss of confidentiality, e.g., by getting unauthorized access to sensitive information. The attacks are then further decomposed into sub-categories for each category (i.e., Integrity, Availability, and Confidentiality). The model can be visually represented as attack graphs, with annotations for defenses and logical arguments.

Figure 7 shows the resulting attack graph for integrity. The graph reads from left to right and first branches out into high-level attack strategies, e.g., Impersonate Client and Publish corrupted IOs. The next levels then provide functional refinements for the attacks. Attack refinement may lead to multiple alternatives (branches). The next level of the attack graph is annotated with mitigated by, indicating the defensive component that addresses the particular attack represented by the branch. Note that an attack strategy may have multiple mitigating defenses (indicated by the mitigated by annotation on a branch). For the cases where mitigation is verified by experimental observation or logical arguments, the attack graph is shown with an additional level, annotated with verified by describing how we determined that the STEG prototype actually addresses the threat.

We used Metrinome to establish and document correct security functionality by measuring a number of metrics listed in the attack tree, including TLS authentication failures, identity mapping failures, authorization failures, and anti-virus filtering failures.

VII. Conclusion and Future

Metrinome has proven to be an effective component in supporting runtime assessment and monitoring, demonstrations and scientific experimentation during execution of the STEG R&D effort. In particular, the integration of end-to-end testing into the continuous build cycle has helped identification and mitigation of run-time bugs.

Going forward, we expect Metrinome to grow as it is adopted by other efforts with extended requirement sets. In particular, we have plans to (1) make it easier to add custom functions without the need to recompile the Metrics Server through a plugin framework, (2) provide capabilities for more complex graph generation, e.g., by providing boxplots via integration with R [21], (3) provide the ability to define metrics over metrics and metrics capturing trends, and (4) implement an adapter layer for ingesting messages other than Logback.

About the Author(s)

Mr. Michael Atighetchi is a Senior Scientist in the distributed computing group at BBN and technical lead on several DARPA- and USAF-sponsored research projects. Mr. Atighetchi has a Master of Science degree in Computer Science from UMASS Amherst and a Master of Science in Informatics from the University of Stuttgart/Germany. Mr. Atighetchi is a Senior Member of the IEEE, member of ACM, and has authored over 60 publications in peer-reviewed conferences and journals on topics including adaptive security, Red Team assessments, identity management, and Cross Domain Solutions. Raytheon BBN Technologies, 10 Moulton St, Cambridge, MA 02138

Dr. Vatche Ishakian is a Scientist in the distributed computing group at BBN working on USAF-sponsored research projects. Dr. Ishakian's has a PhD degree in Computer Science from Boston University and is a member of the IEEE and ACM. His experience spans a broad set of disciplines across networking and distributed systems, including application-level scheduling and management, network economics, data placement, and network architecture. Dr. Ishakian has authored over 15 publications in peer-reviewed conferences and journals. Raytheon BBN Technologies, 10 Moulton St, Cambridge, MA 02138

Dr. Joseph Loyall is a principal scientist at Raytheon BBN Technologies. He has been the principal investigator for Defense Advanced Research Projects Agency and AFRL research and development projects in the areas of information

management, distributed middleware, adaptive applications, and quality of service. He is the author of over 100 published papers. He is a Distinguished Member of the ACM and a Senior Member of the IEEE and of the AIAA. Dr. Loyall has a doctorate in computer science from the University of Illinois. Raytheon BBN Technologies, 10 Moulton St, Cambridge, MA 02138

Dr. Partha Pal is a Principal Scientist at BBN Technologies. His research interest is in the areas of adaptive cyber-defense, resiliency and survivability. As the Principle Investigator in a number of past and ongoing projects sponsored by various agencies, he has been leading the development, demonstration and evaluation of innovative cyber-defense mechanisms, strategies and survivability architectures, and using them to build survivable distributed information systems. He is a senior member of the IEEE and a member of the ACM. He has over 80 publications in peer reviewed conferences and journals, and holds a PhD in Computer Science from Rutgers University. Raytheon BBN Technologies, 10 Moulton St, Cambridge, MA 02138

Mr. Asher Sinclair is a Senior Program Manager at AFRL's Information Directorate working in the Resilient Synchronized Systems Branch (RISB) at the Rome Research Site. His interests include research and development in enterprise systems management, service-oriented architectures, and Cyber security. He has contributed to more than 18 technical papers and conference proceeding publications. He holds a bachelor's degree in Computer Information Systems from the State University of New York and a master's degree in Information Management from Syracuse University. Air Force Research Laboratory, 525 Brooks Road, Rome, NY 13441, USA

Mr. Robert Grant works for the Air Force Research Laboratory Information Directorate in Rome New York. He has a B.A. in English from the University at Buffalo, a B.A. in Computer Science from Oswego State, and is currently working on his Masters in Computer Science at Syracuse University. Air Force Research Laboratory, 525 Brooks Road, Rome, NY 13441, USA

References

[1] Eoin Keary, Integration into the SDLC(Software Development Life Cycle), Retrieved Nov 06 2013, https://www.owasp.org/images/f/f6/Integration_into_the_SDLC.ppt

[2] JUnit Homepage, Retrieved Sep 06 2013, https://github.com/junit-team/junit/wiki/Getting-started

[3] Nessus Vulnerability Scanner, Retrieved Sep 06 2013, http://www.tenable.com/products/nessus

[4] HP Fortify My App, Retrieved Sep 06 2013, https://www.fortifymyapp.com/

[5] YourKit Profiler, Retrieved Sep 06 2013, http://www.yourkit.com/

[6] Grinder, Retrieved Sep 06 2013, http://grinder.sourceforge.net/

[7] Network Management Information System, Retrieved June 10 2013, http://www.sins.com.au/nmis/sample/

[8] Software Testing Automation Framework, Retrieved June 10 2013, http://staf.sourceforge.net/

[9] Java Simon - Simple Monitoring API , Retrieved June 10 2013, http://code.google.com/p/javasimon/

[10] Metrics, http://metrics.codahale.com, Retrieved June 10, 2013

[11] Ganglia Monitoring System, Retrieved June 10 2013, http://ganglia.sourceforge.net/

[12] Graphite - Scalable Realtime Graphing, Retrieved June 10 2013, http://graphite.wikidot.com/

[13] Splunk, http://www.splunk.com/ Retrieved June 10 2013.

[14] ArcSight, http://en.wikipedia.org/wiki/ArcSight

[15] Cody Burleson, "How to setup SLF4J and LOGBack in a web app – fast", Apr 10 2013, https://wiki.base22.com/display/btg/How+to+setup+SLF4J+and+LOGBack+in+a+web+app+-+fast

[16] Fielding, Roy Thomas, "Architectural styles and the design of network-based software architectures", Diss. University of California, 2000.

[17] Jenkins: An extendable open source continuous integration server, http://jenkins-ci.org/ Retrieved July 1 2013.

[18] Kelley Dempsey, Nirali hawla, Arnold Johnson, Ronald John-ston, Alicia Clay Jones, Angela Orebaugh, Matthew Scholl, and Kevin Stine, "Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations", Retrieved June 25 2013 http://csrc.nist.gov/publications/nistpubs/800-137/SP800-137-Final.pdf

[19] Jason Miller, "Agencies struggle with continuous monitoring mandate", Retrieved June 25 2013 http://www.federalnewsradio.com/513/2681377/Agencies-struggle-with-continuous-monitoring-mandate

[20] "R: Box Plot Statistics", R manual, Retrieved June 3 2013, http://stat.ethz.ch/R-manual/R-devel/library/grDevices/html/boxplot.stats.html

[21] "Secure and QoS-Managed Information Exchange between Enterprise and Constrained Environments", currently in submission to appear in Proceedings of ISORC 2014.