AFRL-RY-WP-TR-2020-0002

SCALABLE TRANSPARENCY ARCHITECTURE FOR RESEARCH COLLABORATION (STARC) – DARPA TRANSPARENT COMPUTING (TC) PROGRAM

John Griffith, Derrick Kong, Armando Caro, Brett Benyo, Joud Khoury, Timothy Upthegrove, Timothy Christovich, Stanislav Ponomorov, Ali Sydney, Arjun Saini, Vladimir Shurbanov, Christopher Willig, David Levin, and Jack Dietz

Raytheon BBN Technologies Corp.

MARCH 2020
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

*//Signature//

CHARLES P. SATTERTHWAITE, DR-03
Program Manager
Chief, Resilient and Agile Avionics Branch

//Signature//

DAVID G. HAGSTROM, DR-04
Chief, Resilient and Agile Avionics Branch
Spectrum Warfare Division

//Signature//

JOHN F. CARR, DR-04
Chief, Spectrum Warfare Division
Sensors Directorate

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YY)* March 2020	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* 12 June 2015 – 30 November 2019

4. TITLE AND SUBTITLE SCALABLE TRANSPARENCY ARCHITECTURE FOR RESEARCH COLLABORATION (STARC) – DARPA TRANSPARENT COMPUTING (TC) PROGRAM	**5a. CONTRACT NUMBER** FA8650-15-C-7559
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 61101E
6. AUTHOR(S) John Griffith, Derrick Kong, Armando Caro, Brett Benyo, Joud Khoury, Timothy Upthegrove, Timothy Christovich, Stanislav Ponomorov, Ali Sydney, Arjun Saini, Vladimir Shurbanov, Christopher Willig, David Levin, and Jack Dietz	**5d. PROJECT NUMBER** 1000
	5e. TASK NUMBER N/A
	5f. WORK UNIT NUMBER Y1AZ
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Raytheon BBN Technologies Corp. 10 Moulton Street Cambridge, MA 02138	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory, Sensors Directorate, Wright-Patterson Air Force Base, OH 45433-7320, Air Force Materiel Command, United States Air Force; Defense Advanced Research Projects Agency, DARPA/I2O, 675 North Randolph Street, Arlington, VA 22203	**10. SPONSORING/MONITORING AGENCY ACRONYM(S)** AFRL/RYZC
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RY-WP-TR-2020-0002

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

13. SUPPLEMENTARY NOTES
This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. © 2020 Raytheon BBN Technologies Corp. This work was funded in whole or in part by Department of the Air Force Contract FA8650-15-C-7559. The U.S. Government has for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license to use, modify, reproduce, release, perform, display, or disclose the work by or on behalf of the U. S. Government. Report contains color.

14. ABSTRACT
The STARC project designed, built, and operated a test range and infrastructure to support the development and operation of systems designed to detect and identify malware, most importantly Advanced Persistent Threats operating in a heterogeneous enterprise environment. This effort also involved developing a cross-system event logging system and a policy enforcement module that could read these events and a policy statement, make decisions as to whether or not a proposed action should be allowed or blocked, and return this decision to an endpoint to be acted upon. This program demonstrated that such a system is feasible and deserves further investigation and development.

15. SUBJECT TERMS
cyber range, test range, cyber events, cyber monitoring, cyber defense, event monitoring, large scale data collection and analysis

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT:**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** (Monitor)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	SAR	148	Charles P. Satterthwaite
Unclassified	Unclassified	Unclassified			**19b. TELEPHONE NUMBER** *(Include Area Code)* N/A

TABLE OF CONTENTS

Section Page

LIST OF FIGURES ..... **VI**
LIST OF TABLES ..... **VIII**
1.0 SUMMARY **1**
2.0 INTRODUCTION **2**
3.0 METHODS, ASSUMPTIONS AND PROCEDURES **6**
3.1 Introduction 6
3.2 Architecture 6
3.2.1. Proposed 6
3.2.1.1. Smart Collectors 6
3.2.1.2. Forensic Layer 6
3.2.1.2.1. Master Data Store 6
3.2.1.2.2. Batch Processing Framework 7
3.2.1.2.3. Indexed Forensic View 7
3.2.1.3. Real Time Layer 7
3.2.1.3.1. Real Time Substrate 7
3.2.1.3.2. Stream Processing Framework 8
3.2.1.3.3. Indexed Real Time View 8
3.2.1.4. Query Layer 8
3.2.1.5. Enforcement 8
3.2.1.6. Feedback 8
3.2.2. Lambda Architecture vs. Kappa Architecture 9
3.2.3. What Was Implemented 10
3.2.3.1. Smart Collectors 10
3.2.3.2. TA1 Translators 11
3.2.3.3. Forensic Layer 11
3.2.3.3.1. Master Data Store 11
3.2.3.4. Batch Processing Framework 11
3.2.3.5. Indexed Forensic View 12
3.2.4. Real Time Layer 12
3.2.4.1. Real Time Substrate 12
3.2.4.2. Stream Processing Framework 12
3.2.4.3. Indexed Real Time View 12
3.2.5. Query Layer 12
3.2.6. Enforcement 13
3.2.7. Feedback 13
3.3 Common Data Model (CDM) 13
3.3.1. CDM Goals 13
3.3.2. CDM Conceptual View 14
3.3.2.1. Events and Subjects 14
3.3.3. Objects 16

3.3.4. Tags 16
3.3.5. Extensibility Points 17
3.3.6. Typed Schema Definition 18
3.3.7. CDM Evolution 18
3.4 Engagements 18
3.4.1. Engagement #1 19
3.4.1.1. Goals 19
3.4.1.2. Network 19
3.4.1.3. Hosts 20
3.4.2. Engagement #2 21
3.4.2.1. Goals 21
3.4.2.2. Network 21
3.4.2.3. Hosts 21
3.4.3. Engagement #3 22
3.4.3.1. Goals 22
3.4.3.2. Network 22
3.4.3.3. Hosts 23
3.4.4. Engagement #4 23
3.4.4.1. Goals 23
3.4.4.2. Network 23
3.4.4.3. Hosts 23
3.4.5. Engagement #5 24
3.4.5.1. Goals 24
3.4.5.2. Network 24
3.4.5.3. Hosts 24
3.4.6. Challenges with Reproducibility 24
3.5 Policy Enforcement Capability 25
3.5.1. Overview 25
3.5.2. Motivating Example 25
3.5.3. Policy Enforcement Module Design 26
3.5.4. Evolution of the Policy Enforcement Demonstration 27
3.5.4.1. Engagement #3 27
3.5.4.1.1. The TA1 Delay Issue 28
3.5.4.2. Engagement #4 29
3.5.4.2.1. Engagement #4 Policy Enforcement Demonstration Scenarios 31
3.5.4.2.2. Policy Enforcement Module Update (Engagement #4) 31
3.5.4.3. Engagement #5 32
3.6 Infrastructure 32
3.6.1. Goals 32
3.6.2. Project Evolution 33
3.6.2.1. Base Infrastructure 33
3.6.2.1.1. Description 33
3.6.2.1.2. Separation of Privileges 33

3.6.2.1.3. Separation of Base and Experimental Infrastructure 34
3.6.2.1.4. Management of Experimental Infrastructure 34
3.6.2.1.5. Managing Standard Base Systems 35
3.6.2.1.6. Performance Tuning 35
3.6.2.1.7. Services 36
3.6.2.1.7.1. Monitoring Infrastructure 36
3.6.2.2. Experimental Infrastructure 39
3.6.2.2.1. Network 39
3.6.2.2.2. Hosts 41
3.6.2.2.2.1. Deployment 41
3.6.2.2.2.2. TA3 Experimental Infrastructure Services 42
3.7 Critical Factors Pyramid 44
3.8 Continuous Engagement Process 46
3.9 TC Range Security 46
4.0 RESULTS AND DISCUSSION 48
4.1 Critical Factors Pyramid and Architecture 48
4.2 Infrastructure 49
4.2.1. Network Performance 49
4.2.1.1. Engagement #1 49
4.2.1.1.1. Pre-Engagement Experiments 50
4.2.1.1.1.1. Producer Performance Experiments 50
4.2.1.1.1.2. Kafka Latency Experiment 52
4.2.1.1.1.3. Consumer Performance Experiments 53
4.2.1.1.1.4. Results for Other Language Bindings 54
4.2.1.1.1.5. Stability Testing 55
4.2.1.1.2. Engagement #1 Results 56
4.2.1.2. Engagement #2 58
4.2.1.2.1. Pre-Engagement Performance Testing and Improvements 58
4.2.1.2.2. Results 59
4.2.1.2.3. Data Volume Analysis 60
4.2.1.2.3.1. Data Description 61
4.2.1.2.3.2. Modeling Methodology 61
4.2.1.2.3.3. Known Issues with this Approach 61
4.2.1.2.3.4. Result – Enterprise Model A 62
4.2.1.2.3.5. Result – Enterprise Model B 62
4.2.1.3. Engagement #3 63
4.2.1.3.1. Pre-Engagement Performance Testing and Improvements 63
4.2.1.3.2. TA2 Disk Performance Issues 64
4.2.1.3.3. Results 64
4.2.1.4. Engagement #4 65
4.2.1.4.1. Kafka on CADETS 66
4.2.1.4.2. Publishing Delays 68
4.2.1.4.3. Results 68

4.2.1.5. Engagement #5 69
4.2.1.5.1. Publishing Delay Monitoring 70
4.2.1.5.2. NTP Time Synchronization Monitoring 71
4.2.1.5.3. Network Overhead Metrics 71
4.2.1.5.3.1. TA1 to TA3 Traffic 72
4.2.1.5.3.2. TA3 to TA2 Traffic 73
4.2.1.5.3.3. TA3 to TA3 Traffic 74
4.2.1.5.4. TA1 Performance and Overhead 74
4.2.2. Issues with Mobile Handsets 79
4.3 Policy Enforcement Demonstrations 79
4.3.1. Engagement #3 79
4.3.1.1. MARPLE 80
4.3.1.2. ADAPT 82
4.3.1.3. RIPE 84
4.3.1.4. Engagement #3 Results 84
4.3.2. Engagement #4 84
4.3.2.1. MARPLE 85
4.3.2.2. RIPE 86
4.3.2.3. ADAPT 86
4.3.2.4. Engagement #4 Results 86
4.3.3. Engagement #5 87
5.0 CONCLUSIONS 90
5.1 Architecture 90
5.2 Common Data Model 90
5.3 Infrastructure 91
5.4 Engagements 91
5.5 Policy Enforcement Demonstration 91
6.0 RECOMMENDATIONS 92
7.0 REFERENCES 94
APPENDIX A COMMON DATA MODEL VERSION 0.5 SCHEMA HIGHLIGHTS 96
APPENDIX B ORANGE – THE INFRASTRUCTURE FRAMEWORK DECISION REVISITED 99
B.1 Preamble 99
B.2 Philosophy 99
B.3 Requirements 100
B.3.1. Range Management Software 100
B.3.2. Range Hardware 101
B.3.3. Network 102
B.3.4. Ease of Use 103
B.3.5. Experiment Isolation 103
B.3.6. Experiment Logistics 103
B.4 Research and Experimentation Results 104
B.4.1. Preparations and Toolset Selection 104
B.4.2. OpenStack 105

B.4.3. CloudStack 111
B.5 Conclusion and Recommendations 113
APPENDIX C RE-IMPLEMENTING THE TC RANGE, GUIDE AND INSTRUCTIONS 115
C.1 Open Cloud Services and TC requirements 115
C.2 OpenStack Services for functions not provided by TC: 122
APPENDIX D GETTING STARTED – A GENERAL GUIDE FOR A TEAM PLANNING TO SET UP A TEST RANGE 124
APPENDIX E ENGAGEMENT #1 INFRASTRUCTURE 126
APPENDIX F ENGAGEMENT #5 INFRASTRUCTURE 127
APPENDIX G ENGAGEMENT #3 POLICY DEMONSTRATION GROUND TRUTH INFORMATION 131
G.1 Policy One Ground Truth 131
G.2 Policy Two Ground Truth 131
G.3 Policy Two Ground Truth for FAROS 132
G.4 Policy Three Ground Truth 133
G.5 Policy Four Ground Truth 133

LIST OF FIGURES

Page

Figure 1. The STARC Architecture Was Intended to be Driven by Experience 3
Figure 2. Original STARC Architecture 6
Figure 3. Example Implementation of the Lambda Architecture 9
Figure 4. Example Implementation of the Kappa Architecture 9
Figure 5. Proposed Architecture Supporting Both Real-Time and Forensic Processing 10
Figure 6. CDM Common Graph Data Model 14
Figure 7. Network Diagram for Engagement #1 20
Figure 8. Network Layout for Engagement #2 21
Figure 9. Network Layout for Engagement #3 22
Figure 10. Engagement #3 Network 23
Figure 11. Policy Enforcement System Architecture 25
Figure 12. TA2 Policy Check Transaction 26
Figure 13. App Store Processing Flow 30
Figure 14. STARC Basic Topology 33
Figure 15. TC Range Monitoring Infrastructure 37
Figure 16. Engagement #2 Dashboard 38
Figure 17. Physical Network Layout 39
Figure 18. VLANS Configured for Engagement #1 40
Figure 19. Data Store Architecture for Engagement #3 43
Figure 20. Basic Hadoop Design for Engagement #3 44
Figure 21. The Critical Factors Pyramid 45
Figure 22. CDM Data Accumulation Over Engagement #5 48
Figure 23. Maximum Producer Data Rates 51
Figure 24. Producer Data Rate vs Record Size, Fixed Flushing 51
Figure 25. Producer Data Rate vs Record Size, OS-Managed Buffering 52
Figure 26. Kafka Latency Experiment Setup 52
Figure 27. Java Consumer Throughput 54
Figure 28. Producer Performance for Engagement #1 C++ and Python Clients 54
Figure 29. Engagement #1 Stability Test - Producer Side 55
Figure 30. Engagement #1 Stability Test - Consumer Side 56
Figure 31. Enterprise Model A Storage Requirements Over Time 62
Figure 32. Enterprise Model B Storage Requirements Over Time (250 Servers) 63

Figure 33. Enterprise Model B Storage Requirements Over Five Years for Varying Numbers of Servers..............63

Figure 34. Proposed Kafka Setup for Engagement #4 67

Figure 35. Maximum Data Rates, CADETS vs Normal Kafka Brokers 68

Figure 36. Publishing Delay Example Detail...70

Figure 37. Publishing Delay Over Time, Engagement #5 71

Figure 38. CADETS Data Flow Showing Internal Use of Kafka...73

Figure 39. Mobile Phone Harness with Handset and WebCam 79

Figure 40. MARPLE Event Flow Graph..81

Figure 41. ADEPT Event Flow Graph...83

Figure 42. Event/Flows for Policy Demo Showing Missing IPC Provenance Data for AIA and TRACE 88

Figure 43. Event/Flows for Policy Demo Showing Missing IPC Provenance Data for CADETS and THEIA 88

Figure 44. Event/Flows for Policy Demo Showing Missing IPC Provenance Data for TRACE and MARPLE.............89

Figure B-1. OpenStack Main Components..106

Figure B-2. A Feature-full OpenStack Instance 106

Figure B-3. A Basic CloudStack Deployment...112

LIST OF TABLES

Page

Table 1. General Descriptions of TC Program Technical Areas 2

Table 2. Transparent Computing Performers 5

Table 3. TC Engagement Dates 18

Table 4. Engagement #3 TC Policy Definitions 27

Table 5. CDM NetFlow Delays for TA1 29

Table 6. Engagement #4 TC Policy Definitions 31

Table 7. Policy Enforcement Demos for Engagement #4 31

Table 8. Policy Enforcement Scenarios for Engagement #5 32

Table 9. Engagement #1 Bovia Record Statistics 57

Table 10. Engagement #1 Pandex Record Statistics 57

Table 11. Engagement #1 Stretch Period Record Statistics 57

Table 12. Engagement #1 Total Record Statistics 58

Table 13. Java vs Python 3 Producer Performance, Engagement #2 59

Table 14. Engagement #2 Bovia Scenario Data Counts and Record Sizes 59

Table 15. Engagement #2 Pandex Scenario Data Counts and Record Sizes 60

Table 16. Data Rate Statistics for Engagement #2 60

Table 17. Producer Throughput Experiment, SSL vs Plaintext and Kafka 0.11 vs 1.0 64

Table 18. Consumer Throughput Experiments, SSL vs Plaintext and Kafka 0.11 vs 1.0 64

Table 19. Producer Data Metrics for Engagement #3 65

Table 20. Engagement #4 TA1 Publishing Volumes (One Day) 69

Table 21. Engagement #4 Cross-Host Publishing Volumes (Three Day) 69

Table 22. Representative TA1 → TA3 Traffic 72

Table 23. CADETS-Kafka Internal Traffic Volumes and Rates 73

Table 24. TA3 → TA2 Traffic Rates at Three Points in Time 74

Table 25. TA3 Internal Traffic over a Two Hour Period 74

Table 26. TA1 Metrics Collected Over One Hour - TRACE, THEIA, and CADETS 76

Table 27. TA1 Metrics Collected Over One Hour – MARPLE, AIA and Clearscope 77

Table 28. Engagement #5 TA1 Data Production Results 78

Table 29. MARPLE Engagement #3 Policy Demonstration Results 80

Table 30. ADAPT Engagement #3 Policy Demonstration Results 82

Table 31. MARPLE Engagement #4 Policy Enforcement Results 85

Table 32. RIPE Engagement #4 Policy Enforcement Results 86

Table 33. ADAPT Engagement #4 Policy Enforcement Results 86

1.0 SUMMARY

BBNs *Scalable Transparency Architecture for Research Collaboration (STARC)* is an expandable, flexible reference Transparent Computing architecture instantiated as a system prototype comprised of a multi-layer data collection architecture, a shared data model, and an analysis and enforcement engine. The prototype allowed integration of combinations of the full breadth of event sources and policy enforcement points on heterogeneous endpoint systems and analysis engines from multiple providers, enabling proactive enforcement of security policies, near-real-time intrusion detection, and post-attack forensic analysis.

We developed a data collection system based on the Lambda architecture to support both real-time analysis performed for alerting and policy evaluation, and batch processing used for forensic investigation. The implementation is based on Apache Kafka using multiple Kafka brokers. As the system integrator, BBN developed clients for Java, Python and C allowing endpoints to send data into Kafka Topics and for analysis engines to receive real-time feeds and execute queries from those Topics.

The data itself consisted of records specified by a Common Data Model (CDM) describing security-relevant events occurring on each protected endpoint. The goal of CDM is to provide sufficient detail on the flow of information within each endpoint to allow an analysis to trace information and affects as they move through a system and to allow questions such as:

What files downloaded from the network has Process X read or accessed before requesting privileged access to a system?

Was the file Node A is attempting to load for execution created by an untrusted user?

Was this data transfer out of the network initiated by a user, and not a script?

A primary advantage of a common event definition and record format is that it supports cross-system (Windows, Linux, VxWorks, iOS, Android, etc.) data collection and analysis. While each system may support concepts that are unique, most operations such as file reads and writes, interprocess communications, network reads and writes, shared memory operations, and other such information flows are common across all operating systems.

We developed a near real-time application for the TC infrastructure allowing the collected data – in combination with queries supported by the analysis engines – to enforce provenance-based policies over a single host such as those listed above. Extensions to this system could allow the enforcement of complex, multi-host policies such as 'never allow the transfer any file downloaded from the network from a DMZ host to any host except those specifically authorized'.

Finally, we developed and deployed a small range to support and exercise TC components for five vendors – allowing a penetration tester to attack protected endpoints. An isolated range was necessary as the use of malware was planned in violation of the terms of service of commercial cloud operators. We evaluated OpenStack as the basis for the operations framework at the beginning of the program and decided it was too complex and over-featured for TC needs and created our own, eventually depending on PXE, SaltStack and a number of scripts and data files. As it turns out, some of the assumptions made at the start of the program and changing conditions encountered during the period of the program caused us to re-examine that decision at the end and we determined that using an existing open source framework may have been a better choice after all.

2.0 INTRODUCTION

Modern cyber security currently faces a host of threats for which it has either no antidote or only limited methods to detect and counter. The most devastating of these are the Advanced Persistent Threats (APTs), a set of sophisticated programs that use stealthy long-term reconnaissance and subversion to achieve a set of goals including infiltration, persistence, detection avoidance, privilege escalation, lateral movement, system control and information exfiltration. Current systems typically do not detect APTs until after months or years of subversive activity, and cannot counter them without wiping and resetting a large portion of the targeted network. The Transparent Computing (TC) Program aimed at earlier detection by tagging and tracking causal relationships among activities across an enterprise. This transparency into otherwise opaque computing systems was intended to allow development of an enterprise-wide information plane to reason over these structures, and hence a means to detect APTs earlier and enact effective, granular enforcement policies.

Table 1. General Descriptions of TC Program Technical Areas

Technical Area	Description
TA1 – Tagging and Tracking	Identify and report APT related events and event metadata that can be used to effectively identify the presence of an APT earlier in its life-cycle.
TA2 – Detection and Policy Enforcement	Analyze and reason over large volumes of TA1 generated data to detect and block attacks in progress
TA3 - Architecture	Work with TA1 and TA2 performers to design and develop the experimental prototype simulating deployment in an enterprise IT environment while maintaining security and supporting rapid speed of detection, effective use of network bandwidth, and accurate situational awareness.
TA4 – Scenario Development	Develop operational scenarios involving APT infiltration and operation in order to provide an information base for TA1 and TA2 providers and to serve as the basis for establishing metrics in the latter phases of the effort.
TA5 - Evaluation	5.1 – Adversarial Challenge Team: Create new methodologies and tools to conduct new attacks to identify strengths and weaknesses of the TC solutions. 5.2 – Baseline Team: Use current commercially available SIEM tools to assume the role of a TA1/2 team attempting to detect attacks by the TA5.1 team.

The TC Program was originally envisioned with a set of five Technical Areas (TAs) as noted in Table 1.

As a solution for TA-3, Raytheon BBN Technologies (BBN) proposed the *Scalable Transparency Architecture for Research Collaboration (STARC)*, an expandable, flexible reference TC architecture, instantiated as a system prototype comprising a multi-layer data collection architecture and an analysis and enforcement engine. The prototype was intended to allow integration of combinations of the full breadth of TA1 and TA2 technology alternatives, enabling both proactive enforcement of security policies and near-real-time intrusion detection and forensic analysis. Note that in this introductory section, we outline the high-level vision of the STARC architecture as originally proposed to DARPA and in later sections expand upon this and also indicate where our eventual work diverged from this vision.

Building STARC involved two main thrusts:

- Production of a flexible, expandable reference architecture, including APIs and data formats, based on architectural considerations for coordination and composition of TC technologies, and for integration of these technologies with the enterprise;
- Implementation of the TC system infrastructure components to produce a working prototype of the reference architecture that can accommodate the candidate TA1 and TA2 technologies.

Figure 1 shows a high-level view of STARC, including the components that were in the original design, and how they were intended to interact with the other TC components, the host enterprise and our design and development methodology. The proposed STARC components included a data handling system, a common data model, an analysis and enforcement dashboard, security, and finally APIs to allow these components to interact with each other. All of these used the Critical Factors Pyramid methodology to guide their cooperative design and implementation.

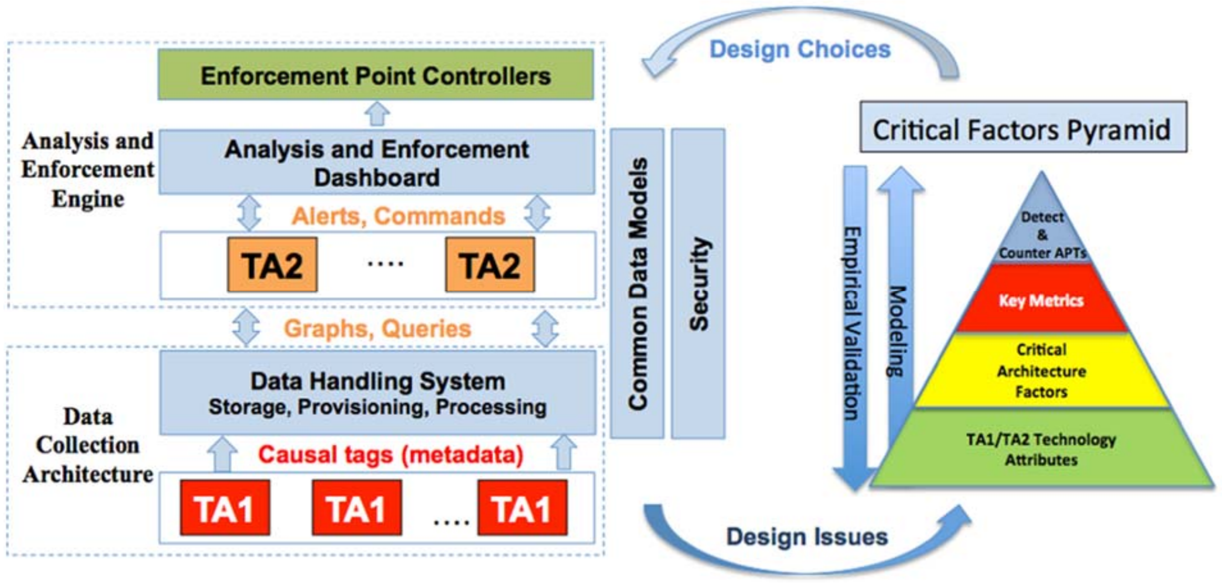


Figure 1. The STARC Architecture Was Intended to be Driven by Experience

The proposal included a number of innovative claims dealing with the initial architecture, the common data model (CDM), and the process of evolving both the architecture and the Critical Factors Pyramid:

- Application of the Lambda Architecture [1] pattern for the data handling system enables sub-second processing latencies over terabytes of data and horizontal scaling of both real-time and forensic processing.
- Use and adaptation of the W3C provenance data model [2] and domain-type enforcement fosters common conceptual data models, workflow provenance and policy representations.

- Implementation of an analysis and enforcement dashboard for fusing TA2 alerts allows enhanced situational awareness view and confidence in dispatching suggested enforcement actions.
- Feasibility assessment of using introspection and meta-policy within the TC system to prevent information leakage and provide defense against compromised devices.
- Use of the Critical Factors Pyramid facilitates thorough analysis of architectural design choices and provides a process for a design and validation cycle, and
- Continuous engagement process provides a cooperative environment for overall system development and allows rapid adoption of new developments during program execution.

The core of STARC was a multi-layer data collection architecture that integrated different TA1 technologies operating across various software layers, applications, and platforms. A central challenge for TC was to handle the voluminous granular data produced by multiple TA1 instantiations throughout the enterprise. BBN's **data handling system architecture** was based on the big-data Lambda Architecture [1] [3] (LA) pattern. The LA pattern is a state-of-the-art, industry proven best practice for performing *infinitely scalable and arbitrary computations on large volumes of data in real-time and forensically*. STARC was designed to use smart collectors and enterprise instrumentation for tag propagation across system boundaries. It was well suited for cloud deployments and grounded in standard, open-source technologies.

Another set of technical challenges involved the creation of **common conceptual data models** for workflow provenance data and policy representations, required for the creation of common data formats and API specifications. BBN's plan was to closely collaborate with all technology performers to build common shared data models and specifications using the emerging World Wide Web Consortium (W3C) Provenance Data Model [2] (PROV-DM) standard as a starting point. In addition, it was expected that complete root-cause analysis would require harmonization of the potentially orthogonal TA2 data for consistent policy enforcement and complete actionable situational awareness. Thus, STARC was designed to use a common policy representation based on a variation of the standard Domain-Type Enforcement (DTE) Model [4] as a starting point for reconciling policy alerts raised by TA2s.

A key TC system goal was to provide enhanced situational awareness, allowing prompt detection and proactive enforcement of policies against APTs. To address this challenge, STARC intended to build an **analysis and enforcement dashboard** to fuse TA2 outputs (e.g., alerts, actions) and interface with enterprise devices to trigger the enforcement mechanisms.

Securing the TC architecture itself against tampering and leakage would be a critical requirement for transitioning the technology to an actual working enterprise. STARC's approach to securing the architecture was based on a combination of applying proven security best-practices and technologies, and leveraging novel capabilities produced by the TC architecture to enable **self-protection using introspection**. BBN worked with TA1 and TA2 performers to investigate the feasibility of building an introspection capability in order to allow the TC system to monitor itself.

Assembling a large breadth of TA1 and TA2 technologies, each with potentially different design parameters, required a methodical approach for exploring the design space. Our methodology, governed by our **Critical Factors Pyramid framework**, ensured complete bottom-up coverage of the design space, including quantitative and qualitative assessment of the design choices, with a focus on meeting the overall program goals. The topics we assessed include tradeoffs relating to the integration of TC technologies with enterprise networks, potential partial deployment of TC technologies, composition of different TA1s, and the centralization of TA2 instantiations. We built and refined our Critical Factors Pyramid framework through direct collaboration with all the performers and DARPA to ensure the design effort met the expected program goals.

BBN had developed numerous architectural designs and performed as integrator for DARPA and other government customers, and had acted as the challenge team in multiple programs. In order to achieve success and provide a venue for all contributors to demonstrate their capabilities, each of these efforts required a deep understanding of all of the core technologies being developed by the program. To develop STARC, BBN used a **Continuous Engagement Process** that ensures constructive interaction among the TC performers, creating a strong collaborative design effort. Continuous interactions provided an ongoing dialogue to capture changes and new findings quickly and efficiently, and allowed each participant to elucidate and understand their own and others' contributions. Further, this process ensured that the architecture and the instantiated prototype would evolve and adapt to new findings and results as they emerged.

References to the performers of TC will be made throughout this document. For clarity, we provide a list of their names and the TAs they participated in here for reference.

Table 2. Transparent Computing Performers

Short Name	Full Name	Team Members	TA Participation
ADAPT	A Diagnostic Approach for Persistent Threat Detection	Galois, University of Edinburgh, Oregon State University	2
AIA	Acuity Intelligence Agent	Five Directions	1
CLEARSCOPE	Clearscope	Massachusetts Institute of Technology, Aarno Labs	1
FAROS	FAROS	University of Florida	1
MARPLE	Mitigating APT Damage by Reasoning with Provenance in Large Enterprise Networks	IBM, Stony Brook University, Northwestern University, University of Illinois at Chicago	1, 2
RIPE	Rapid Identification and Prevention of Exfiltration	BAE Systems, University of Texas at Austin	2
STARC	Scalable Transparency Architecture for Research Collaboration	Raytheon BBN Technologies	3
THEIA	Tagging and Tracking of Multi-Level Host Events for Transparent Computing and Information Assurance	Georgia Institute of Technology	1
TRACE	Tracking and Analysis of Causality at Enterprise Level	SRI International, Purdue University, University of Wisconsin, University of Georgia	1
5.1		Kudu Dynamics	5.1
5.2	CPT	2nd Cyber Protection Battalion, US Army	5.2

3.0 METHODS, ASSUMPTIONS AND PROCEDURES

3.1 Introduction

In this section we will describe our original proposed approach for the architecture, Common Data Model (CDM), conduct of engagements, the policy enforcement model, and the underlying infrastructure supporting the operation and monitoring of the range. We describe the metrics used to evaluate system performance and throughput and the steps we took to alleviate any issues discovered and the rationale behind them.

3.2 Architecture

3.2.1. Proposed

The original architecture proposed for STARC is shown in Figure 2. The numbers show the sequential data flow, with the two subsets - 3, 4 and 5 and 6, 7, and 8, each occurring in parallel. Each of the major components is discussed below.

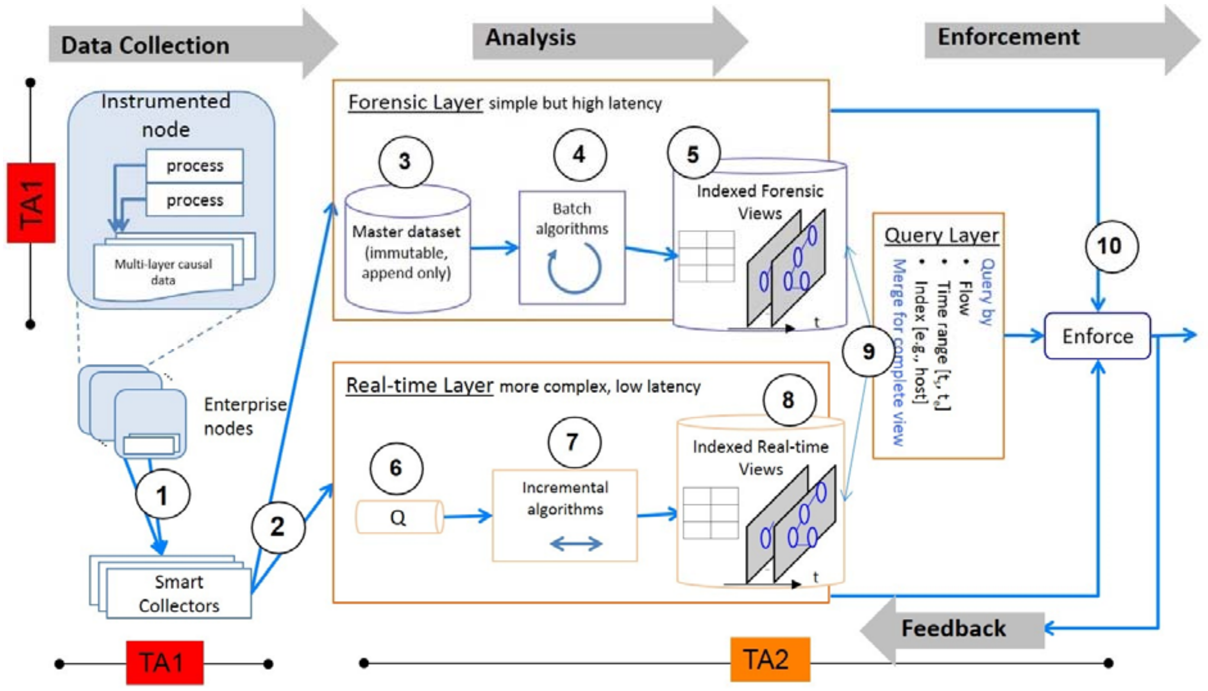


Figure 2. Original STARC Architecture

3.2.1.1. Smart Collectors

The idea behind the smart collectors was that granular causal tags and metadata produced by TA1 performers could be collected there and delivered to the TA3 central architecture in a consistent manner using proven secure protocols. The smart collectors could also augment the data stream with additional data useful to analysis.

3.2.1.2. Forensic Layer

3.2.1.2.1. Master Data Store

The master data store is intended to be the source of ground truth in the system. A good master data store possesses the following qualities:

- Fast sequential write capability to keep up with incoming records
- Preserved order of incoming records
- Almost never delete data
- Scale out to retain data for as long as necessary

The data store should not attempt to structure or index the data, as that could affect its ability to keep up with the incoming write speeds. Furthermore, because it is the source of ground truth, data is almost never deleted from the store. The only times data should be deleted from the master data store is in rare cases where the data itself is entirely corrupted or incorrect. Analysis and conclusions based on the data should happen at later stages of the processing pipeline by combining, aggregating, and augmenting the ground truth.

3.2.1.2.2. Batch Processing Framework

The batch processing framework is the mechanism for processing all historical data into smaller views of data. It may calculate summarizing statistics or it may use algorithms to look for anomalies. A batch processing framework has the following characteristics:

- Performs high-latency-high-accuracy calculations
- Duration of algorithm depends highly on data set size

The batch processing framework can make calculations with high accuracy because of its high latency. It is able to go back and query the data store as many times as necessary in order to get the correct answer. It is not subject to any real time constraints, and in particular it is not influenced by the rate at which new data comes in.

3.2.1.2.3. Indexed Forensic View

An indexed forensic view is an output of the batch processing. There may be multiple views calculated over a single run of algorithms through the batch processing framework. The indexed forensic view provides the following:

- Stores the output of batch processing framework for analysis
- Provides low latency access to last completed batch output, where last batch may be stale
- Accessible through a common query layer

The output provided by the indexed forensic view is in some sense always stale. Because the batch processing framework takes a long time to run a job, and because new data is always coming in, there will essentially always be data missing from the indexed forensic view. In this sense, while the views should be highly accurate given the data they were computed from, they are not highly accurate in terms of the data available to the entire system. Also, it is worth pointing out that unlike the master data store, the indexed forensic views may be written into a data base that requires random writes for the views being computed.

3.2.1.3. Real Time Layer

3.2.1.3.1. Real Time Substrate

The real time substrate is the input point for data coming into the system. It requires the following characteristics:

- Fast sequential write
- Introduce as little latency as possible

- Provide fault tolerance

The fast sequential write requirement is similar to the master data store, but only the most recent data needs to be present. The real time substrate should provide access to data almost as soon as it comes in, but since it is the entry point for all data (even for data destined for the master data store), it needs to implement some amount of fault tolerance in the event that a machine has a fault.

3.2.1.3.2. Stream Processing Framework

The stream processing framework is similar to the batch processing framework, except that it needs to implement similar features with far lower latency. The stream processing framework needs the following qualities:

- Provides framework to perform processing over data stream
- Only operates on data starting from whenever the last batch processing job was completed
- When historical data is needed, use estimators and summarizing statistics

Data may not be totally accurate due to the inability to look up and calculate things based on historical data. It may not be obvious what values are needed in advance, depending on the algorithm being implemented. In that case, one may need to rely on precalculated summarizing statistics or estimators in order to make progress. Furthermore, the real time processing system is more complex in that if a job fails, it cannot start back up from historical data without violating its real time constraint. As such, it needs more fault tolerance and redundancy built into the system assuming the real time aspect cannot be temporarily suspended in the event of operational issues.

3.2.1.3.3. Indexed Real Time View

The indexed real time view is the output of stream processing. It provides the following:

- Stores output of stream processing framework for analysis
- Accessible through a common query layer

Similar to the batch views, writing the real time views to their data base often requires random writes. This is another major complication in the data processing pipeline which can cause backup of data flow if it is not carefully designed.

3.2.1.4. Query Layer

The query layer is intended to provide a common query interface to both indexed batch views and indexed real time views. It should provide the capability of merging the two views to combine the highly accurate forensic view data with the more recent real time view data.

3.2.1.5. Enforcement

We envisioned an enforcement dashboard for analysts to view what was going on in the system and take some action through appliances like firewalls or through the TA2 components.

3.2.1.6. Feedback

We envisioned the TA2s having a mechanism for controlling what TA1 sensors should focus on in data collection. This would allow for the data flow from TA1s to either be limited to what TA2s wanted, or to ask TA1s to gather more data on specific focus areas.

3.2.2. Lambda Architecture vs. Kappa Architecture

The original STARC proposal was to implement a Lambda architecture using agreed-upon tools across TA2s. During the project, a new type of architecture, the Kappa architecture (https://www.oreilly.com/radar/questioning-the-lambda-architecture/), started to gain momentum. Its creator claimed that in their opinion, stream processing is not inherently less accurate than batch processing, but rather the tooling available at specific points in time has been the limiting factor. Furthermore, the author believed that the cost of creating and maintaining code for both stream and batch processing at the same time is not worth the gains. The architectural differences between the Lambda and the Kappa architectures are shown in Figure 3 and Figure 4.

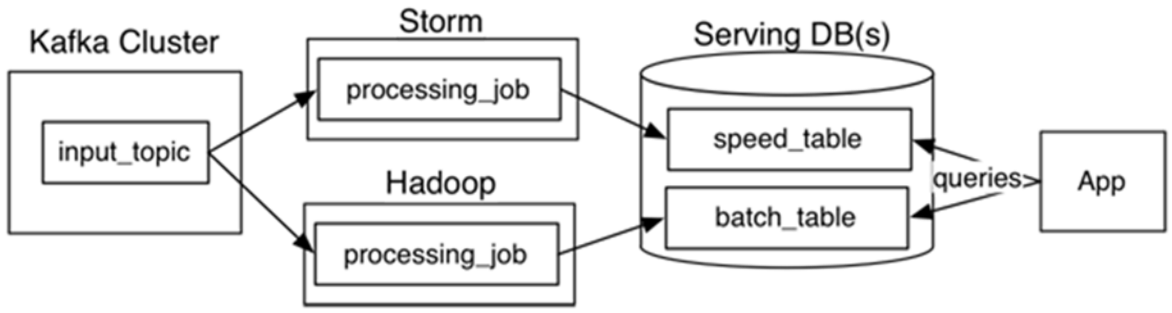


Figure 3. Example Implementation of the Lambda Architecture

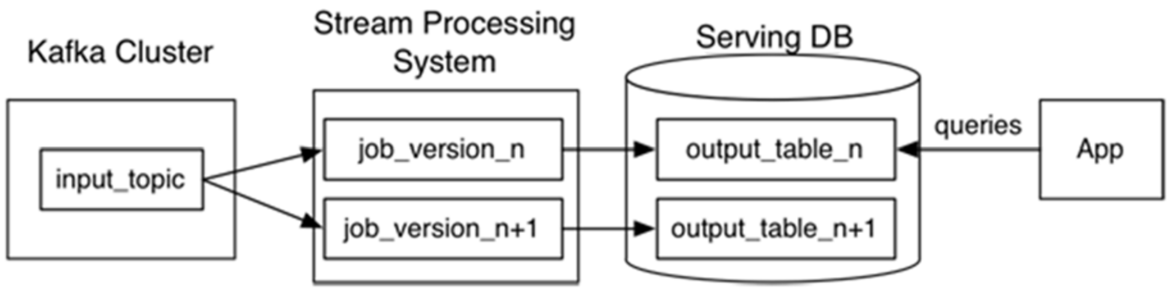


Figure 4. Example Implementation of the Kappa Architecture

About half way through the program, we presented the Kappa architecture to the TA2 teams. We suggested that whether or not stream processing was capable of performing accurate calculations for their algorithms was largely up to those teams, as they should have a good sense of what kinds of things they were doing at that point of the program. The major upside to using the Kappa architecture was that it was almost exactly what all of the teams were already doing. It is a simple solution which still extends to working in many operational cases, and that includes the cases we went through for the first few engagements. The TA2s were only streaming data from Kafka and processing it as it came in. If their code had a bug in it or if a transient crash occurred, they would simply reprocess the data from the beginning.

One major downside to the Kappa architecture that we saw for a program like TC was that it relied on replaying data for a limited history, as the data is only stored in a place like Kafka. The inherent problem is that Kafka topics, each a container of a single data stream, are not intended to horizontally scale sequentially as the stream grows. They can be split into partitions, each of which can reside on different machines, but those partitions are parallel processing constructs, and not a mechanism for expanding an ordered data stream across multiple machines. We felt that forensic analysis was a core piece of what an analyst may need to do once an APT was detected, and that the storage limitations of any single machine in a Kafka cluster should be independent of the desired retention policy of data to be used for forensic analysis. Therefore we suggested that the data still be hooked up to a long term data store as shown in Figure 5. The thought was that if long term reprocessing was needed, then data could essentially be "paged in" from the long term data store into Kafka for replay if the team preferred that approach. This could be used to process a window of historical data, or it could be used to reprocess all data from a point up until the current time in the event of an analysis layer bugfix. As long as data could be processed faster than new data came in, then eventually, the paged in data would overlap with the live topic, at which point the consumer could switch back to processing the live topic, ignoring records which it had already processed from that topic and starting from those it hadn't processed yet until it caught up to real time.

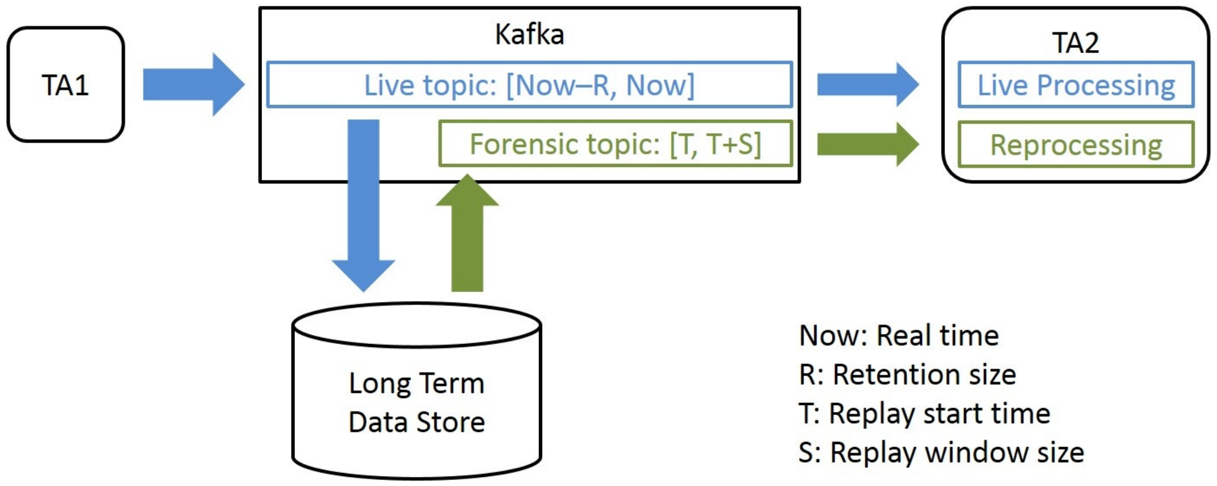


Figure 5. Proposed Architecture Supporting Both Real-Time and Forensic Processing

3.2.3. What Was Implemented

3.2.3.1. Smart Collectors

Kafka already had the machinery present to give us metadata such as timestamps and host IDs. In particular, Kafka tracks timestamps on either the producer client side or the server side. Also, we decided to use a single topic per producer host, and thus we were able to use the hostname of the producer as the Kafka topic name. A Common Data Model (CDM) supported a common schema and syntax across all TA1 producers. Kafka APIs were both robust and easy to use, with our API wrappers making integrating with our Kafka clusters even easier due to pre-configuration in the distributed code. Each TA1 was able to easily integrate with Kafka, so the need for smart collectors was alleviated.

3.2.3.2. TA1 Translators

While we did not use smart collectors, most TA1 teams opted to use translators running as separate processes, sometimes even running on separate hosts. For example, the Five Directions team used a process running on a separate host to take in records using their internal format and convert it into a CDM data stream. This function is very similar to one of the intentions for the smart collectors, although the sole focus is really taking a TA1-internal format and converting it into the common data format.

3.2.3.3. Forensic Layer

3.2.3.3.1. Master Data Store

We explored setting up HDFS as a long term data store, and we set up the code to automate and manage deployment. We also deployed master data stores for each TA2 to use leading into Engagement #3. We ended up not using these deployments, however, and we ultimately shifted away from further development of the master data store. For the TC program, the length of the engagements meant that the data set sizes were not overly large and we could easily fit all topics with data replication into our cluster of 6 Kafka servers each with 4 TB of data.

Prior to Engagement #3, we proposed artificially limiting the data retention in Kafka to some period of time shorter than the planned engagements with the intention that the data would be available in the Master Data Store. The TA2 teams expressed concern about us adding in this new change, in particular because we were still developing pieces for integration between Kafka and the data store and we were already experiencing delays and issues integrating the TA1 technologies. Because of this, we did not finish setting up the infrastructure to automatically write from Kafka topics into HDFS prior to Engagement #3, opting to focus on other integration issues instead.

We still believe that the long term data store is a critical component for maintaining both scale out for data retention and fast sequential writes in a real deployment. Because of the limited scale of the TC engagements, it was not critical for the program evaluations due to the duration of the engagements and the relatively small volume of data generated.

It is worth pointing out that the MARPLE team ultimately had their own version of a master data store that was implemented on top of HDFS, and they wrote to that data store when consuming from Kafka. The RIPE team also created their own master data store by consuming the CDM data, converting it to their own format, and writing that data into a file. Each TA2 had 16 TB of disk space at their disposal, but even this seemed limiting at times due to the desire to keep historical data which had been processed from previous engagements.

3.2.3.4. Batch Processing Framework

The choice of batch processing framework was made irrelevant by the fact that two of the TA2 teams chose to focus on a TA3 Kappa-style architecture, and the third TA2 team (RIPE) not participating fully in the last two engagements. If we had proceeded with a TA2 team with a Lambda-style architecture, then the choice of framework would have depended on what the TA2 wanted based on their preference of language and platform. We likely would have set up a stack for each Lambda-style TA2.

3.2.3.5. Indexed Forensic View

This also ended up being driven by TA2 decisions. The RIPE and ADAPT teams each implemented their own forensic views of the data by building up private databases in the real time pipeline and used the query language provided by their database framework to do their analysis. At least one TA2 team changed their choice of database implementation over time as they encountered performance and feature issues, but many of them ended up using something like Neo4j. Neo4j seemed to have many convenient properties for analysis, but it makes use of random writes, which exacerbated some of the input/output rate challenges that we saw during the engagements. This was discussed with the TA2 teams over the final three engagements but they had, by the beginning of engagement four, solidified their design decisions.

3.2.4. Real Time Layer

3.2.4.1. Real Time Substrate

We implemented a Kafka cluster for this component. Kafka ended up being the perfect tool for the real time substrate, and it was the one constant that remained steady throughout all engagements of the program. We ended up using a very limited feature set of Kafka, though, in order to meet the needs of the program. In particular, Kafka provides a mechanism called partitioning which is a method to balance the set of produced messages across a set of servers and also allows for parallelization of consumption.

Kafka's partitioning does have a drawback in that it removes strict ordering of records across partitions, something that the TC performers felt was necessary. Reconstructing this strict ordering could have done by using timestamps created on the producer side and placing them into each CDM record, but in order to reshuffle the data to get the ordering back, we would have needed to send the data through a single bottleneck, taking away the benefits of parallel processing.

3.2.4.2. Stream Processing Framework

The TA2 teams ended up choosing their own processing pipelines. For the most part, it seemed that they were not using the standard commercial streaming engines which we had envisioned (such as Apache Spark or Apache Storm) and built their own custom processing engines. This was a point of discussion leading into the first engagement, with TA3 wanting to let the TA2 teams explore what worked for them before trying to find a common framework for all teams to agree on. Ultimately, each TA2 explored different directions based on their experience and, once they decided on which one to use they were unwilling to change. We believe more familiarity with the standard streaming engines on the part of the TA2 developers at the outset might have made a decision to use one of them more palatable.

3.2.4.3. Indexed Real Time View

Similar to the indexed forensic views and because each TA2 chose their own stream processing framework, this was also a per-TA2 decision. Our observation was that their views were not precomputed, but rather analysis was performed on the fly based on triggers such as point detectors.

3.2.5. Query Layer

Because TA2 teams each controlled the prior stages of the architecture, they ultimately decided what their query language looked like.

3.2.6. Enforcement

Various versions of an enforcement dashboard were discussed during the program, including a Nagios-like green/yellow/red interface for viewing network and system status. While we did not implement an enforcement dashboard, we tested system-level elements of policy enforcement through a series of demos which is documented at Policy. This enforcement mechanism put an enforcement point in the data plane which was much more flexible than a simple firewall. It allowed for very fine-grained policy to be enforced based on data flow throughout the network.

3.2.7. Feedback

A TA2 control plane into TA1 systems was explored before the first engagement, but both TA1 and TA2 teams generally seemed wary of it. The FAROS team explored options for their own feedback plane to control the level of granularity of their data collection, but this ultimately was not used. Both the FAROS and THEIA teams explored control planes for replaying data, but this was more about augmenting data with finer grained details rather than controlling data collection granularity at run time.

3.3 Common Data Model (CDM)

The TC program aims at early detection of APTs and root cause analysis by making otherwise opaque enterprise systems transparent [5].

Transparency is achieved through granular tagging and tracking of causal relationships among programs and data across the enterprise's communication/computation planes, assembly of these dependencies into end-to-end behaviors, and reasoning over the behaviors both forensically and in real-time. A wide breadth of techniques exist for building causal relationships among activities. These may be coarse-grained such as those based on audit logging systems (e.g., MPI [6] or MCI [7]) or finer-grained such as with information flow taint tracking systems (e.g., TaintDroid [8] or Panorama [9]). These techniques also differ by platform (e.g., desktop, server, mobile, embedded) and layer of the software stack.

The primary technical challenge we had to address was how to record and preserve these causal relationships to support the largest breadth of such tagging and tracking technologies. Commercial forensic technologies tend to use the logging capabilities afforded by the operating system audit and log features, which tend to be either very find-grained or very coarse in nature.

3.3.1. CDM Goals

The TC performers as a whole developed a set of primary goals for the model. They are:

- capture metadata about both the data and control planes with high fidelity and various granularities,
- allow the TA1 performers to naturally express their information without adding undue burden on them, and
- contain the semantics and information needed to enable real-time detection and forensic analysis by TA2 performers.

Three of the TA2 teams, ADAPT, MARPLE, and RIPE, submitted proposals that could each support their tool as then currently envisioned. The TC participants worked as a group to consolidate these proposals over a number of months into a single usable model.

3.3.2. CDM Conceptual View

The agreed upon model was developed as a combination of the three proposals, and then added to over time based on experience. The basis of the initial model was the MARPLE event-based model where events are first-class entities, along with control flow (order) and data flow (tags). The hierarchical relationships between subjects and objects were proposed by the ADAPT team in its PROV-TC model, allowing the model to capture mixed granularities and containment semantics. The RIPE team added the concept of representing events and data as vertices in a graph and using this representation to explicitly show the relationships. The core data model is shown in Figure 6.

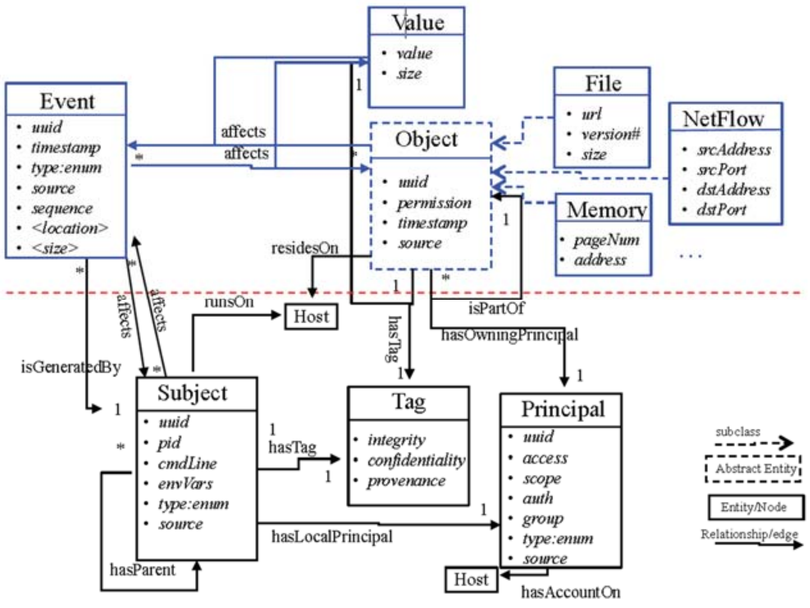


Figure 6. CDM Common Graph Data Model

3.3.2.1. Events and Subjects

Events represent actions executed on behalf of subjects. Events could include system calls, function calls, instruction executions, or even more abstract notions representing a "blind" execution such as black boxes that are not instrumented (more shortly). Events are the core entity in the model and they are the main abstraction for representing information flow between data objects, and subjects.

Events are assumed to be atomic so there is no direct relationship between events. Instead, events are related to other events through the affected subjects and objects.

Events can have different granularities but they are still atomic. For example, a function boundary may execute many atomic events within it, hence the function entry call would be captured as an event and so would the function exit call. The function boundary itself may be captured as a subject in this case if needed. If the function boundary is not instrumented however (black box), the function execution may still be captured as a "blind" event that relates the input and output subjects and objects (more on blindness shortly).

The event sequence represents the logical order of the event relative to other events within the same execution context. This guaranteed which is why we kept sequence. The event's location and size attributes (from ADAPT) are optional and they refer to the location and size of the data affecting the event (e.g., the read offset in the file and the number of bytes of data read for the read system call event). Subjects represent execution contexts and include mainly threads and processes. They can be more granular and can represent other execution boundaries such as functions and blocks if needed. For example, a function within a thread within a process can be represented as three subjects where the function *hasParent* thread, and the thread *hasParent* process. Here the function subject is an execution boundary that is more granular than the thread (while we can represent this granularity, as of now it is unclear if and when this will be useful).

An event *isGeneratedBy* a subject. It may affect other subjects such as when the output of a system call (event) forks a process (subject). A subject may affect an event as well such as when the subject is the input to the event as is the case with the kill system call.

An event affects objects and values. These may be the outputs of the event execution. Similarly, an object or value can affect an event such as when the objects are input arguments to the event.

a. **Events as First-Class Entities:** In order to understand the rationale behind choosing events as first class entities, it is worth describing the alternative first. An alternative model (such as W3C PROV) treats events as binary edges between subjects and objects. These edges, such as "*wasDerivedBy*", are annotated with event types and other metadata. While this is valid model, we found that treating events as edges creates unnecessary complexity in the model. This is especially true when events relate more than two objects and/or subjects. In this case, simulating this n-ary relationship using binary relationships complicates things. Instead, n-ary relationships may be natively supported in the Common Data Model since events are vertices in the graph and they can be related to an arbitrary number of subjects and objects. It is important to note that it is straightforward to map this event-based model to a W3C PROV model if a TA2 should wish to do so: each event would be decomposed into a subset of the predefined PROV-DM relations. This mapping may or may not be lossy depending on how accurately the PROV-DM model would be able to capture the rich and complex semantics of system events. To avoid this mapping, we found it more natural and intuitive to use a more native event-based view.
b. **Sensor Blindness**: An interesting discussion point raised by the ADAPT team was sensor blindness i.e., what happens when we don't have events (say with black boxes) but a TA1 technology is still able to capture the information flow into and out of the black box. We discussed two options here. In the first option, we can add edges directly between objects and between objects and subjects to represent this information flow. The other option that is more consistent without event-based model is to create a special event type for "blind" events that still serves as the main entity for connecting the affected objects and subjects. We agreed on the latter model; hence, data cannot flow directly between objects or subjects without going through an event.

MARPLE and ADAPT each identified a superset of events corresponding to system level calls. Over time, we agreed on these event types and consolidated their definitions.

Finally, a subject in the MARPLE proposal had attributes for "reference ObjInfo representing executable" and "list of references to libraries". We think these are already captured in the model as input objects to the event that created the subject so these were not included.

3.3.3. Objects

Objects, in general, represent data sources and sinks which could include sockets, files, memory and variables, and any data in general that can be an input and/or output to an event. This model expands the definition of objects to explicitly capture the flow of data inputs and outputs to and from events. The model also treats event arguments as first class objects (instead of attributes of the event entity) where each object may have its own provenance again explicitly showing the flow from inputs to outputs. Syscalls, functions, and instructions have inputs and outputs which could be objects or subjects.

We identified some key object attributes. The object timestamp is the creation time. The file URL is its local or remote path/location. Files have version numbers. As files get updated, a new file object is created with a new version number.

a. Transient data: One of the main discussion points was whether we should model arguments as Objects or whether we should explicitly distinguish transient data (e.g., arguments) from more persistent data (e.g., files). We agreed that it makes more sense to distinguish the two mainly because they have very different attributes (arguments don't have ownership and path attributes and have a value attribute). Just like objects, values have tags and affect (and are affected by) events.
b. Granularity and types: We discussed representing objects at various granularities. ADAPT modeled object containment using the *isPartOf* relationship associated with the object/artifact. This allows us to represent a set of objects being part of a parent object, such as when writing buffers (objects with different tags) to a file (parent object). To explicitly differentiate between different types of objects, we also decided to subclass the abstract object into Files and Network Flows and Memory objects (and others we shall identify) instead of modeling those using a type attribute of the object entity. This is mainly to keep the model clean given the difference in attribute sets among the object types. Finally, we also discussed whether we should model mobile phone sensors (GPS, camera, gyro, etc.) as a separate entity from Object, called Resource (as proposed in ADAPT). One observation here is that the sensing subsystem in Android for example might have enough differences to warrant this distinction. The counter argument is that Android uses Linux underneath and one might be able to get away with an Object abstraction for all such data devices. For now, we decided not to make the distinction and keep it for future consideration.
c. Provenance: As discussed earlier, we explicitly model the relations between events and objects and subjects using the affects relationship which is intended to identify the nature and semantics of the relation. The explicit relations between objects and events represents direct data flow (top part of Figure 6 in blue). ADAPT defined a location attribute for the objects. This model made location an attribute of the event instead since it is specific to the event (such as the location in a file where the read or write event happens). The location may be the offset being read from a file or socket. We similarly added size as an optional attribute of the event.

3.3.4. Tags

The tag design was primarily fleshed out by the MARPLE team. The primary motivation for tags is the ability to capture flows at a finer granularity and with increased precision. Specifically, tags can capture:

- *Increased precision*: Tags should be able to capture information flow from a (strict) subset of inputs represented in a causality graph. For instance, a subject may read from 100 files, and then write one file. Rather than reporting that the output depends on these 100 inputs, a TA1 system can indicate that it depends only on 10th and 97th input files.
- *Nature of dependence*: The behavior of the system can be understood in terms of a series of operations on inputs to produce outputs. Fine-grained tracking can identify the effect of these operations on data flows. These effects can be described using tag operations, which can capture:
    - common operations on data such as concatenation, mixing, compression, and so on,
    - declassification and endorsement, the central operations used in information flow control literature, and
    - strength of dependence, e.g., control dependence or implicit dependence.
- Flows involving internal (unreported) objects: Especially for in-memory objects, not all updates and flows may be reported by a TA1 system. A TA1 system will internally keep track of these updates, and can then make the information available at the time when these objects contribute to an event argument.

Tag are associated with subjects, objects, and values.

We identified three types of tags: (i) provenance (source dependence), (ii) confidentiality, and (iii) integrity tags, all of which can exist in parallel. One can think of these tags forming three parallel planes each capturing a different aspect of data flow. The integrity (respectively, confidentiality) tag may be used to specify the initial integrity (respectively, confidentiality) of an entity, or to endorse (respectively, declassify) its content after performing appropriate checking/sanitization.

a. *Provenance tags*: Defines specific data sources (inputs). A tag identifier (an integer) is bound to a source and used by the tracking system to capture dependence on this source input. We sought a simple representation that was expressive enough to capture all TA1 data but not too complicated.
b. *Integrity tags*: Integrity tags can have values *untrusted*, *benign*, or *invulnerable*. The integrity tag *untrusted* indicates that the content is potentially malicious, while the tag *benign* indicates that it is from a source that is non-malicious. The tag *invulnerable* is applied only to programs, and indicates that the program is trusted to be free of vulnerabilities.
c. Confidentiality tags: Confidentiality tags can take values *secret*, *sensitive*, *private*, or *public*. The label *secret* is meant for entities such as passwords. The label *sensitive* is used for all confidential data that needs to be protected, while the *public* tag is reserved for information that can be freely shared with the world. Non-public (but not confidential) data is designated *private*.

3.3.5. Extensibility Points

As part of the initial definition, every participant agreed that creating a complete and concise model on paper only is very difficult and that we would have to include extensibility points to support the evolution of the model based on experimental results.

3.3.6. Typed Schema Definition

TA3 is developed a versioned and typed schema that codifies the above conceptual model. The actual data model did evolve based on TA1-TA2 discussions over the course of the program. The schema definition was maintained in parallel to match the latest version of the data model.

The schema is specified using the Apache Avro Interface Description Language (IDL) language. Avro IDL is a high-level language for authoring Avro schemata. It provides a familiar feel for developers in that it is similar to common programming languages like Java, C++, and Python. Also, it is similar to IDLs in other serialization frameworks such as Thrift and Protocol Buffers. Avro provides a tool to convert the Avro IDL specification into an Avro schema in JSON format, which is used by Avro serialization frameworks for automatic serialization and deserialization of objects that conform to the schema. Avro (and the TA3 APIs) additionally provide the tools for automatic syntax checking to verify the data conforms to the typed schema.

Appendix A is an excerpt of the typed schema based the common data model of Figure 6. This excerpt includes Subject, Event, Object (and subclasses), and Value entities, and all edges between them.

3.3.7. CDM Evolution

Since the first implemented version (0.5), the CDM has had 15 subsequent versions as we learned from our shared experience preparing for and during the engagements. Some changes made to the schema included:

- The Value entity was ‘demoted’ from a first class entity to an attribute of an Event
- We added a SensorObject to represent Android sensors, genericized that to SourceObject, and finally to SrcSinkObject
- Instead of depending solely on a Host UUID and Stream ID in a single StartMarker record at the beginning of a stream, these two values are now included in every record to assist TA2s with associating event records with each other.

3.4 Engagements

The TC program used a series of five engagements providing a structured series of increasingly difficult scenarios to validate the design and implementation of each of the components provided by the TA1 and TA2 performers, the infrastructure provided by BBN as the TA3 performer, and the procedures and processes used by all performers as a group. Table 3 provides the dates for each of the engagements.

Table 3. TC Engagement Dates

Engagement	Date
1	Sep 6 – 9 2016
2	May 1 - 27 2017
3	April 2 – 13 2018
4	November 9 - 21 2018
5	May 7 - 17 2019

3.4.1. Engagement #1

3.4.1.1. Goals

Engagement #1 was mostly aimed at creating a steel thread for the initial versions of the TC technologies. We wanted to exercise the system to make sure that information could flow from TA1s through to TA2s, and that attacks could be performed at detected at some level. For this engagement, the red team was carrying out two different "scenarios". Bovia and Pandex were supposed to represent different enterprises being targeted by different actors carrying out different campaigns. These scenarios were developed by TA4 and were used as input into TA5.1's attack planning.

3.4.1.2. Network

In the lead up to Engagement #1, the network evolved while we determined how to create a design that would best integrate with the Lincoln Adaptable Real-time Information Assurance Testbed (LARIAT) [10] . LARIAT was used to create a realistic, but controllable image of the internet to support testing and malware injection. Initially, the plan was to set up an entirely closed network with LARIAT providing a simulated Internet backend. As we received feedback, we decided it would be good if the hosts in the test range had the ability to reach out to resources on the real Internet for development purposes (e.g. pushing and pulling code to and from external repositories). After this, we decided that TA3 would run a router which would act as the default gateway for all TA1 target hosts in the test range, and the routes could be switched to point at either the real Internet or the simulated Internet backend on demand. This approach was both coarse grained (at least without some type of source-based routing), potentially bandwidth limiting, and difficult to implement due to specific restrictions imposed by LARIAT. Specifically, LARIAT provides a router that either needs to be the first hop, or else it needs to be set up to perform Border Gateway Protocol (BGP) transactions, which is something we did not want to set up. Finally, we decided to set up a system where the hosts would switch their default gateways in order to switch networks. In addition to this change, we also needed to change the Domain Name Service (DNS) name servers used by each host in addition to flushing old entries from the cache. Changes between Internets were performed manually by TA3 on request. More details on the system that performed this can be found in the Infrastructure section.

After sorting out how to connect to LARIAT and the real Internet, we ended up with four VLANs, as can be seen in Figure 7.

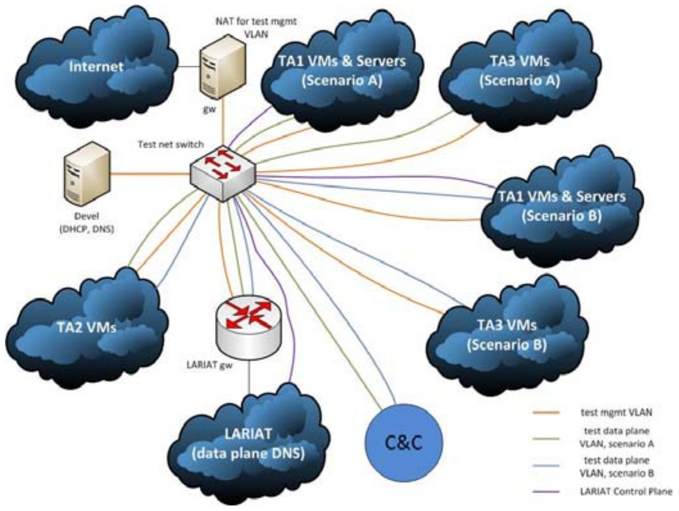


Figure 7. Network Diagram for Engagement #1

The primary concepts incorporated in the Engagement #1 test network design were:

- A test management VLAN, which was the only VLAN which could reach the Internet
- An experimental data plane VLAN for the Pandex scenario
- An experimental data plane VLAN for the Bovia scenario
- A LARIAT control plane VLAN for driving the LARIAT agent on Windows hosts

Over the course of the first engagement, we found that the LARIAT agent was disruptive to attempting to perform attacks, and therefore the agent was ultimately dropped, as was the control plane VLAN for LARIAT.

3.4.1.3. Hosts

In Engagement #1, there were two scenarios, and the test range was set up in a way where both scenarios were set up separately on different hardware. The idea was to do as much set up as possible in advance, and to not have to reset anything between running the different scenarios. As such, there were two instances of each TA1 and two Zookeeper and Kafka clusters. Each TA2 used a single cluster due to limited hardware.

To set up the TA1 systems, the TA1 teams hand-created images and provided them to TA3. TA3 converted the images to the proper format for KVM to run them and updated the images by hand to fit into the test range network architecture. A single account was used to log into the systems for setup, and another account was intended to be the account which was monitored during the engagements. The images were augmented by hand by TA1 teams and TA5.1 in order to be ready for the engagement. This led to problems in reproducibility, manifesting itself as conflicts during setup and mismatches between the two replicated hosts which largely should have been identical. Furthermore, there was at least one instance where a configuration file was in an unexpected state and no team which had access to the system claimed responsibility for the change. These issues demonstrated a need for improved reproducibility and accountability on the systems.

3.4.2. Engagement #2

3.4.2.1. Goals

Engagement #2 was the first attempt to move beyond simply exercising the system. It increased the level of attack difficulty while also trying to ensure that problems encountered in Engagement #1 were addressed. Once again, two scenarios were used.

3.4.2.2. Network

In Engagement #2, the LARIAT control plane VLAN was removed and the LARIAT agent was replaced by custom scripts running on each end host. Kafka traffic from the TA1 hosts was intended to not be in scope for this engagement, which was an oversight from the previous engagement, so a new VLAN was introduced for Kafka traffic only. There was still a VLAN per scenario. Switching between the real Internet and the simulated Internet was automated and brokered through a web service reservation system. Figure 8 shows the design used for this engagement.

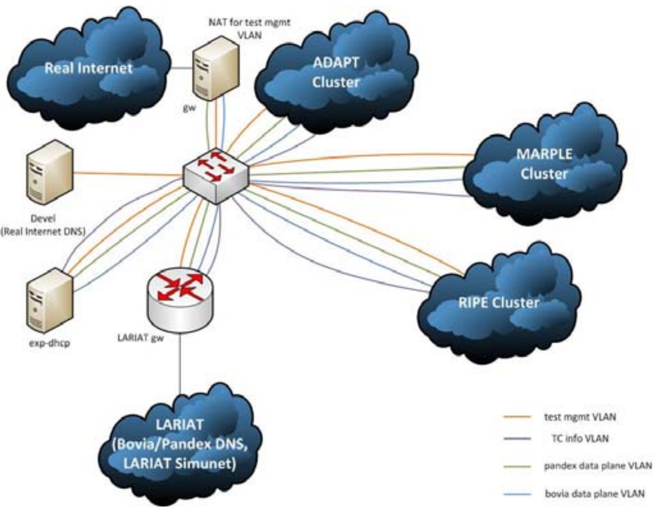


Figure 8. Network Layout for Engagement #2

3.4.2.3. Hosts

For host allocation in Engagement #2, an emphasis was placed on fairness of resource allocation for TA2s. There were TA1 instances, Zookeeper clusters, and Kafka clusters allocated for each TA2. To prepare for the second scenario, all the hosts had to be updated after the first scenario.

To prepare the TA1 systems, some TA1s delivered images and others worked with images that TA3 provided and they tried to create more reproducible mechanisms to perform their setup steps. TA3 tried to create hand-made scripts for each TA1 that would perform basic reproducible setup of their systems in a way that was compatible with TA5.1 background activity generation and attacks. This approach was an improvement over the previous approach, but it was very time consuming and as deadlines approached, it ultimately fell back to manual changes for teams which were late to meet deadlines. Teams that manually changed their systems once again had issues with systems not being identical across their three instances. In addition to these changes, each team was given their own shared accounts which could be used for setup. The goal here was to introduce some level of improved accountability when things changed on a system.

3.4.3. Engagement #3

3.4.3.1. Goals

Engagement #3 introduced cross-host tracking for the analysis teams to follow. In addition, the Kafka traffic was added to the scope of the analysis team, and outsider attacks were also added to the scope. The addition of cross-host tracking and in-memory attacks increased the difficulty for full detection. Engagement #3 still included multiple scenarios, but the infrastructure for each scenario was converged.

3.4.3.2. Network

Engagement #3 was the first engagement that introduced a simplified network approach with only two VLANs. Figure 9 shows that there was once again a test management VLAN which could reach the real Internet, but the only other VLAN was a single converged experimental data plane VLAN. The reasons for this change was that there was no need to split out traffic for the scenarios into different VLANs, and Kafka traffic was in scope for the engagement.

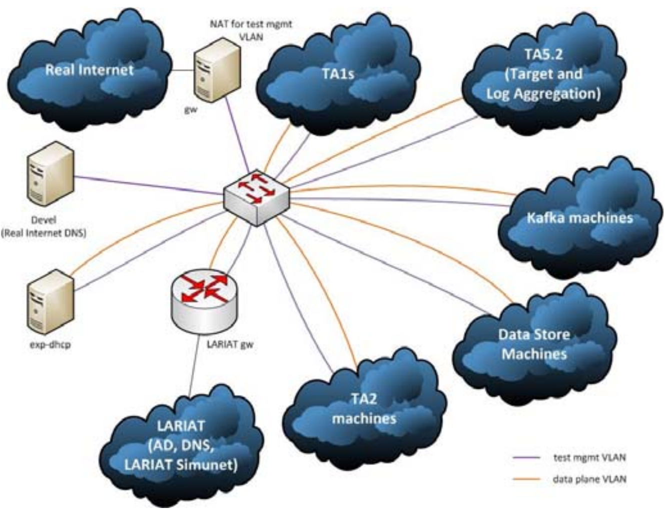


Figure 9. Network Layout for Engagement #3

3.4.3.3. Hosts

Engagement #3 tried to maintain a per-TA2 level of fairness, but the number of hosts and the maintenance burden of separate TA1 and TA3 deployments per TA2 was too high. In addition, Engagement #3 targeted the introduction of a long term data store cluster per TA2, which is what many of the hosts were earmarked for. Ultimately, the long term data stores were not used in the engagement but this did impact the availability of hosts. There was ultimately a single Kafka and Zookeeper cluster, and for most TA1s there was a single instance of their system. The only exceptions were that THEIA's replay systems were replicated to have an instance per TA2 for fairness per TA2, and FAROS systems were also replicated for TA2s, given their plan to have TA2s trigger coarse-grained and fine-grained collection modes.

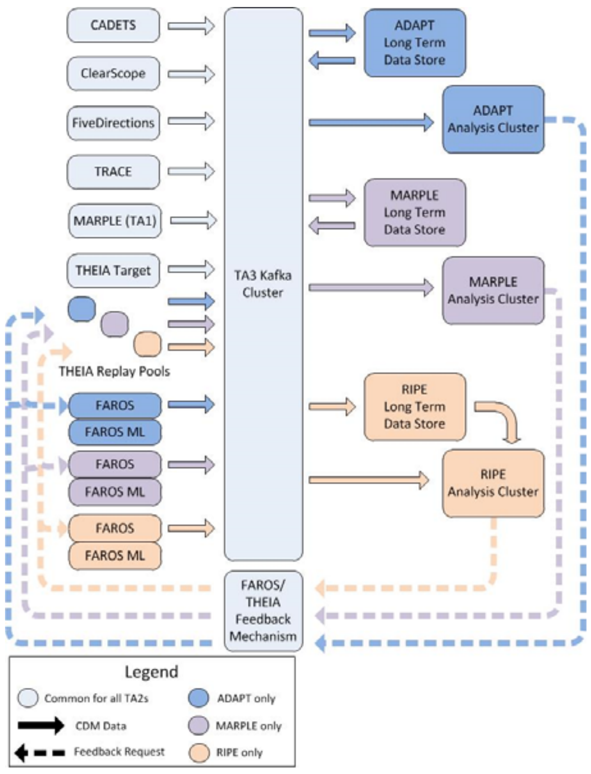


Figure 10. Engagement #3 Network

In order to set up TA1 systems and TA5.2 systems, Continuous Integration (CI) for image generation was introduced prior to Engagement #3. TA3 created some canned images with basic platform requirements needed by individual TA1s, and those were fed into processes which automatically generated images which could be deployed. TA3 also revamped the provisioning system to be part of the tc-salt-services package, which leverages SaltStack for idempotent configuration management. TA5.1 significantly changed their background activity generation to seem more like realistic user traffic at a low level. More TA1s delivered their own provisioning scripts, but the process for TA1 setup was largely still very manual. Engagement #3 was ultimately delayed due to late code deliveries and problems with testing during deployments.

3.4.4. Engagement #4

3.4.4.1. Goals

The goal of Engagement #4 was to focus on depth for each of the individual TA1s. It focused on isolated attacks on specific days for a pair of hosts running a single TA1, and then a separate week of isolated attacks on pairs of hosts running pairs of pre-chosen TA1 pairs. The ability of TA1 technologies to be used for TC self-monitoring was also evaluated.

3.4.4.2. Network

The network design was not changed significantly from Engagement #3.

3.4.4.3. Hosts

Engagement #4 was a one-off engagement where the focus was on probing on each TA1's depth. As such, each TA1 was set up with a pair of hosts which could be the focus of an attack within a single day. The following week, one of those two hosts would be used in a day focused on a pair of TA1 hosts. For this work, a single Zookeeper and Kafka cluster was used. However, a secondary Kafka and Zookeeper cluster was set up on top of CADETS hosts to evaluate the feasibility of TC self-monitoring. The original plan was to set up Kafka in a way where CADETS hosts were mixed with non-CADETS hosts in a single cluster in a way where each topic would have a replica on one regular Kafka machine, however it was found that Kafka on CADETS had significant performance issues and some stability issues, to the point that it had to be taken out of the critical path for the stability of the engagement. In the secondary setup, Kafka on CADETS was set up to consume from the primary cluster.

Image generation was once again provided through CI, and both TA1 and TA3 provisioners were refined and updated for new TA1 features. TA5.1 provided some automation of basic attacks as well as their background activity generation management through Jenkins.

3.4.5. Engagement #5

3.4.5.1. Goals

Engagement #5 focused on scaling up the infrastructure to provide an increased surface area of hosts for analysis and a larger data flow to analyze in real time. In addition to multiple instances of each TA1 system, uninstrumented hosts were also introduced into the infrastructure to test the ability of TA2 teams to detect cross-host activity when an intermediate host provided no data.

3.4.5.2. Network

The network design was not changed significantly from Engagement #4.

3.4.5.3. Hosts

Engagement #5 was largely about scaling up. Each TA1 team had three instances of their target hosts all running at the same time. In the case of THEIA, they also had a replayer host per TA2. A single Zookeeper and Kafka cluster of 6 nodes was used, and was sufficient for the engagement. It is worth noting that the RIPE team did not participate, which ultimately reduced the burden placed on the cluster by consumers compared to previous engagements. Attacks were once again made more difficult, and that included adding in five uninstrumented hosts which could be used as pivot points for TA5.1 within a multi-host attack.

The engagement ended up running more smoothly than the previous two engagements due in large part to a much more strict set of rules for meeting the timeline leading into the engagement. For the most part, the image generation and provisioning process had also been in place for a while, and thus only needed to be tweaked and refined to accommodate new features. TA5.1 also continued to improve the automation of their processes.

3.4.6. Challenges with Reproducibility

There were a number of challenges across many of the engagements related to achieving reproducibility in system provisioning. First of all, some performers had processes as part of their build and system setup which were very difficult to automate. Those things would typically be documented and performed by hand, but that could still lead to some amount of inconsistency across deployed hosts. To a lesser extent, some performers had processes which could be automated, but they took a very long time to run. Another issue we had was that the split between the image creation process in Jenkins and the post-install setup provided by the performer led to confusion about where certain changes should have been put in place. In general, the goal should be to push as many things as possible into the provisioner so it can be updated after an install with the caveat that things that always take a long time to run should probably be put into the image creation process to avoid provisioners that take too long to run. In terms of image creation, we also found that Jenkins may not have been the best tool to use. While creating images in Jenkins let us create pipelines where we could run unit tests and integration tests after the image was built, Jenkins made it awkward to track and deploy images into the test range once they had been generated.

3.5 Policy Enforcement Capability

3.5.1. Overview

The goal of the Policy Enforcement Demonstration is to investigate how well the TA1 CDM data can be used in real-time to enforce policy, rather than in post-incident forensics. We conducted a demonstration of this potential capability, rather than a full evaluation with a red team evaluation of a hardened system. Figure 11 shows the system architecture used in Engagements 4 and 5. A client accessing a policy protected web server runs on a TA1 monitored host, and the policy protected web server also runs on a different TA1 monitored host. Later iterations of this demonstration also included secondary servers, such as a database server, where the main server issued requests to the database server while handling a request. The Policy Enforcement Module (PEM) stands in between the client and server, intercepts requests, and sends policy check queries to a TA2 server. The TA2 server consumes the CDM data produced by the TA1 monitored systems from the TA3 Kafka cluster and analyzes it to answer the PEM generated policy queries.

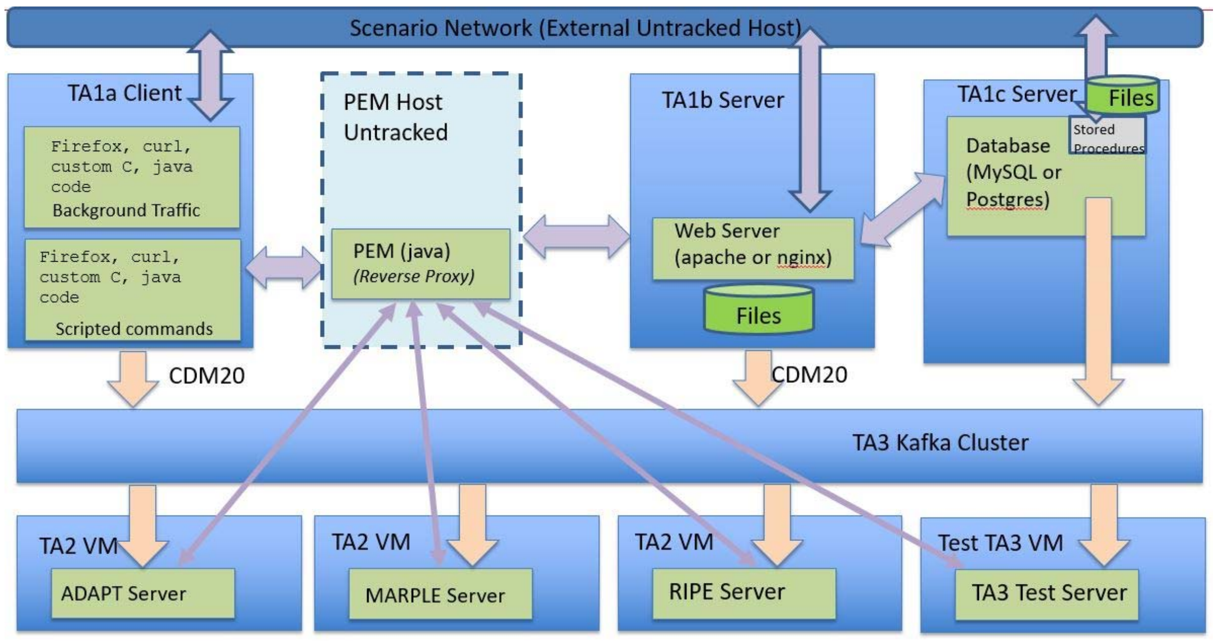


Figure 11. Policy Enforcement System Architecture

3.5.2. Motivating Example

Consider the case where a host on a network is quarantined due to suspected malicious activity. Also running on this network is a web server that should be keep up and servicing requests during the investigation of the quarantined host. We want to add a policy that blocks any requested web page that was generated using files that originated (at least in part) from the quarantined host.

Web pages are dynamically created from files on the TA1 instrumented server, and back-end requests from that server to another TA1 instrumented database server

The policy should block cases where the database process running on the database server host has read from a file that originated from the quarantined host. As requests are blocked, the administrators may restart the database, which would allow requests to continue until it reads from another such file. This policy would be temporary, and could be removed when the intrusion is fully identified and cleaned.

The calls from the Policy Enforcement Module (PEM) to the TA2 policy server will only reference the original request from web client to main server. The TA2 will then need to trace any traffic from the main server to the database server

3.5.3. Policy Enforcement Module Design

For Engagement #3, The PEM operated as a standard forward proxy. Implemented as a separate application and running on a separate, un-instrumented host, the TA1 client was configured to access the protected web server by sending requests to the PEM's host and port. The PEM would then intercept any incoming request, compare the requested URL with a white-list. Any white-listed URL is then immediately forwarded to the protected web server without any policy enforcement. For any non-white-listed URL, the PEM blocks the request and begins the TA2 policy check API, depicted in Figure 12.

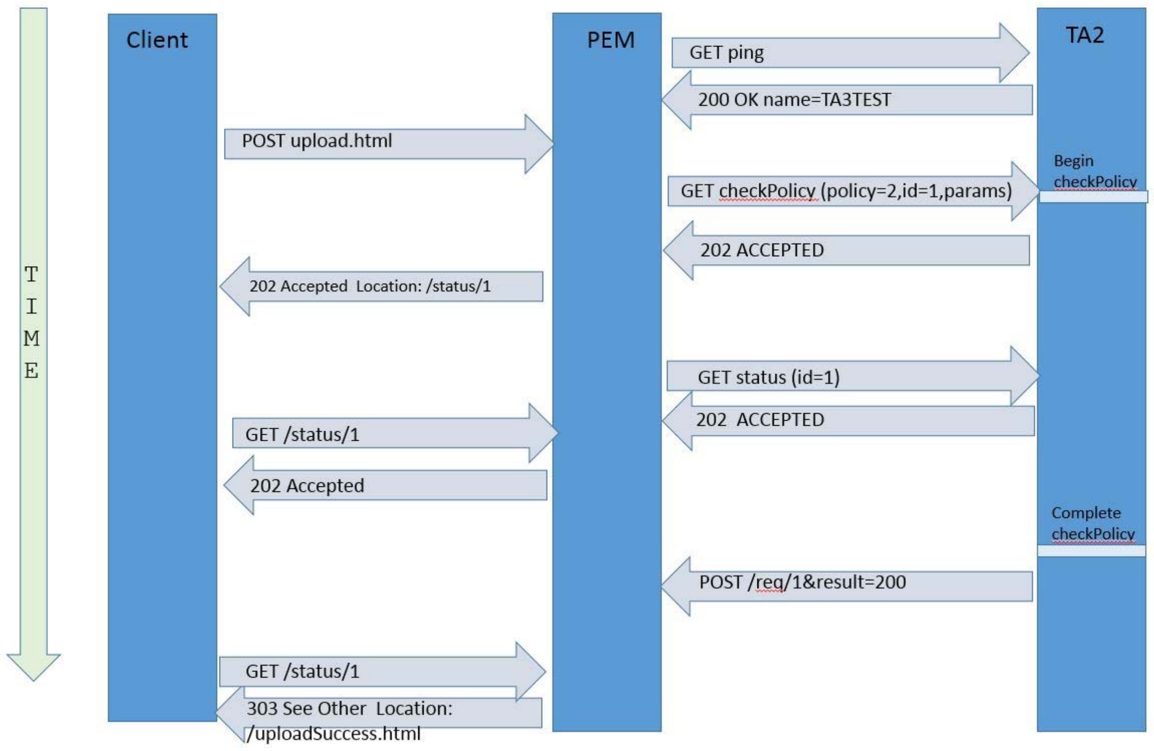


Figure 12. TA2 Policy Check Transaction

After intercepting a policy protected (not white-listed) URL, such as "upload.html" in the Figure above, the PEM extracts parameters such as the client host, port, and timestamp , along with the configured policy number and a unique identifier, and generates a checkPolicy request which is sent to the TA2 server. The policy number is configured at initialization time, and refers to one the four policies described below. We specifically did not attempt to author a general policy definition language, since such an effort would have been beyond the scope of this program. Instead we defined four example policies, and referred to one of the four when issuing a policy check message.

Table 4. Engagement #3 TC Policy Definitions

Policy ID	Policy Name	Description
1	Originating User ID	Block if the user originating the process that sent the HTTP request is NOT in a specific group or not a specific user
2	Block if tainted by a suspect server	Block if the process that sent the HTTP request ever communicated with a specific remote server
3	Block automated scripts	Block if no portion of the request originated from a definite User Interface action
4	Block network data uploads	Block uploads that contain data downloaded from a network connection

The policyCheck message is asynchronous, and returns a result immediately that indicates whether the TA2 was able to understand the request, parse the expected arguments, and is ready to start working on the answer. Our assumption was that the process of analyzing the CDM data to compute the result of the policy query could take the TA2 system a few seconds to a minute or two. In practice, due to a lower rate of data (minimal background traffic), the TA2 system answered the query in tens of seconds rather than minutes. While the TA2 system is processing the query, the PEM can check on status periodically by issuing status queries, which the TA2 system responds to immediately with a status of 202 (request in progress), 404 (request not found), or 500 (internal error). Finally, when the TA2 has completed processing and has an answer, it sends a POST message to the PEM with the result. When the PEM sees the POST, it finally responds to the client by either issuing a 400 Bad Request response if the policy check failed and the request should be blocked, or a 303 Redirect if the request passes the policy check. This redirect pointed the client to the real web server to get the real requested page. Using a redirect approach would not work in a real deployed system since a client could determine the protected web server's host and port from the redirect and use that directly in subsequent requests to bypass the PEM. We chose to use redirection here, since it was a fast solution that provided the needed functionality for a demonstration.

3.5.4. Evolution of the Policy Enforcement Demonstration

3.5.4.1. Engagement #3

For the first version of the policy enforcement demonstration, we focused on client side policies. The server was running on a separate TA1 monitored host, however no policy query required analysis of the server data. We created a background traffic script that ran on the client which generated periodic traffic to white-listed URLs on the policy protected web server. We created another script that included a set of 6 requests to the policy protected server, some of which are expected to be fail the policy check. These policy enforced requests were sent sequentially, we waited until one request had completed before sending the next. For Engagement #3, we ran each TA2 separately, resulting 6 runs of the demo for each TA2 (each TA1 as the client), for a total of 18 demo runs.

The demonstration was conducted in two phases. First, we generated forensic data as examples for the TA2s to evaluate, analyze, and use to implement their solutions. Forensic data was generated by using a TA3 generated reference implementation of the TA2 server API. This implementation always returned a PASS result. We ran each scenario with a specific policy chosen for each TA1, saved the CDM data, and provided it to TA2, along with a description of the ground truth and the policyCheck messages sent by the PEM to the reference implementation TA2 stub. The ground truth description indicated for each request, exactly what steps were taken on the TA1 client to generate the request, and what the expected result was (PASS or BLOCK).

The second phase was a live demonstration, where we ran with a live TA2 server responding to requests in real time. For this demonstration, we used the same TA1 policy pairings as the sample data, and ran through the same requests in a random order, with some additional bonus requests that the TA2s did not see beforehand.

3.5.4.1.1. The TA1 Delay Issue

There are two data pathways in this policy enforcement architecture. After an outgoing message is sent from the TA1 system to the Policy Enforcement Module (PEM), the PEM receives the message with only a tiny network stack delay, and can nearly immediately send the TA2 system a policy query. The TA2 system will then have a query to respond to, but may not yet have the data necessary to answer the query. Simultaneously, the outgoing network message on the TA1 system has to be processed by the TA1 infrastructure, converted to CDM, published to the TA3 Kafka cluster, and then consumed and processed by the TA2 system. This second pathway will take longer than the first, since there are more steps and more complex processing taking place. The largest component of this pathway, is the initial step where the event has to be processed by the TA1 and converted to CDM. This step dominates the time required so much that the other processing delays in publishing and consuming from Kafka are unimportant. Therefore, the TA2 server has to wait before attempting to answer the policy query, to ensure that enough time has passed for the relevant CDM data to be ingested.

In the initial Engagement #3 policy demonstration, we analyzed the CDM delay by using a TA3 generated reference implementation of the TA2 server API that consumed the Kafka CDM data and recorded when NetFlow records were processed. We computed the delay between when an outgoing message from the TA1 was received by the PEM (first pathway) and when the matching CDM NetFlow Record was consumed by the TA2 implementation (second pathway). The rough average delays for Engagement #3 are shown in the table below. For the live demonstration, we configured the PEM to wait twice as long as these times before forwarding the policy query to the TA2 (first pathway). For TRACE, the NetFlow delay was heavily dependent on the amount of background traffic, since TRACE has an internal queuing system that collects events and translates and publishes them in batches. We were able to modify the parameters of this queuing system, along with generating additional traffic with some scripts running on the TA1 system, to get the delay down to around 7 seconds. Further reduction would be possible with more configuration, but the timing was considered sufficient for policy enforcement.

For the FAROS TA1 system [11], we attempted to create a workaround for the fact that it only processed events in batches at 5 minute intervals. A newly generated event is added to the current 5 minute batch, and published at least 5 minutes later. Since the TA2 performers were unable to derive any policy relevant information from the FAROS PSA data, we didn't attempt to further reduce the delay for this pathway.

Table 5. CDM NetFlow Delays for TA1

TA1 Client	Policy	CDM NetFlow Delay
CADETS	1 (User)	~ 1 Minute
TRACE	4 (Net Data)	~ 7 seconds
AIA	4 (Net Data)	~ 1 second
FAROS	2 (Communication)	15 minutes
THEIA	2 (Communication)	~ 5 seconds
CLEARSCOPE	3 (UI Actions)	~ 1 second

3.5.4.2. Engagement #4

For Engagement #4, we expanded the scope of the policy demo in five ways.

1. We expanded the policies to include analysis on the server, in addition to client side only policies. Similar to client policies, server side policy enforcement were separate API calls from the Policy Enforcement Module (PEM). TA2s are required to consume either the client CDM data (for client policies) or server CDM data (for server policies)
2. Fine grained policies: Instead of returning a boolean (either pass or block), the policy checks asked more detailed questions, such as "Which user initiated the request", rather than "Did the 'admin' user initiate the request. The response was then processed in the PEM with some hardcoded logic to determine pass or block (block if the user that initiated the request was not 'admin'). This change increased confidence that the TA2 server was returning the proper response, instead of just getting lucky with a boolean flag.
3. Multiple server hosts. The main policy server may fulfill a request by accessing a database server and/or an image server running on another TA1 instrumented host. This will require *cross host tracking*. The initial check policy API call from the PEM will contain NetFlow information for the initial client A -> server B. TA2s will need to follow NetFlows from server B to server C, if they exist. Server policies will be for any host involved (B or C) (e.g. Which external hosts were contacted by any server?)
4. All TA1s must handle the client side UI policy. CADETS was handled slightly differently, due to the lack of a direct UI. We instead used the serial console and had the policy differentiate between requests that are triggered via the remote shell versus the serial console. Effectively, use of the serial console was considered to be a "UI Interaction".
5. The PEM can work with multiple TA2s simultaneously. PEM will send check policy API messages independently to multiple TA2 servers and record results separately, and compute pass/block decision based on TA2 results separately. A voting mechanism will be used to determine the final pass/block. This allows us to run a live demo once, so we can run more cases, and potentially run longer scenarios.

In addition to the policy demonstration scope changes, we switched the protected server to use a more realistic application, a sample App Store implemented in PHP with a MySQL database. This App Store application was developed by MIT Lincoln Laboratory for the DARPA CRASH program as an example for a Capture the Flag experiment. This application has built in vulnerabilities such as operation system command injection, SQL injection, and directory traversal that enabled us to use some more realistic requests, such as the example shown in Figure 13.

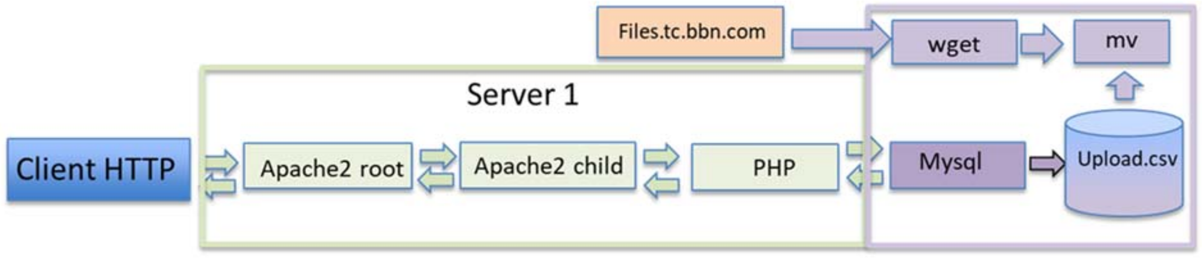


Figure 13. App Store Processing Flow

In this example, the client sends a request to the protected web server (through the PEM), which is handled by an apache2 root process on Server 1. This apache root process hands off the request to a child process, which then directly calls PHP. For Engagement #4, we configured apache to call PHP directly using mod_php, so this call wasn't an inter process communication event. That PHP processing results in a cross host call to a MySQL database running on Server 2 (in purple). The mysql call reads statements in a file called "upload.php" into a table. Upload.php was earlier downloaded from a remote host (files.tc.bbn.com) via a wget command executed directly on Server 2. That downloaded file was then moved into another directory where the MySQL process could read it, through an mv command. This is an example of a SQL injection attack, since loading this file was not intended functionality and was in fact an extra SQL command delivered through a vulnerability in the PHP code (improperly sanitized inputs). For a policy 4 check (File Origination), we expected the TA2 server to trace the cross host traffic from Server 1 to Server 2, note a read event executed by the MySQL process, trace the read file to "upload.csv", trace the provenance of "upload.csv" through the mv command to the wget command, and finally trace the NetFlow traffic from the wget command to the remote originating host.

Table 6. Engagement #4 TC Policy Definitions

Policy ID	Policy Name	Description
1	Originating User ID	Block if the originating user of any involved process was not admin (Server side policy for Engagement #4). The policy check query asked the TA2 to return the originating user for ALL processes involved in responding to the request (main web server process and any additional processes such as PHP, a database, or a second image server)
2	Block if tainted by a suspect server	Block if any process communicated with restricted host or CIDR. Policy check query asked the TA2 to return the list of all hosts the processes involved in the request communicated with.
3	Block automated scripts	Block if there was no user interface action on the client. Client side policy, policy check is still a PASS or BLOCK boolean, however we asked the TA2s to also include a CDM record that indicated UI interaction for any PASS response.
4	Block network data uploads	Block if there was no user interface action on the client. Client side policy, policy check is still a PASS or BLOCK boolean, however we asked the TA2s to also include a CDM record that indicated UI interaction for any PASS response.

3.5.4.2.1. Engagement #4 Policy Enforcement Demonstration Scenarios

For Engagement #4, we chose the scenarios/combinations outlined in Table 7. We wanted to demonstrate each policy and also to show how a single policy would work across all the TA1 clients.

Table 7. Policy Enforcement Demos for Engagement #4

Client	Main Server	Secondary Server	Policy
Untracked	TRACE	MARPLE TA1	1 (Originating User)
Untracked	MARPLE TA1	THEIA	2 (Communication)
Untracked	THEIA	FiveDirections	1 (Originating User)
Untracked	FiveDirections	Clearscope	4 (File Origination)
Untracked	Clearscope	FiveDirections	2 (Communication)
Untracked	CADETS	TRACE	4 (File Origination)
TRACE	*Untracked*	*Untracked*	3 (UI Interaction)
MARPLE TA1	*Untracked*	*Untracked*	3 (UI Interaction)
THEIA	*Untracked*	*Untracked*	3 (UI Interaction)
FiveDirections	*Untracked*	*Untracked*	3 (UI Interaction)
Clearscope	*Untracked*	*Untracked*	3 (UI Interaction)
CADETS	*Untracked*	*Untracked*	3 (UI Interaction)

3.5.4.2.2. Policy Enforcement Module Update (Engagement #4)

In order to handle server policies, we had to redesign the Policy Enforcement Module (PEM) for Engagement #4. First, since we used a more sophisticated server application, a simple redirect was problematic, since we would need to rewrite each page to ensure follow up links or page reloads also went through the PEM. Second, we needed the PEM to intercept the response going back to the client and impose a policy check. A redirect approach would have caused the server to generate the page twice, once before the PEM policy check, then the result would be discarded either way, and the page would have been generated again after the client followed the redirect.

The Engagement #4 Policy Enforcement Module was redesigned so it functions as a reverse proxy as depicted in Figure 11. Incoming requests from the client go directly to the PEM via the PEM host/port. The PEM first does a client policy check, if the requested page was not white listed, and there is a client policy defined. If this passes, the PEM sends the request to the server and intercepts the response. When the response is received fully, the PEM does a server policy check, if the requested page was not white listed, and there is a server policy defined. If the server policy check passes (or there was no server policy check), the response is then forwarded back to the client. The client is therefore unaware that the PEM is enforcing policies, and simply treats the entire PEM, T2 Server, and real content server combination as the single server it communicates with.

3.5.4.3. Engagement #5

For Engagement #5, we kept the same structure, policies, server application, and Policy Enforcement Module. As shown in Table 8, we reduced the number of scenarios from 12 to 8, and allowed for more requests per scenario. This let us explore each policy in some more detail. We used the expanded number of requests to simulate regular web site usage to test the false positive rate of the policy enforcement. In addition, we reconfigured the web server to run in FastCGI mode, which is a normal operating mode for web sites of this type. In FastCGI mode, the PHP process is a long lived process that handles the PHP processing separate from the main apache or nginx web server process. This introduced more inter-process communication for the TA2s to track.

Table 8. Policy Enforcement Scenarios for Engagement #5

Client TA1	Web Server TA1	Database TA1	Policy
Clearscope	*Untracked*	*Untracked*	3 (UI Interaction)
TRACE	*Untracked*	*Untracked*	1 (User Origination)
MARPLE	*Untracked*	*Untracked*	1 (User Origination)
THEIA	*Untracked*	*Untracked*	3 (UI Interaction)
Untracked	TRACE	MARPLE TA1	4 (File Origination)
Untracked	FiveDirections	TRACE	2 (Remote Communication)
Untracked	FiveDirections	CADETS	3 (File Origination)
Untracked	CADETS	THEIA	2 (Remote Communication)

3.6 Infrastructure

3.6.1. Goals

The primary goal of the TA3 team in the Transparent Computing project was to support the greatest breadth of TC technologies possible, and this same goal extended to the infrastructure design and deployment. We attempted to provide a fair set of resources to all teams, but in specific cases we granted exceptions to allow for individual performers to try to further their research. Throughout the program, TA3 also tried to provide some amount of performance isolation at varying parts of the program to avoid giving any team a chance at an unfair advantage. To the extent possible, we tried to minimize the burden of experimental artifacts on TA1 and TA2 performers while also enforcing some boundaries intended to help with integration and reproducibility. We also tried to make sure that TA3 both as an integrator and as a service provider in the TC architecture remained a resource for all the other performers to assist with issues rather than act as a bottleneck which slowed things down.

3.6.2. Project Evolution

3.6.2.1. Base Infrastructure

3.6.2.1.1. Description

The general layout of the base infrastructure is shown in Figure 14.The bulk of the initial base infrastructure was comprised of 34 standard 1U servers interconnected through two networks. These servers were equipped with 32 GB of RAM, and in general, at least 5 TB of storage across two disks. The two networks were a 100 Mbps management network and a 1 Gbps experimental network. There were also four high-end chassis-based servers (Dell C6000s), each holding four blade servers, connected to a 10 Gbps experimental network. All experimental switch interconnections ran at 1 Gbps. The C6000 servers were primarily used for TA2 analysis servers, as they were equipped with 192 GB of memory per blade. These servers were also equipped with a hardware RAID, which we thought would be useful to use data replication on in case the TA2 precomputed views required significant time or manual effort to generate. Each RAID was initially configured as a RAID 5, resulting in a single 4 TB disk being presented to the bare metal OS. The disks which were used for these C6000 servers were a non-standard form factor which were designed to be hot-swappable. An inventory of the test range used for engagements 1-4 is provided as APPENDIX E .

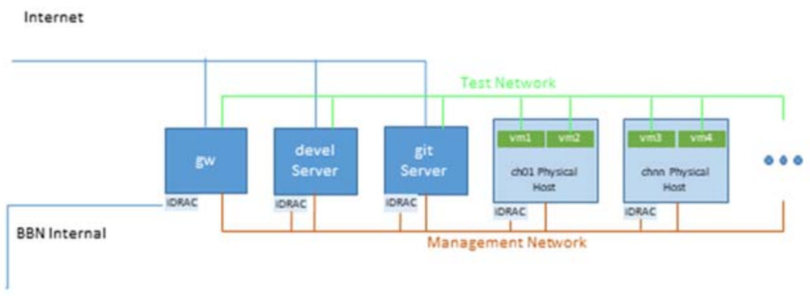


Figure 14. STARC Basic Topology

For Engagement #5 we added an additional 50 hosts to support a total of no less than three engagement endpoints per TA1 performer, additional support for integration and system testing, and to support the needs of several performers who needed to run their systems on ‘bare metal’ rather than in a virtual machine. The Engagement #5 range inventory and assignments is provided as APPENDIX F .

3.6.2.1.2. Separation of Privileges

Initially, the TC network was designed to be fully closed other than a few open ports for services such as GitLab and a file server. Over time, more ports opened up for more services such as monitoring and Jenkins, but the network was still largely accessed through ssh access through a gateway (bastion) host. In addition, the base infrastructure, including the bastion host, had a much more strict privilege model compared to the full access given on the experimental systems, typically with only TA3 members having full administrative privileges to the bare metal systems, infrastructure VMs, and similar systems. The bastion host exclusively supported public key-based authentication, and it was governed by a policy that each user needed their own account. This use of a bastion host coupled with standard separation of privileges led to practical defense in depth. Defense in depth was demonstrated when a security incident occurred due to a violation of policy for each user having their own account. A shared account had been compromised by an external entity, and they used it to access our bastion host. Due to the limited privileges of the compromised account on the bastion host, the external entity was not able to exfiltrate any sensitive information before the infiltration was identified and the compromised account was disabled.

3.6.2.1.3. Separation of Base and Experimental Infrastructure

As mentioned before, the security model of the base infrastructure was different from that of the experimental infrastructure. In fact, the experimental infrastructure was decoupled from the base infrastructure as much as possible not only for security reasons, but also for increased convenience and flexibility. Most performers had their systems run in VMs on top of a KVM hypervisor managed by tools in libvirt. This was the mechanism to decouple the base and experimental infrastructures, allowing us to have different security models for the different realms. Experimenters were given full administrative privileges of these systems to help speed up development and testing efforts, and they were also given Internet access from their VMs (via a firewall providing Network Address Translation (NAT) services) to ease development and deployment. This also led to operational flexibility in that if we had suspicion of a specific machine having hardware problems, migrating the experimental infrastructure to another system was simple. It is worth noting that certain performers at specific points of the project ultimately operated on the bare metal infrastructure. This was ultimately allowed when it was deemed necessary for proper experimentation. For example, TRACE ended up moving to bare metal for demonstrated performance reasons. When running in a VM, they found that they were losing records in their auditing system, and testing on bare metal did not have the same issue. The Clearscope team attempted to run an ingester process connecting to the phones via ADB over USB through VMs, but the hypervisor layer consistently showed stability issues during long running tests, so they ended up moving to managing containers directly on a bare metal system. Finally, in the last engagement, the CADETS team moved to bare metal as they wanted to run the FreeBSD-based hypervisor **bhyve** to collect additional data as part of their efforts, and there was concern about trying to run that on top of a KVM-based infrastructure. Fortunately, none of those teams had hardware issues during the engagements.

3.6.2.1.4. Management of Experimental Infrastructure

Early in the project, the TA3 team evaluated using OpenStack, a full-fledged cloud management solution, in order to manage the experimental infrastructure for the project. We created a basic demo of running OpenStack in our base infrastructure and evaluated it to see if it met our needs. We ultimately decided against using OpenStack, as it seemed complex to maintain, and it solved a number of problems that we did not experience on the smaller scale TC program. For example, for the TC project we felt that we were better off just running a single VM per host for performance reasons, and that seemed like it would be simple to do with standard libvirt tooling. We also had enough hardware to provide a unique host for each TA1, TA2, and the TA3 infrastructure, eliminating the need to share hosts and turn VMs on or off based on what hosts are available. Even when we expanded the scope of the engagements to include three copies of each TA1 host for Engagement #5, we predicted that the cost of buying additional hardware was less than the system administration costs would be at that time for retooling the range, installing, and maintaining OpenStack would have been.

One of the main problems solved by a tool like OpenStack is sharing of hardware resources between shorter lived experiments. Instead, we used libvirt tools, including command line tools to create, destroy, and augment VMs, and tools to open local consoles to the VMs. For systems where bare metal installation was required for the performer, much more manual effort was needed to get the system to a working state, and reinstalls were only performed occasionally.

The trade-off analysis between a home-grown libvirt-based system and OpenStack was performed early in the project and while it was revisited at least once later in the program, a retooling of the range infrastructure was never seriously reconsidered. Over time, new cloud frameworks have arisen and many improvements to OpenStack itself and the surrounding support tools have been made. We also faced internal pressures for changes from security and IT staff who were attempting to implement measures required by the NIST 800-171 compliance program over the last half of the program. Finally, we found that many issues encountered during the engagements were caused by integration problems and differing environments between the TA performer's home development environments and the test environment. To that end, we revisited the 'build or buy' decision one last time, taking advantage of the lessons learned during the engagements and while supporting OpTC testing, and documented our results in a report incorporated in APPENDIX B below.

3.6.2.1.5. Managing Standard Base Systems

To manage most base systems, we installed an LTS variant of Ubuntu and used salt to deploy systems to specific configurations. To manage user accounts, we used OpenLDAP with the ground truth of which accounts to be set up defined in our salt deployment. This worked quite well for the most part, although over the lifetime of the project, salt integration with LDAP broke, which would put our systems into broken states where no one could log in using standard accounts. This led to several instances where we had to pin installed versions of salt to older versions and tracking salt bugs and releases, but we ultimately worked through these issues.

Over the life of the project it was necessary to reinstall the OSes multiple times on the bare metal machines. As the fleet of machines grew and it became clear that reinstalls would be fairly common, a PXE server was ultimately set up to reduce the amount of time and effort needed to install each machine. This saved significant time on some of the batch reinstalls that were performed between engagements.

3.6.2.1.6. Performance Tuning

For the most part, the 1U servers did not cause many issues. The C6000 servers, however, caused some issues during the course of the project. The initial issue reported after Engagement #1 was that disk IO appeared to be slow at times. We explored moving from a virtio qcow-backed disk image with the default KVM caching to a raw disk image with no caching enabled. This did seem to result in generally better performance, so we used this approach for Engagement #2.

During Engagement #3, there were a litany of issues with TA2 systems on the C6000 machines which were all in some way attributable to disk IO. The first issue is that while we had not overcommitted memory on any of the underlying C6000 machines, we had not left enough memory in the base OS to prevent it from going over Linux's default swapping thresholds. Specifically, the VMs running on the bare metal hosts had heavy write workloads that were filling up a cache in the OS, which ultimately resulted in used memory for the VM process on the bare metal OS. The guests may start swapping or freeing up the disk cache space, but in the hypervisor, this memory is not freed from the VM process. At the same time, the hypervisor may start proactively swapping on its own due to the significant memory usage, even though the memory is not fully used. The two fixes to this problem were to reduce the Linux *swappiness* parameter to make this proactive swapping less aggressive, and to reserve more memory at the hypervisor under the assumption that proactive swapping occurs not in terms of absolute remaining memory, but in terms of percentage of remaining memory. The next issue was that under the circumstances listed above, some of the processes in the VMs were dying. The way to fix this was to increase the swap partition size up to **floor(sqrt(MEMORY))** for each VM, where **MEMORY** is the amount of memory allocated to the VM. Both of these changes seemed to help in Engagements #4 and #5.

The final issue is that the TA2s complained that the disk writes were too slow. We discussed multiple ways of trying to address this. One option was to procure SSDs for the C6000s. This option was decided against both because the custom form factor of these disks made that option too expensive, and it also had the downside that it would make TC deployment more expensive. Another option was to change the hardware RAID from a RAID 5 (which has slow writes partially due to replication) to RAID 0 (striped data with no replication). This put the TA2s at risk of losing a single disk corrupting data, but it resulted in better write throughput. Ultimately, there were no data loss issues, but using the RAID 0 did come with significant risk during the engagements as we did have multiple C6000 disks fail during the earlier parts of the program.

3.6.2.1.7. Services

Multiple infrastructure services were set up during the program. From the beginning, we had a GitLab deployment available to all performers to host repositories. This is also where we hosted a lot of our APIs and documentation that was used by the other performers. After Engagement #1, we also set up a monitoring system in order to keep an eye on the operational health of the systems. This was set up using services known as Prometheus and Grafana. External users were able to log into Grafana's web interface and view custom dashboards that we had set up in order to monitor things like publishing rates. As time went on, more metrics were added, such as publishing delay monitoring, NTP drift checking, and standard system level checks (including memory, CPU, network, and disk) for both the base and the experimental infrastructure. Alerting was also used to catch operational events of interest without needing to have someone monitor the dashboards constantly. A chat server for TC users was set up at the beginning of the program, and this service saw significant use during engagements, both for resolving operational issues in real time and also for TA2 performers reporting engagement results in real time. A file server was used to host TC data sets in an easy-to-download archive format for offline use. Finally, a Jenkins server was set up both for the purpose of image generation as well as automated testing of TA1 systems.

3.6.2.1.7.1. Monitoring Infrastructure

We implemented a monitoring system to let us and external participants view the state of the system during testing and the engagements. This is a critical infrastructure component since our engagements ran for up to two weeks with a live red team evaluating real time performance of the TA2 analysis systems. If any components of the infrastructure or TA1 systems suffered crashed or degraded performance, we needed to know as soon as possible so we can apply the appropriate fixes and return the systems to full operation. In addition, the TA2 analysis depended on previous events from the TA1 hosts, so we needed to ensure that the TA1 components were up, running, publishing, and performing as expected continuously, or important events could be lost, which could cause the TA2 analysis to fail.

For Engagement #1, we used the open source **kafkamonitor** tool, which is a web server that could be used to check the current state of the Kafka cluster. We updated and configured this web server to allow us to see current data rate for each TA1 topic, along with Kafka infrastructure state. The current data rate could be used to tell whether a TA1 system was up and publishing, however this was insufficient to go back in time and tell us when a TA1 stopped and how long it was down for. We implemented a tool to check periodically and store historical data rates during the engagement. Additional issues with this simple system were:

1. it relied on someone noticing an error rather than pushing alerts and notifying us when an error was detected,
2. it only showed broker status, not detailed information that could tell us if a broker was under-performing,
3. the TA1 data rate only shows if they're publishing currently, not if they're running behind real time or dropping records, and
4. it failed to gather historical information that could tell us how long ago a detected problem occurred or allow us to predict future problems, requiring us to develop additional one-off tools.

Therefore, we redesigned the STARC monitoring infrastructure as depicted in Figure 15 below. The core of this system is a Prometheus open source monitoring database and scraper tool that periodically scrapes metric data from metric gathering client programs. These metric gatherers can run anywhere, on hosts, VMs, or inside specific applications. The metric gatherers return simple metrics (key, value, timestamp) to the database when they are scraped by Prometheus. The metric data in the database can then be queried by a web browser. In addition, we added a Grafana dashboard server that performs these queries directly and makes the results available through a web interface as a set of tables, graphs, pie charts, and single value panels.

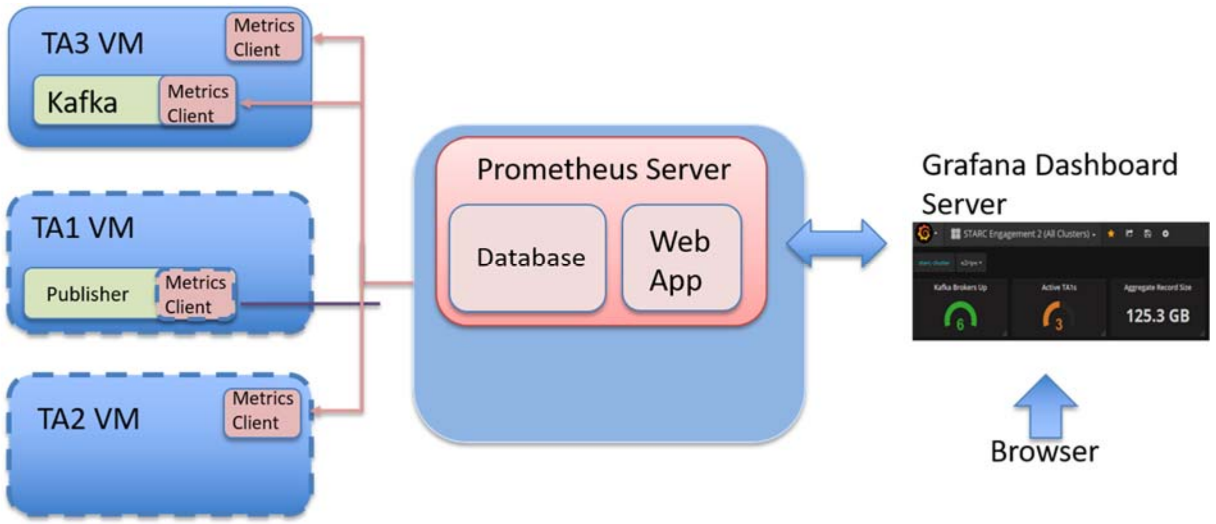
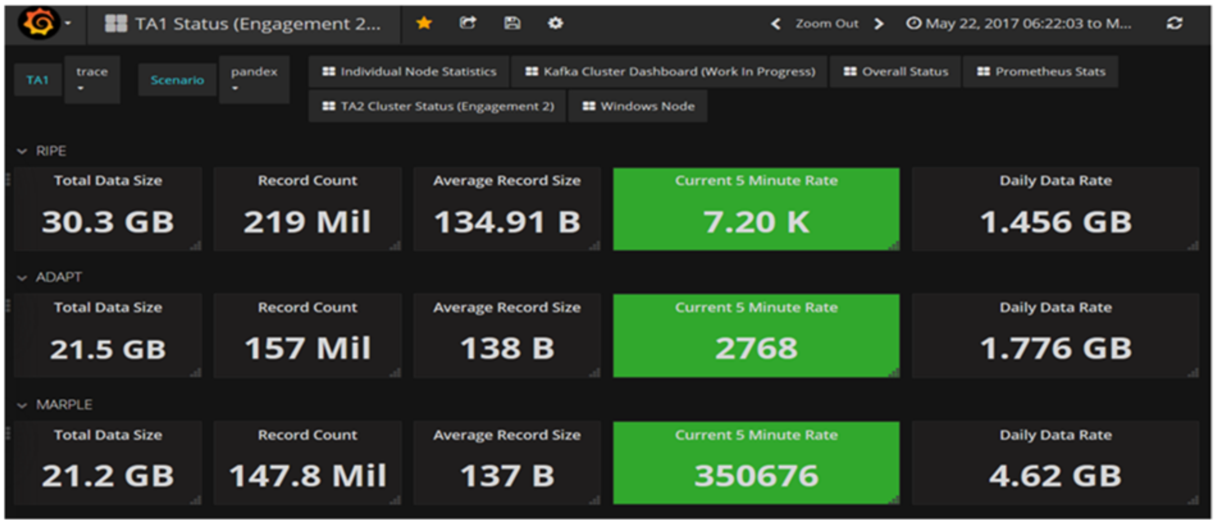


Figure 15. TC Range Monitoring Infrastructure

For Engagement #2, we ran metric clients on the TA3 VMs and inside the Kafka broker processes. These metric gatherers provided detailed monitoring showing disk space, CPU, and memory usage that notified us of potential errors such as a disk filling up slowly, before they became catastrophic failures. The internal Kafka monitoring clients provided information such as which TA1s are currently publishing, how many records they've published, average record size, and total record size, as well as providing a graph of these metrics over time, allowing us to see exactly when a TA1 stopped or slowed down or experienced a burst of records. The dashboards were made accessible to all performers and the government through a web server on our development machine. An example of the Engagement #2 dashboard is shown in Figure 16.


TA1 Status (Engagement 2...
Zoom Out
May 22, 2017 06:22:03 to M...
TA1
trace
Scenario
pandex
Individual Node Statistics
Kafka Cluster Dashboard (Work In Progress)
Overall Status
Prometheus Stats
TA2 Cluster Status (Engagement 2)
Windows Node

	Total Data Size	Record Count	Average Record Size	Current 5 Minute Rate	Daily Data Rate
RIPE	30.3 GB	219 Mil	134.91 B	7.20 K	1.456 GB
ADAPT	21.5 GB	157 Mil	138 B	2768	1.776 GB
MARPLE	21.2 GB	147.8 Mil	137 B	350676	4.62 GB

Figure 16. Engagement #2 Dashboard

For Engagement #3, we added monitoring clients embedded in the TA1 publishing API that let us gather metrics on the TA1 side before records were published to the TA3 infrastructure. This was used for ruling out any infrastructure errors when debugging TA1 performance. In addition we added an optional metric gathering client on the TA1 VMs that gathered host level information such as disk usage, CPU usage, and memory. We made this optional for Engagement #3, since TA2 analysis would need to account for it and treat it as background traffic, though we did end up running the monitoring client on most TA1 systems. We also added alerts that sent an email to the engagement email list whenever a TA1 stopped publishing during the engagement.

We refined the dashboards for Engagement #4, collecting additional data, and adding additional alerts for things like disk space filling up. For Engagement #5, we developed an additional metric gatherer called the Publishing Delay Checker. This gather periodically consumed the latest Event record from each TA1 topic and checked the timestamp of the CDM record with wall clock time. The delta between wall clock and CDM time was the publishing delay metric and showed us whether a TA1 was slowly slipping behind real-time. This publishing delay helped us discover multiple issues with TA1 configuration and design ahead of the engagement, and allowed us to inform TA2 and TA5.1 if a TA1 was experiencing delays significant enough to warrant further investigation and potentially a restart. In different cases we found TA1 configuration errors, TA1 bugs, and TA5.1 background traffic script errors, all of which we could resolve before they caused significant Engagement delays. Finally, for Engagement #5 we enhanced the TA1 host metric gatherer dashboards to allow us to calculate overhead statistics, which will be discussed later in the Results section.

3.6.2.2. Experimental Infrastructure

3.6.2.2.1. Network

As can be seen in Figure 17, all hosts in the infrastructure were attached to two physical networks. One network was for management, and that is where we ran configuration management for bare metal systems. The management network was also closed off from the rest of the Internet. It was connected to the bastion host and the development host, but the subnet was not routable outside the internal network and not attached to any NAT on those hosts.

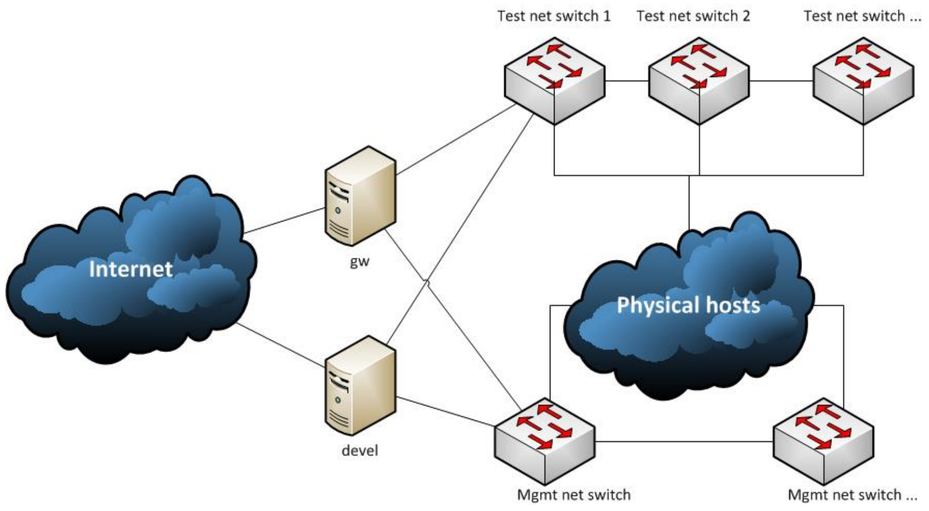


Figure 17. Physical Network Layout

The experimental network was broken into multiple VLANs for different purposes. Those VLANs evolved throughout the different engagements, but there was always at least one experimental management network which was NATed to the real Internet, and at least one experimental data plane VLAN which could reach the simulated Internet for both background traffic and attack traffic generated from a remote subnet. Figure 18 shows an example of the VLANs provisioned for Engagement #1. More information on each engagement can be found in the engagements section.

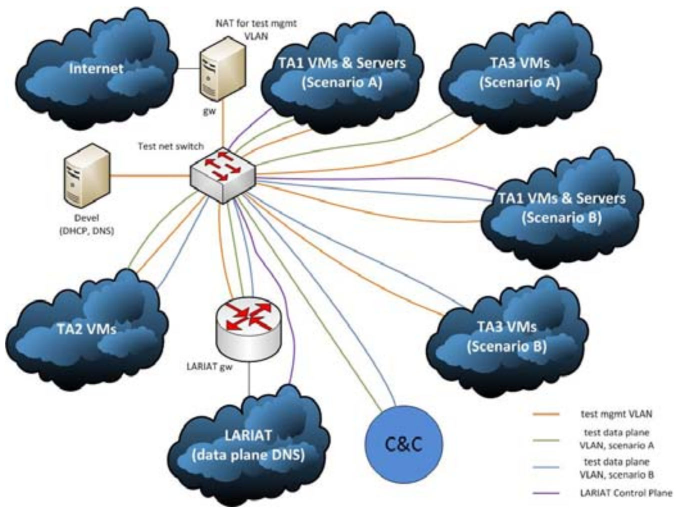


Figure 18. VLANS Configured for Engagement #1

There were specific challenges encountered with integration of performers into this basic model. One of the first challenges was coming up with a way for TA1 systems to be able to reach out to the Internet during deployment and development while also being able to reach the simulated Internet backend during engagements. The original plan put forward by TA3 was to insert a router and have all TA1 systems use that as their default route. Then, when there was a need to switch between the real and simulated Internet, we could update the routing table. While this could have provided the ability to switch individual hosts between the different networks if we installed a large number of routes, it ended up being insufficient and incompatible with how LARIAT expected to work. It was insufficient in that this approach would still leave a host pointing at an incorrect DNS server which would properly work for the real Internet, but would result in incorrect IPs returned for the simulated Internet's hostnames. It was incompatible with LARIAT in that if LARIAT is not the default gateway, then we would have needed to set up a BGP service (as opposed to some type of basic static routes) between the LARIAT gateway and our router, and that sounded more complicated and manpower intensive than what we had hoped for.

Instead, we opted to update the default route and the DNS name server on the end hosts through the Dynamic Host Configuration Protocol (DHCP) server. On a request to cut a given host over, TA3 changed the values in the DHCP server configuration for that host's default route and name server to the selected one, then we restarted the DHCP server. After that, we would reboot the host which would both obtain a new DHCP lease and flush the DNS cache for the host. For Engagement #1, this process was entirely manual, which made having a TA3 person around a blocker for doing this. For Engagement #2, we wanted to automate the process, but we realized that different teams sometimes wanted to use the same host and do testing which might require a specific network. To deal with this, we created a web interface which allowed users to make reservations for specific hosts, and that reservation system was tied into a backend process that would automatically cut a host over using the above described process assuming that there was no conflicting reservation. This seemed to work out well.

The other minor challenge with the experimental networking infrastructure was getting a wireless network set up for Clearscope. The first problem we had was that we used wireless home gateway devices which we had on hand, but those ended up introducing issues such as serving DHCP addresses and providing NAT services. For later engagements, we purchased wireless access points which could specifically be configured as a pass-through device, only converting a wireless network to wired. This generally worked, but in certain operational cases, such as when the access point was rebooted while its upstream switch was down, the access point would fall back to performing DHCP when something connected to it, resulting in very confusing results. Once identified, we were able to watch out for this sort of issue. Finally, the Clearscope team had a requirement for their CI setup that the phone be able to reach the public Internet through its wireless connection. This was a challenge because it required setting up a wireless network which could reach the outside, which had many more policy constraints than the other wireless networks we had set up. We were able to work through this and come up with a reasonable plan to allow for this connectivity without overburdening TA3 operators.

3.6.2.2.2. Hosts

3.6.2.2.2.1. Deployment

For the bulk of the program, we grappled with controlled ways of deploying and managing TA1 systems, as they needed to have specific configurations specified by the TA1 team, TA3, and TA5.1. As the program progressed, we took more principled measures to perform this control, but it was never perfect. The ideal goal was to create a method to automatically provision images and use them as a baseline for deployment into the test range. Following that, the next step was to deploy each image and run idempotent provisioners on deployed instance to perform more evolutionary steps (i.e. updating basic packages and tweaking configuration files). In addition, two disks were deployed for each VM. One was intended to be a persistent data disk, and the other was a transient OS disk available to be overwritten if a new image needed to be deployed. The intent for this procedure was to provide explainability of a system state, reproducibility of specific images over time, and reproducibility of images across multiple nodes. While this approach at a high level is reasonable, it was difficult to realize as most teams were unable to fully automate their deployment procedures. This varied from team to team. The need for this type of automation was pretty apparent throughout the program though, as there was a clear correlation between teams who automated things also being the teams that typically delivered on time and had fewer problems with their systems during the engagement. In the early engagement in particular, we had numerous problems where one team would report that a system had seemingly changed without explanation, and this problem essentially went away once we began using provisioners and limiting access to hosts during engagements.

TA3 attempted this high level approach for all TA3 deployments throughout the program. TA3 images were initially generated using a process that was also used for TC-in-a-box and tools for translating VirtualBox images to KVM images. Starting with Engagement #3, the procedure for generating images was largely done by running the TA3 provisioner. The TA3 provisioner was rolled up in a wrapper around Salt which we call **tc-salt-services**. Under the hood, a lot of the code is a combination of Salt states, pillars, and custom modules. The interface into those is a custom configuration file and a set of shell scripts for automating deployment of Salt infrastructure to all nodes within a cluster defined in the configuration file. The custom configuration file set things up on a node such as which infrastructure services should be run, which hostnames to use, and other parameters which might be placed in service-specific configuration files. In later engagements, some configuration management of TA1 systems was also pulled into the TA3 provisioner.

The TA2 deployment was largely left to the TA2 teams, with some guidelines provided by TA3 such as limits on total memory to use for VMs on a given host. In some of the later engagements, TA3 provided additional recommendations based on debugging IO performance issues described previously.

3.6.2.2.2.2. TA3 Experimental Infrastructure Services

Throughout all engagements, we used a Kafka cluster comprised of 6 nodes with a replication factor of 2 for all topics. This was paired with a Zookeeper cluster of 3 nodes. For some engagements, we decided to deploy multiple clusters (i.e. one per TA2 in Engagement #2), but ultimately we found that a single cluster of 6 Kafka nodes was sufficient. Starting with Engagement #1 and throughout the program, performance testing was run on the deployed Kafka cluster to make sure it could keep up with our anticipated data rates based on spot checks of TA1s and number of systems deployed. We did not have any issues with the Kafka cluster keeping up for Engagements. For the most part, Kafka and Zookeeper were stable and worked well throughout the course of the program, only showing issues during major operational events such as incorrect shutdowns. We created custom workflows for managing both Kafka and Zookeeper through our provisioners and management tools to manage the distributed configuration as well as starting and stopping.

We deployed and managed the TA3 cluster using the **tc-salt-services** provisioner. This tool allowed us to start, stop, and manage the Kafka, Zookeeper, and Prometheus monitoring services using a collection of simple salt commands. This provisioner also allowed us to easily migrate a service to a different host, increase the number of brokers (we experimented with clusters of 1, 3, 4, and 6 brokers), and push configuration changes to all services with a single command.

Starting at Engagement #3, TA3 made a push to include a long term data store into the architecture for TA2s to use. The idea was that a TA2 could choose whether they wanted to pursue a Lambda-style or a Kappa-style architecture, but they should choose one. Both Kappa and Lambda architectures can benefit from a long term data store of some kind. In the case of the Kappa architecture, the long term data store can be used for horizontal scaling of historical data while preserving ordering of records within a stream. Kafka's notion of a partition cannot scale horizontally across multiple nodes, and as such any long term storage requires either vertical scaling of disk on a single system or very careful assignment of data to partitions within a topic, which results in loss of ordering of records between partitions. In the case of the Lambda architecture, the long term data store is necessary for running batch jobs. Ultimately, TA2 teams split between Lambda and Kappa, which was fine. ADAPT and MARPLE chose to use a Kappa-style architecture, which effectively meant no major changes needed for their analysis engines from past engagements, and RIPE chose a Lambda-style architecture, meaning the need to add in a batch processing architecture. We decided to deploy a long term data store for each TA2 to avoid contention between TA2 teams on the data store IO. As seen in Figure 19, the way that different services interacted for each TA2 was dependent on their choice of architecture.

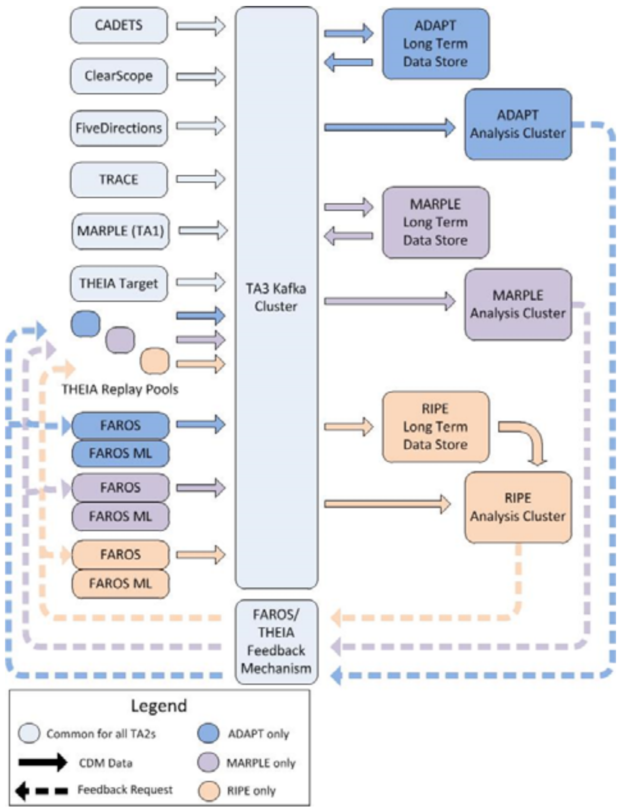


Figure 19. Data Store Architecture for Engagement #3

TA3 opted to use HDFS for the long term data store due to it being compatible with numerous processing engines. An initial design of a basic Hadoop stack was prepared, as shown in Figure 20. The data controller and the batch job controller were replicated across multiple hosts for fail-over, but both services they were co-located on the same host to limit overusing hardware resources. The data storage nodes, which is where the batch jobs also ultimately run in a Hadoop cluster, were set up with 4 nodes, and the data controller was set up to replicate the data within the cluster to avoid data loss. This design ensures that there is no single point of failure in any of the components and that a single node going down would not result in data loss. It projected to have enough storage and parallel processing to be used as a starting point of a Hadoop cluster that a TA2 could use for the TC engagements. The TA3 configuration management suite was updated to both deploy and manage the long term data store systems and services in this configuration for all three TA2s, and basic functionality testing was performed to ensure that data could be read and written, and jobs could be run across the clusters.

Ultimately, the long term data stores were not used in Engagement #3. The four reasons were as follows:

1. TA3 had not completed robust Kafka/HDFS integration before the TA1 system integration frenzy began

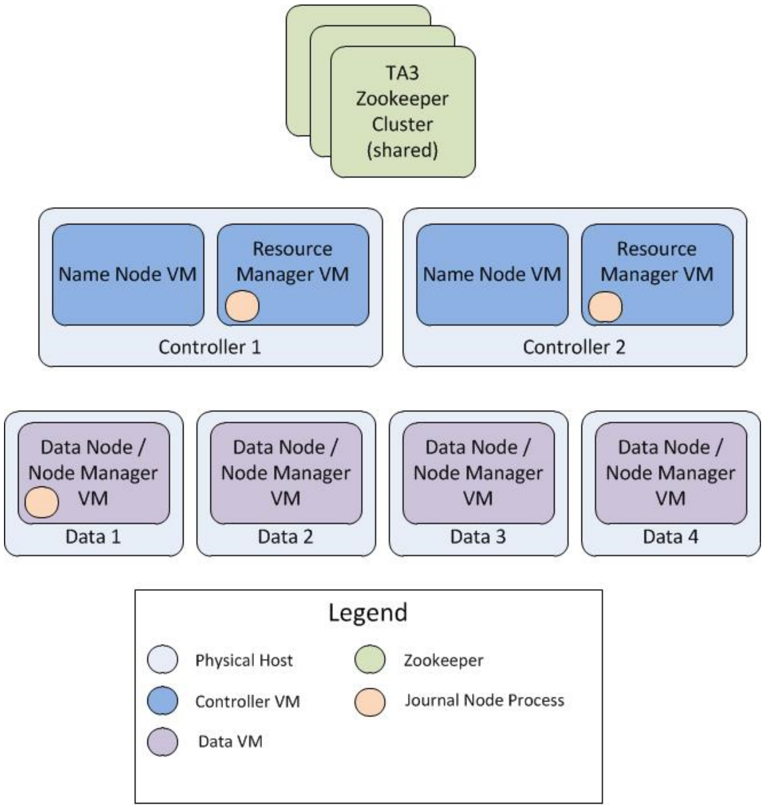


Figure 20. Basic Hadoop Design for Engagement #3

2. TA2s using the Kappa architecture, namely MARPLE and ADAPT, expressed concern that the engagement would realistically be the first time we would have a chance to test replay from a long term data store, and they were more concerned with other problems that needed to be addressed
3. RIPE, the lone TA2 team using the Lambda architecture, decided that they would not have time before Engagement #3 to build up any batch jobs or add other frameworks
4. The volume of data generated in the engagements was not enough to fill up a Kafka cluster since it only had a two-week duration, so data was fully retained in Kafka

At that point, we kept the data stores up, but we chose to run a small artificial test between Engagements 3 and 4 for verification leading into Engagement #4. This never happened either, due to continued high effort on TA1 system integration, the restructuring of how Engagement #4 was run (focus on depth of TA1 technologies) and as more emphasis was put on the policy demo, OpTC efforts, and TC self-monitoring. Ultimately, the RIPE team was the most likely team to need a long term data store, but they did not participate in Engagement #5. It is still strongly recommended for any transition efforts of TC technologies to include a long term storage mechanism other than Kafka, or at least to have a warning about data retention policies if Kafka is used as the long term storage mechanism.

3.7 Critical Factors Pyramid

One of the requirements laid out in the TC BAA was a methodology "…for evaluating and selecting among different architectural design choices…" as well as "…architectural thinking that will allow for adaptation to new findings." In order to address this, BBN developed the Critical Factors Pyramid in order to provide a principled, scientific methodology.

As shown in Figure 21, at the apex of the pyramid was the goal of TC program: detect and counter APTs using data provenance tracking and exploitation. Supporting this overall goal were key metrics used to assess the overall value of the TC technologies toward achieving the program goal. Many of these key metrics were drawn directly from the program metrics in the BAA (e.g., accuracy, speed of detection, and completeness). Others stemmed from basic considerations of overall enterprise performance (such as compute and network overhead). Enforcement granularity was also key to enterprise performance; overly restrictive or coarse policies might block undesired behaviors at the expense of the critical workflows that support the mission.

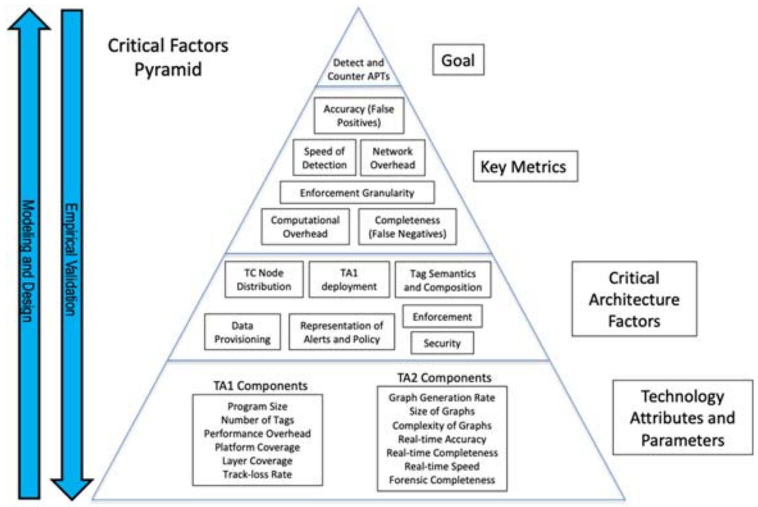


Figure 21. The Critical Factors Pyramid

The key metrics were informed by a number of critical architecture factors that BBN intended to explore to make design choices over the course of the program. Architectural decisions would be evaluated based on the impacts to the key metrics. Following are examples of the critical design issues the STARC effort planned to consider:

- **TC node distribution and integration with enterprise platforms:** The TA2 and STARC infrastructure software could be deployed to dedicated computing nodes, could share computing nodes with enterprise applications and TA1 data collectors, or some combination of the two. Our analysis was deigned to inform this decision, taking into account the performance properties of the TC technologies and practicalities such as that SCADA and mobile platforms might only host TA1 nodes.
- **TA1 composition and deployment:** Given a collection of TA1 tracking technologies (targeted at specific platforms or specific layers on a platform) and an enterprise configuration, we needed to determine how best to deploy TA1 collectors to achieve the most complete situation awareness of sensitive workflows, subject to resource constraints. The analysis should also have been able to inform TA2 performers of coverage boundaries due to partial deployment.
- **Tag semantics and composition:** STARC was required to coordinate the tracking of causally related events by different TA1 technologies cross-platform or cross-layer.

- **Enforcement points:** Given new observables provided by TA1, STARC should have been able to determine where to place enforcement points that can respond to the policy violations detected by TA2.

Finally, the bottom of the pyramid in Figure 21 shows a collection of attributes for TA1 and TA2 performers, including all the TC program metrics by which TA5.1 evaluated these performers. These attributes were key inputs that both constrained STARC architectural decisions and were driven by lessons learned as the program executed.

3.8 Continuous Engagement Process

In order to bring the greatest breadth of technologies together to explore and demonstrate the capabilities of the TC system and to support overall system design and development, BBN developed a continuous engagement plan involving close and constant contact with other performers throughout the program to enable a collaborative design and development effort.

Starting with sidebars at the kickoff meeting and proceeding throughout the program, BBN engaged the TA1 and TA2 performers in discussions to understand their approaches and technical requirements, while working with them to converge on common semantic and data formats usable by all performers. We also ensured that all technology providers are able to interface with the TC architecture and that the overall system was able to leverage all of their developments.

Our process started at the program kickoff, where we requested and obtained sanitized proposals from all TA1 and TA2 performers in order to understand their current technology requirements as well as their plans moving forward. We coupled this by accompanying the DARPA personnel on their initial round of site visits in the fall of 2015, allowing us to directly interact with each team and clarify any issues.

As the program proceeded, BBN used sidebars during Quarterly Program Review (QPR) meetings to obtain up-to-date information from each team at 6 month intervals. Before each engagement, BBN hosted weekly calls for the entire collaboration in order to provide up to date information about test planning and progress as well as review issues. After each of the engagements, BBN held its own internal lessons learned meeting and then led a collaboration-wide call in order to obtain feedback and adjust planning for future engagements. Among the issues revealed in early engagements were problems with communication and recording of decisions, which led to the better use of the BBN TC GitLab Wiki capabilities as well as the use of Instant Messaging (IM) capabilities such as Jabber, to allow real-time communications with the archiving of information.

3.9 TC Range Security

TC system security was envisioned to occur in two stages. In the first, BBN would use standard mechanisms for authentication and data confidentiality (including forward security) to form an initial security base. However, since an APT could obtain full control of a target device and subvert local TC instantiations, further measures would be needed to protect the TC system.

A natural extension of TC capabilities would be to use introspection, wherein TC components monitor each other for misbehavior. In our proposal, we hypothesized that the activities of TC components could be tagged and tracked by TA1s and formulated into policies on which TA2s would be able to alert. In order to do so, one or more TA1s would be redundantly deployed to monitor other TC components, sending the resulting data to multiple TA2s for analysis; this "fault-tolerant" approach would enable detection of compromises in either TA1s or TA2s.

The first stage of security was enabled starting in Engagement #3, using the capabilities of Kafka (version 0.9 and higher) which itself relies upon SSL (or SASL) to provide strong cryptographic assurances of authentication and confidentiality. Hosts on the BBN test range were configured with centrally generated private/public key pairs and corresponding X.509 certificates. These were then configured into Kafka and used transparently to provide security assurances.

For Engagement #4, BBN started exploring the concepts required for TC introspection. As the TC Kafka brokers ran on Ubuntu servers, BBN investigated the possibility of adapting either the TRACE system from SRI or the CADETS system from BAE to run on the Kafka servers during the engagement. After initial tests, it was determined that CADETS was the most promising candidate and BBN engaged with BAE to adapt and configure their system. However, the CADETS system suffered significant problems during the preparations for Engagement #4 due to a series of major internal changes in their system architecture. As a result, BBN was unable to obtain a working version of CADETS in enough time to deploy it on the production Kafka servers.

For Engagement #5, BBN again tried to load and configure Kafka with CADETS installed. One half of the servers were configured with Kafka and the other half were left in standard configuration to ensure that actual TC data would not be lost even if the CADETS system did not function completely. While the system worked initially, it quickly became obvious that CADETS was overloading the servers under the nominal background traffic load provided by the Red Team. As an example, with one TA1 publishing to the Kafka-CADETS cluster, we got an average data rate of 9.8 Mb/s with high variability, for a single consumer (TA2). On the original cluster (no CADETS), we were achieving 28Mb/s consistently, with low variance. The high variance and lower throughput meant that the bottleneck in the processing pipeline was in the Kafka broker itself, leading us to conclude that the setup would be unusable with more publishers during a live engagement. There was not enough time and manpower available from the CADETS side to help BBN experiment with and change the configuration of the servers, so the CADETS-configured servers were turned off and not used in Engagement #5.

If TC is adapted or transitioned, it would be interesting to see if a light-weight system such as the one from Five Directions would suffice to allow TC to provide introspection capability.

4.0 RESULTS AND DISCUSSION

From the sole viewpoint of TA3, Engagement #5 was a success. There was only one issue related to architecture or infrastructure that affected the results of the experiment on day three of the engagement and that was resolved within an hour. The remainder of the issues were due to TA1 components failing or issues with their configuration.

4.1 Critical Factors Pyramid and Architecture

The Critical Factors Pyramid was used throughout the program, and served as a useful guide for making changes to the architecture and infrastructure, but most of 'design-build' iterations ended up isolated in direct interactions between the TA1 and TA2 performers as opposed to system-wide changes involving changes to the architecture.

We had planned to implement a shared long-term data store for all the TA2 performers based on HDFS, however for the reasons stated in 3.2.1.2.1 above we were not able to get agreement for its use from the other participants and complete the implementation. The primary reason for this was that the volume of data involved was small by current standards and storage technology. As shown in Figure 22, the entire three week engagement – involving 24 endpoints –generated less than 2 TB of CDM data spread over six Kafka servers.

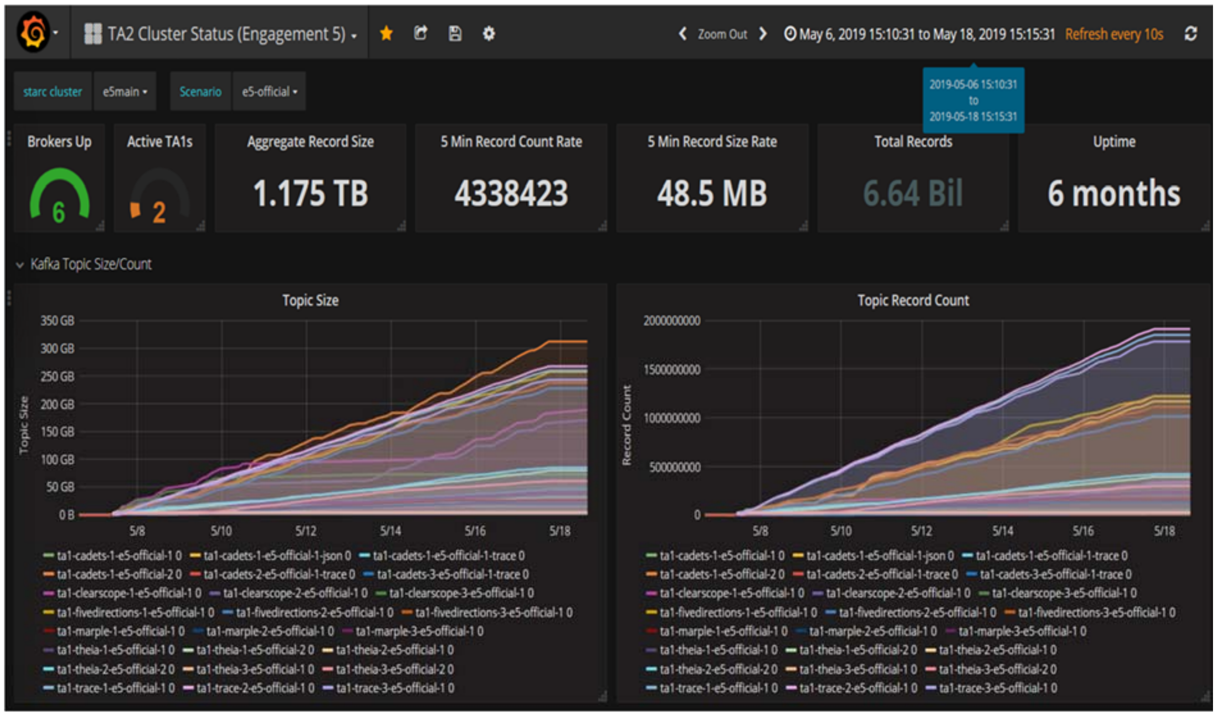

TA2 Cluster Status (Engagement 5)
Zoom Out
May 6, 2019 15:10:31 to May 18, 2019 15:15:31
Refresh every 10s
starc cluster
e5main
Scenario
e5-official
2019-05-06 15:10:31 to 2019-05-18 15:15:31
Brokers Up
6
Active TA1s
2
Aggregate Record Size
1.175 TB
5 Min Record Count Rate
4338423
5 Min Record Size Rate
48.5 MB
Total Records
6.64 Bil
Uptime
6 months
Kafka Topic Size/Count
Topic Size
Topic Record Count
Topic Size
Record Count
350 GB
300 GB
250 GB
200 GB
150 GB
100 GB
50 GB
0 B
2000000000
1500000000
1000000000
500000000
0
5/8
5/10
5/12
5/14
5/16
5/18
ta1-cadets-1-e5-official-1 0
ta1-cadets-1-e5-official-1-json 0
ta1-cadets-1-e5-official-1-trace 0
ta1-cadets-1-e5-official-2 0
ta1-cadets-2-e5-official-1-trace 0
ta1-cadets-3-e5-official-1-trace 0
ta1-clearscope-1-e5-official-1 0
ta1-clearscope-2-e5-official-1 0
ta1-clearscope-3-e5-official-1 0
ta1-fivedirections-1-e5-official-1 0
ta1-fivedirections-2-e5-official-1 0
ta1-fivedirections-3-e5-official-1 0
ta1-marple-1-e5-official-1 0
ta1-marple-2-e5-official-1 0
ta1-marple-3-e5-official-1 0
ta1-theia-1-e5-official-1 0
ta1-theia-1-e5-official-2 0
ta1-theia-2-e5-official-1 0
ta1-theia-2-e5-official-2 0
ta1-theia-3-e5-official-1 0
ta1-theia-3-e5-official-2 0
ta1-trace-1-e5-official-1 0
ta1-trace-2-e5-official-1 0
ta1-trace-3-e5-official-1 0

Figure 22. CDM Data Accumulation Over Engagement #5

This small data set size allowed the program as a whole to sidestep the issue of maintaining historical data – the Kafka servers could hold all the data covering the entire engagement easily and the TA2 performers could merely return to Kafka to obtain raw data. Several TA2 performers incorporated their own databases into their design

While not an issue for the TC program itself, a transition program named Operational Transparent Computing (OpTC) would have exposed this problem. OpTC planned to run one pair of TA1 and TA2 technologies – HostAcuity from Five Directions and RIPE from Boston Fusion – over a much larger set of hosts, with the initial tests to include 200 endpoints. The eventual goal was to deploy the TC technology into an operational enterprise with thousands of endpoints. If one uses the data generation rate of 2.5MB/endpoint/day demonstrated by TC during Engagement #5, retaining data for a six month period for a thousand endpoints would require more than 425TB of raw data storage, far more than can reasonably be managed and retrieved by a Kafka cluster alone.

We had proposed a long-term data store based on HDFS. We hadn't considered using a Time Series Database (TSDB) implementation at the time, but specialized databases such as InfluxDB [12], Prometheus [13] and OpenTSDB [14] offer capabilities and features optimized for time series data storage, querying, retrieval and visualization at the data scales required for such a system. We believe that an implementation of such a system might have provided advantages over the local implementations created individually by several of the TA2 participants.

4.2 Infrastructure

The original proposed infrastructure proved to be quite adequate for the TC program, but did need additional machines to accomplish the goals of Engagement #5, especially with regards to the needs for additional processing power and disk speeds for the TA2 performers, and the needs for some performers to run on 'bare metal' hosts because of difficulties with specialized I/O needs (such as USB communications with mobile phones).

4.2.1. Network Performance

We collected network performance metrics during each engagement with the goal of verifying that the infrastructure was not a bottleneck in the processing pipeline of any of the performers and to identify where enhancements would be necessary to remove them for future engagements.

In order to accomplish this, we ran experiments before each engagement using the intended configuration to see if we could identify any such issues.

We established the expected TA1 data rates and record sizes by examining data sets produced by them, observing that the two rate produces, TRACE and FAROS, produced a maximum of 1.8MB/s and 2.0MB/s of data respectively. Both of these producers used the Java bindings to connect to the Kafka brokers.

4.2.1.1. Engagement #1

The express purpose of Engagement #1 was to "kick the tires" so to speak, and provide a first overall look at all the parts of the TC system. The base goal for the overall system was to be able to generate, tag, and process data from end-to-end successfully. Beyond this, successful detections of APT activities was considered a bonus. For a detailed analysis of actual performance of the TA2 analysis platforms, see the Engagement 1 Final Report, submitted by Kudu Dynamics to the program.

For TA3 specifically, our goals were to:

- Validate the overall data handling architecture and verify the performance of the system and APIs
- Understand the workload and required activities necessary to stand up the run the test range during an engagement

For the first point, see the following subsection with detailed performance test results.

As for the second point, we realized by the end of the test that each technology provider was unique in some way, thus requiring a set of individualized build, install, and test procedures on each platform. The approaches used by each team varied greatly and often utilized vastly different underpinnings, such that lessons on one did not apply to another. At this time, we did not have a central repository with test and build capabilities, and were still receiving source or binary dumps from performers the week before the engagement.

4.2.1.1.1. Pre-Engagement Experiments

We established the expected TA1 data rates and record sizes by examining data sets produced by TA1s on the range and observing the traffic generated by them. TRACE and FAROS, the two highest rate producers, produced maximum rates of 1.8MB/s and 2.0MB/s respectively. Both of these producers used the Java bindings to connect to the Kafka brokers.

4.2.1.1.1.1. Producer Performance Experiments

In these experiments, the goal was to establish the maximum production rates for the Kafka client under three conditions:

1. *No Serialization:* The publisher performs the minimal amount of local processing possible, just sending the same set of raw bytes repeatedly. The goal here was to establish the maximum data rate of the infrastructure.
2. *Static Records with serialization:* The publisher sends the same set of raw bytes repeatedly, but serializes them with the Avro library just as a TA1 performer would need to. Serialization reduces the data size by 50 to 66 percent.
3. *Random Records with serialization:* In this experiment the publisher is fed a set of randomly generated records, which the publisher serializes with Avro and sends.

Figure 23 shows the maximum data rates achieved by the producers for the expected record sizes (in blue) and for very large record sizes (orange) – much larger than those expected in practice. The result is that the upper limit for producers for records of the sizes expected is shown to be about 18 MB/s.

We then reran the experiments varying record size, focusing collection within the ranges expected for our TA1 clients while monitoring CPU, memory and disk to identify any performance limitations imposed by these resources. The producers in this set of experiments were bound to a three broker Kafka cluster and published data to a single topic.

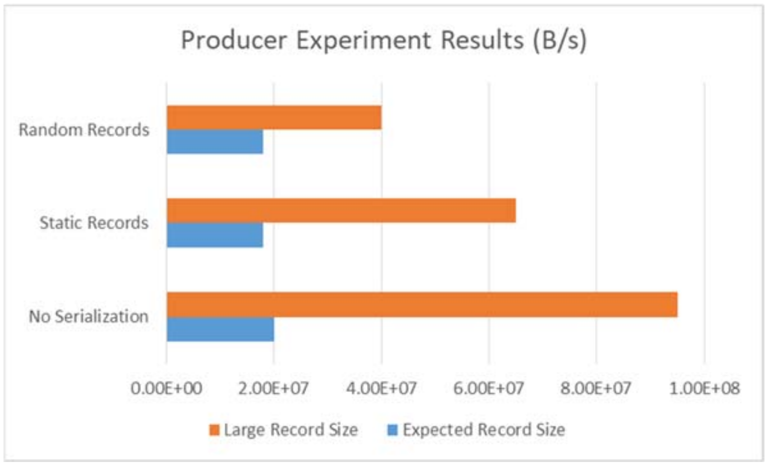


Figure 23. Maximum Producer Data Rates

As can be seen in Figure 24, in the lower range of record sizes and within the ranges expected in production the disk IO rate on the Kafka servers was found to be the limiting factor in performance. As the records grew in size, the bottleneck shifted from the server side disk I/O capacity to the CPU on the client side. Within the expected data rate ranges, the overhead of serialization and the Avro library were relatively small, but as record sizes increased well beyond the expected size, the overhead of the Avro library became significant.

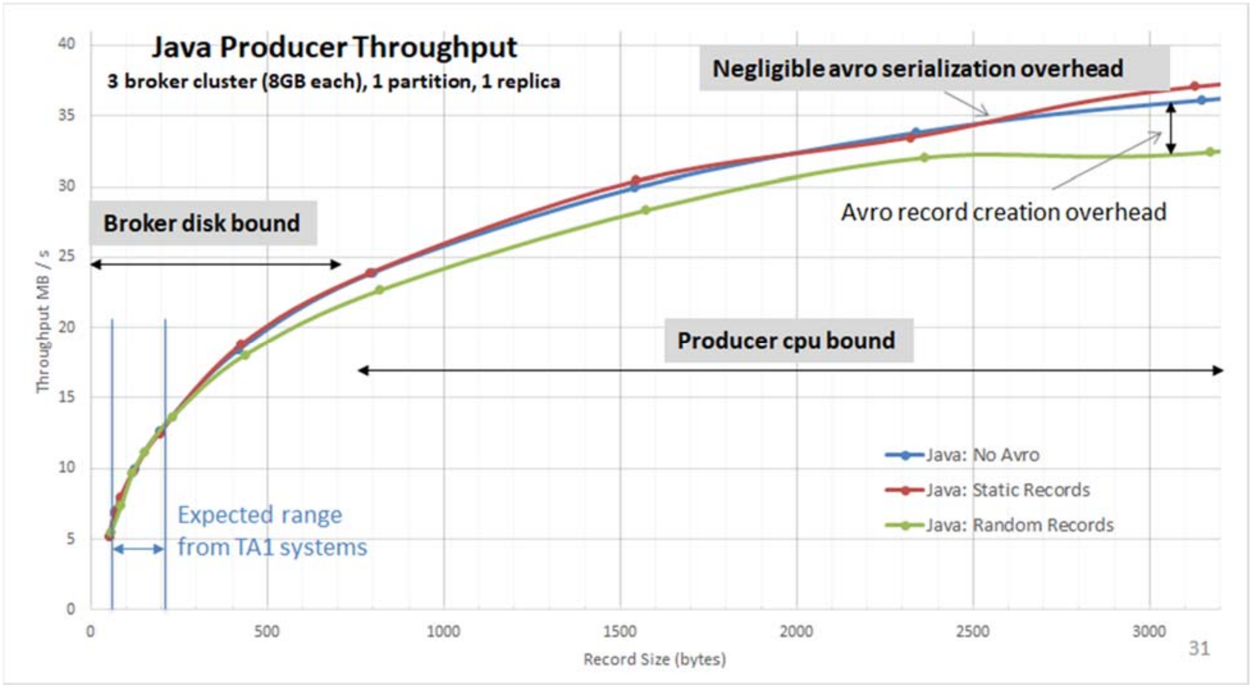


Figure 24. Producer Data Rate vs Record Size, Fixed Flushing

In this initial scenario, we had configured Kafka to flush received data to the disk every 10,000 messages. We decided to retry the experiment while allowing the OS disk I/O system to manage the memory buffer space. The end result was that more data was stored in memory before being written to disk, but that overall throughput was more than doubled to closely match the producer limit (Figure 25). The tradeoff is that the collection system is more vulnerable to a broker failure during an engagement (or in production), but the use of a backup replica mitigates the risk of such an event.

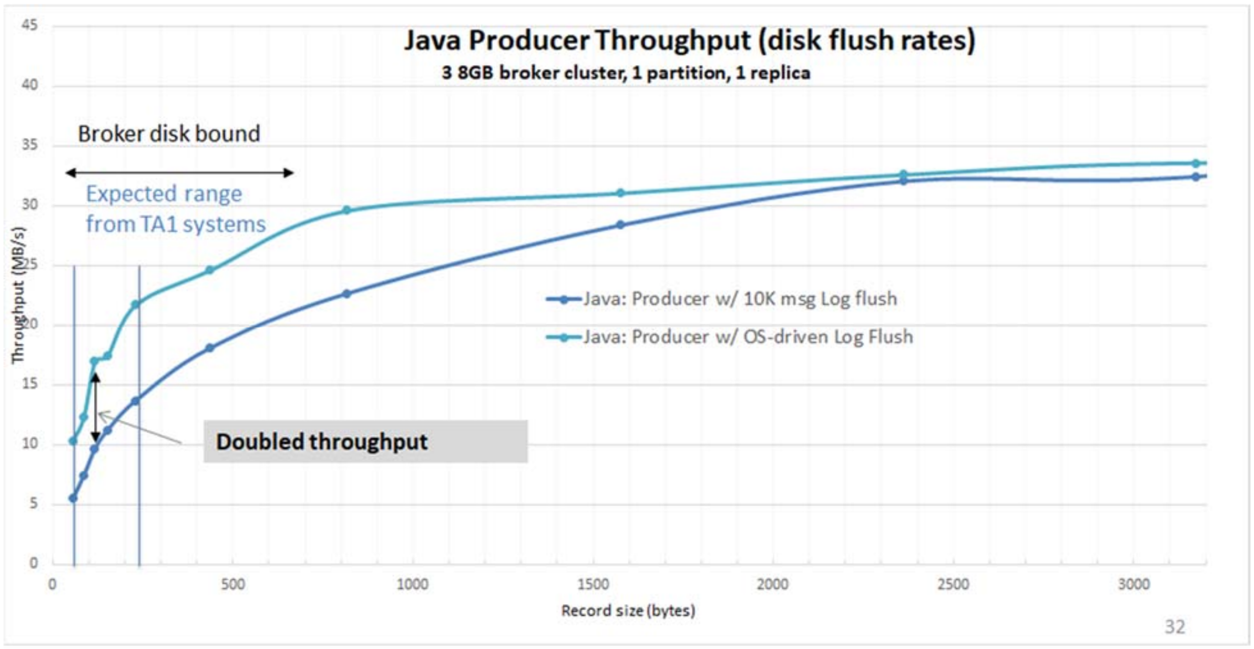


Figure 25. Producer Data Rate vs Record Size, OS-Managed Buffering

4.2.1.1.1.2. Kafka Latency Experiment

The next step in the data processing chain consists of the Kafka brokers themselves. We wanted to establish the data latency from the point at which a producer publishes data to a Kafka broker to the point at which a consumer received the data. In order to perform this experiment, we created the test environment shown in Figure 26.

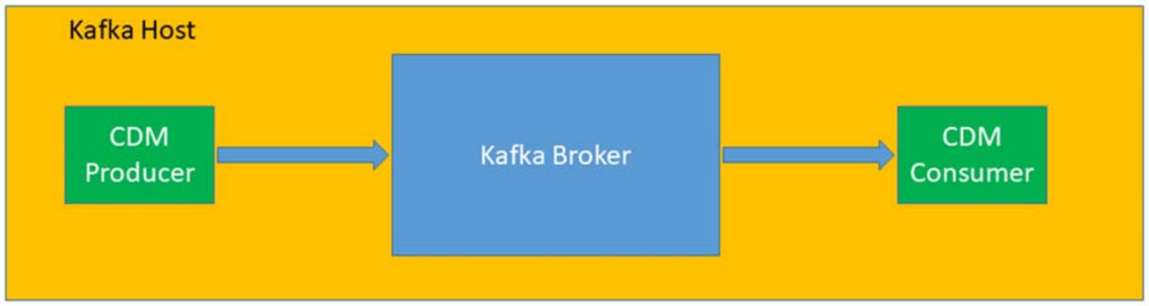


Figure 26. Kafka Latency Experiment Setup

For this experiment, we created a producer and a consumer on the same host as the Broker itself, eliminating network overhead and issues related to dealing with clock skews between hosts. The producer included a timestamp set when each record was created, and the consumer checked that timestamp against the current time when each record was received.

The Kafka infrastructure shown exhibited latencies of 1-2 ms, which were minimal in relation to the overall system.

4.2.1.1.1.3. Consumer Performance Experiments

The final step in the data processing chain is the Kafka to Consumer link. We created a Kafka topic filled with records of the desired size, then consumed them as quickly as possible using two scenarios:

1. *No Deserialization*: Similar to the first test on the producer side, the received records were consumed as quickly as possible and immediately discarded. This test established the infrastructure limits using the Java bindings at 86 MB/s.

With Avro Deserialization: Each consumed record was deserialized into a Java data structure, simulating the process TA2 clients must complete in order to perform any processing on each record.

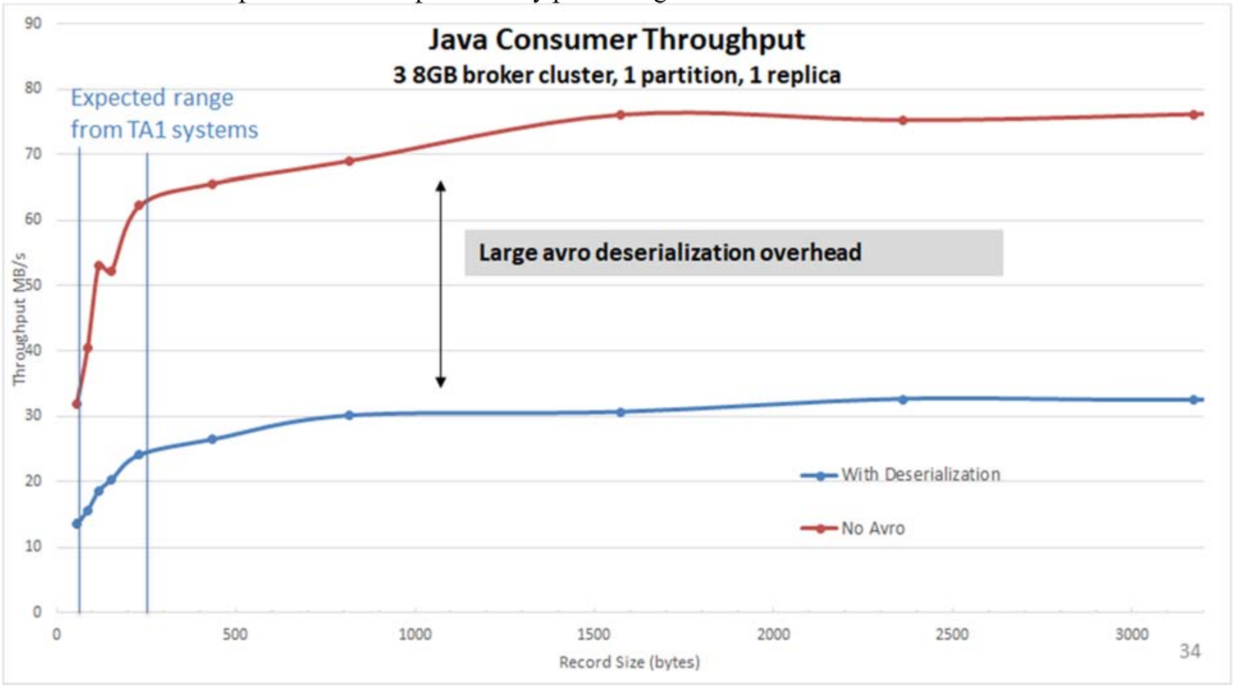


Figure 27 shows the results of these two experiments over a range of record sizes.

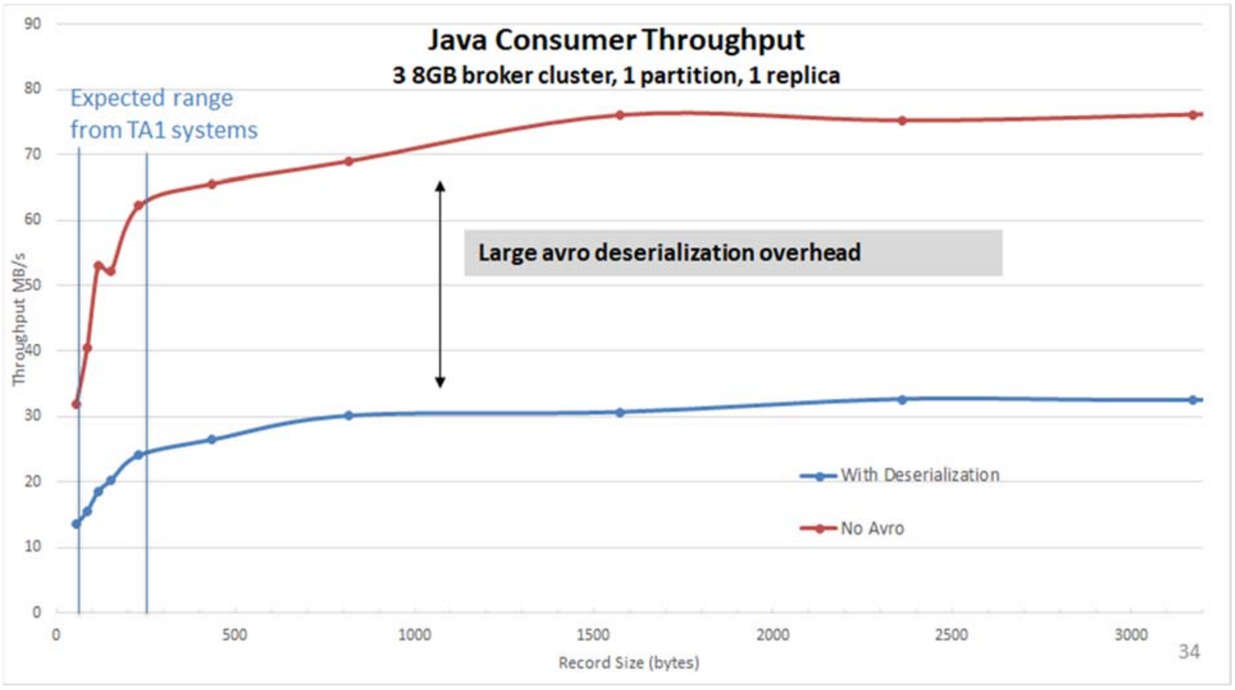


Figure 27. Java Consumer Throughput

As can be seen in the figure, the Avro library imposed a significant performance penalty on throughput, cutting the throughput to a maximum of 32 MB/s for large records, and closer to 25 MB/s in the expected range of data record sizes.

4.2.1.1.1.4. Results for Other Language Bindings

We repeated the producer experiments using the Python 3 and C++ bindings, the results of which are shown in Figure 28. As can be seen, the C++ producer performance is adequate given the maximum expected data rates, but significantly less than those measured with the Java bindings (15MB/s for C++ vs 20MB/s for Java). The Python Kafka bindings performed significantly worse, however, achieving only 1.6MB/s on the producer side (random records with serialization) and 2MB/s on the consumer side (with Avro deserialization). These results were sufficient to support the anticipated data rates for those TA1s that would be using them, but identified these implementations as areas to devote resources to before the next engagement.

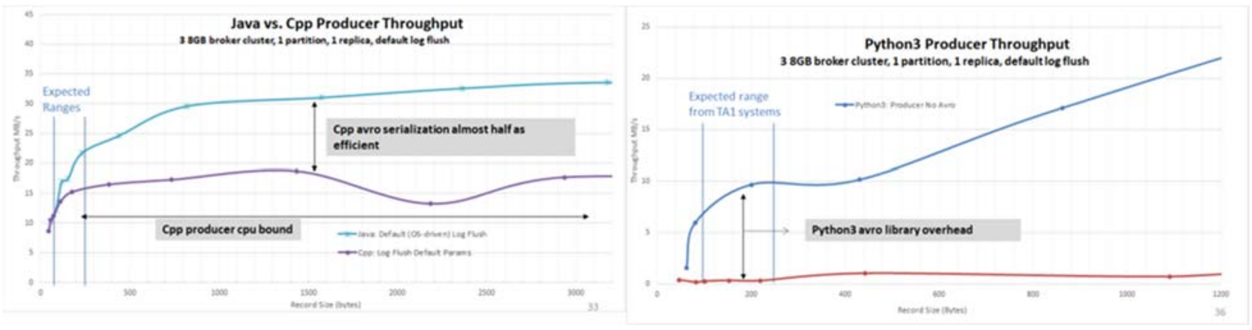


Figure 28. Producer Performance for Engagement #1 C++ and Python Clients

4.2.1.1.1.5. Stability Testing

Finally, we ran system stability tests where we investigated reliability, throughput changes over time, and long duration maintenance issues such as log rotation and disk space usage. Our sample experiment had five producers publishing simultaneously, some with consistent data rates, and some with burst traffic where the data rate spiked up occasionally. Additionally, we had three consumers consuming from each topic continually. After 48 hours of continuous operation we observed no change in average producer or consumer throughput and no reliability issues.

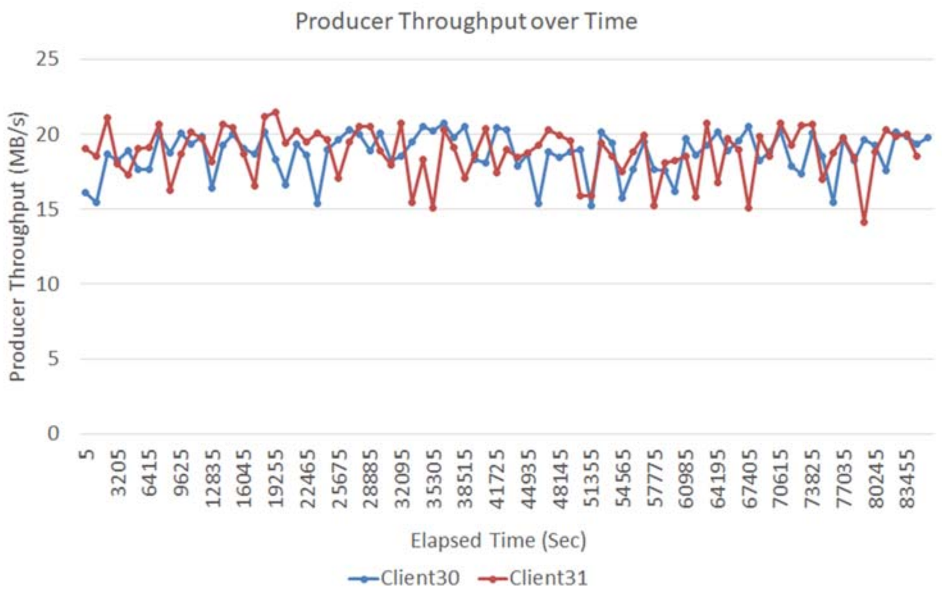


Figure 29. Engagement #1 Stability Test - Producer Side

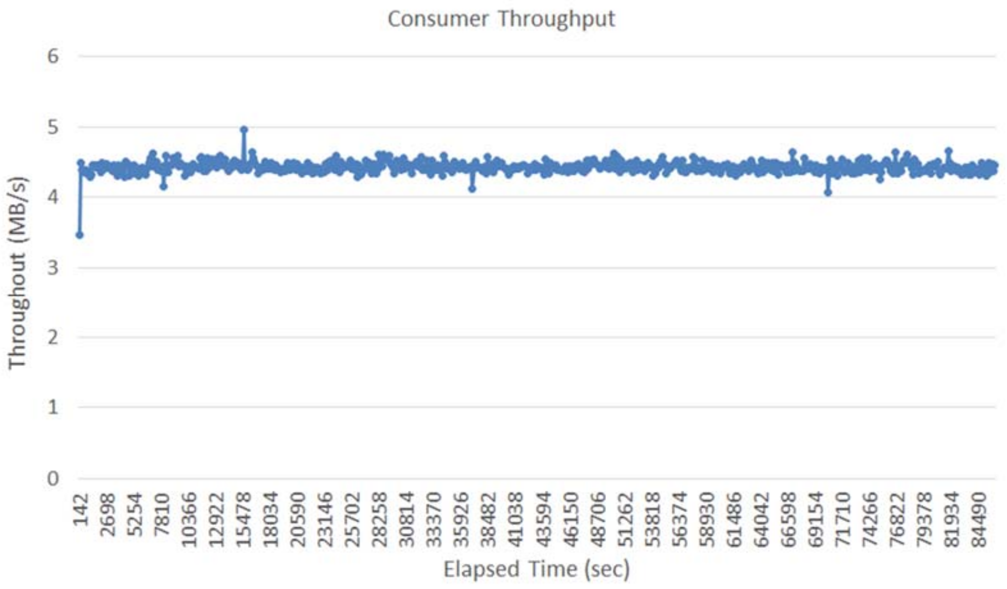


Figure 30. Engagement #1 Stability Test - Consumer Side

The overall result from these experiments was proof of our underlying assumption that the infrastructure was not going to be the bottleneck for Engagement #1. The expected live TA1 throughput rates were orders of magnitude under the theoretical observed limits. We determined expected data rates by examining the sample data sets produced by the TA1s and observed data rates ranging from 1.8 MB/s for TRACE, to 100 Kb/s for THEIA. Since the highest two producer data rates (TRACE at 1.8 MB/s and FAROS at 2.0 MB/s) were both using the Java producer bindings, we determined that our infrastructure would not be the bottleneck for Engagement #1, and likely not for the entire program. The Python bindings limits are much closer to the expected TA1 throughput rate, however, and could become a bottleneck in future engagements as the TA1 technologies produce more data.

4.2.1.1.2. Engagement #1 Results

During the conduct of Engagement #1, we used three scenarios – the Bovia and Pandex scenarios devised by the TA4 participant and a stretch goal scenario devised by TA5.1. The record count and size statistics gathered during each scenario and the totals are shown in Tables 9 through 12.

The stretch goal scenario was created in response to the fact that the attacks in both the Bovia and Pandex scenarios were well known to the TA1 and TA2 performers and the program team wanted to see how their offerings behaved in situations their tools could not be prepared for in advance. The stretch goal test consisted of the TA5.1 team using attacks not disclosed previously to the TA1 and TA2 performers.

Table 9. Engagement #1 Bovia Record Statistics

BOVIA TA1	RECORDS	SIZE (Gb)	Avg Record Size (bytes)
CADETS	24448281	2.75	112.48
CLEARSCOPE	59158240	9.27	156.70
FAROS	264279791	37.55	142.08
AIA	14122189	1.16	82.14
THEIA	50426744	4.93	97.77
TRACE	2340926572	175	74.76
SUM	2753361817	230.66	

Table 10. Engagement #1 Pandex Record Statistics

PANDEX TA1	RECORDS	SIZE (Gb)	Avg Record Size (bytes)
CADETS	23801471	2.68	112.60
CLEARSCOPE	7329057	1.19	162.37
FAROS	248721339	34	136.70
AIA	20496264	1.64	80.01
THEIA	19854503	1.93	97.21
TRACE	1485144886	111	74.74
SUM	1805347520	152.44	

Table 11. Engagement #1 Stretch Period Record Statistics

STRETCH TA1	RECORDS	SIZE (Gb)	Avg Record Size (bytes)
CADETS	2826453	0.33	116.75
CLEARSCOPE	2657023	0.4	150.54
FAROS	184275823	27	146.52
AIA	508893	0.049	96.29
THEIA	9646664	0.99	102.63
TRACE	168341411	12.58	74.73
SUM	368256267	41.349	

Table 12. Engagement #1 Total Record Statistics

TOTAL TA1	RECORDS	SIZE (Gb)	Avg Record Size (bytes)
CADETS	51076205	5.76	112.77
CLEARSCOPE	69144320	10.86	157.06
FAROS	697276953	98.55	141.34
AIA	35127346	2.849	81.10
THEIA	79927911	7.85	98.21
TRACE	3994412869	298.58	74.75
SUM	4926965604	424.449	

4.2.1.2. Engagement #2

For Engagement #2, the overall program goal was to successfully show APT detections under a basic set of conditions for typical attacks, with a stretch goal of detecting additional, harder to find, attacks. For TA3 in particular, our goals were:

- Continue to ensure that system and API performance would keep up with performer needs
- Support real-time detections and higher accuracy by TA2 performers
- Provide support for evolving technology performer needs and requirements

For performance test results, see the next subsection. For real-time and forensic detection results, see the Engagement 2 Final Report submitted by Kudu Dynamics to the program. For the last point, most of the changes among performers were incremental and did not require large adjustments.

Overall, nearly all performers showed a modest to significant improvement in their readiness for Engagement #2, with better documentation and support, more mature software, and easier installation and execution procedures.

4.2.1.2.1. Pre-Engagement Performance Testing and Improvements

Prior to Engagement #2 we were able to develop a simulation based on data captured during Engagement #1. This simulation allowed us to replay the data stream from each TA1 at either its original rate or at a maximum rate, the former allowing us to recreate Engagement #1 in real time, and the latter allowing us to stress test the system. We also added ‘work’ to each producer and consumer by having them write each data field to a log file, intending to create a more realistic workload on each system.

As the performance of the Python implementation measured for Engagement #1 was of concern, we decided to try a new Avro library called **quickavro** [15]. Instead of being implemented in Python, QuickAvro is a Python wrapper library around the official C API and has a primary goal of greatly improving Avro performance. We also updated to a new version of the C Kafka API and made other improvements to our Python implementation. These changes made substantial improvements to the performance of the Python Kafka bindings, as shown in Table 13. These improvements provided a 5x performance improvement over the original Python implementation and only one-half the performance of the Java implementation.

We also attempted to improve the performance of the C++ API, with less successful results. We were able to modify the library to reduce the number of copy operations, which produced a large increase in throughput, however only at record sizes much larger than seen in Engagement #1. The throughput performance improvement for CDM sized records was under 5%. We further analyzed the C++ API and discovered a large number of **mutex** operations. Addressing these would be a larger effort, and the C++ maximum throughput numbers are still an order of magnitude higher than the data rates seen in Engagement #1, so we left the C++ API without any further modification for Engagement #2.

Table 13. Java vs Python 3 Producer Performance, Engagement #2

Language	Experiment	Throughput (MB/s)
Java	No Avro	70.34
Java	Minimal Work	34.23
Java	Verbose Output	8.31
Python 3	No Avro	51.88
Python 3	Minimal Work	15.
Python 3	Verbose Output	2.66

4.2.1.2.2. Results

We collected metrics for both the Bovia and Pandex scenarios and analyzed the results, breaking them out for each TA1-TA2 pair. The results are shown in Tables 14 and 15.

Table 14. Engagement #2 Bovia Scenario Data Counts and Record Sizes

BOVIA	RIPE			ADAPT			MARPLE		
	Count (Mil)	Size (MB)	Avg (Bytes)	Count (Mil)	Size (MB)	Avg (Bytes)	Count (Mil)	Size (MB)	Avg (Bytes)
CADETS	1.80	363	201	2.23	464	202	1.90	382	201
CLEARSCOPE	16.40	8520	520	19.30	9710	504	16.60	8540	515
AIA	1.58	253	200	1.88	294	193	1.58	254	215
THEIA	28.70	4320	152	15.30	2147	127	13.03	2052	152
TRACE	5.68	698	123	5.79	730	126	15.60	2019	129
SUM	54.17	14154		44.50	13345		48.71	13247	

Table 15. Engagement #2 Pandex Scenario Data Counts and Record Sizes

PANDEX	RIPE			ADAPT			MARPLE		
	Count (Mil)	Size (MB)	Avg (Bytes)	Count (Mil)	Size (MB)	Avg (Bytes)	Count (Mil)	Size (MB)	Avg (Bytes)
CADETS	6.39	1302	204	8.98	1832	204	6.51	1328	204
CLEARSCOPE	60.30	30600	507	60.30	30700	509	57.20	29000	507
FAROS	17.70	2666	150	16.90	2540	150	15.90	1395	151
AIA	20.60	3080	149	24.00	3760	156	18.20	2709	149
THEIA	7.52	1141	150	15.70	2273	139	10.37	1624	149
TRACE	220.00	30300	135	157.00	21500	138	166.00	23600	137
SUM	332.51	69089		282.88	62605		274.18	59656	

Data rate statistics were also of great interest, where we measured both the average rates and maximum rates for each TA1. These are shown in Table 16.

Table 16. Data Rate Statistics for Engagement #2

Data Rate Variation	RIPE		ADAPT		MARPLE	
	Avg Rate (k)	Max Rate (k)	Avg Rate (k)	Max Rate (k)	Avg Rate (k)	Max Rate (k)
CADETS	2.8	74.0	4.4	77.0	3.0	82.6
CLEARSCOPE	25.7	39.8	25.8	57.1	26.3	37.2
FAROS	509.0	998.0	457.0	1035.0	471.0	1052.0
AIA	7.8	110.2	4.8	137.6	6.9	107.0
THEIA	14.4	83.8	13.5	78.0	2.2	77.1
TRACE	73.0	1076.0	20.0	337.0	83.0	1076.0

As can be seen in the table, the highest peak data rates achieved were on the order of 1MB/s and the average rates fell well under those values. CADETS, FIVE DIRECTIONS, TRACE and THEIA were very sensitive to background traffic, presenting large spikes in data rates when specific activity occurred on the host. CLEARSCOPE and FAROS, on the other hand, emitted a much more consistent rate of data over the course of the engagement. The peak rates were of particular interest for the TA2 and TA3 performers when planning and configuring the infrastructure and the TA2 systems for future engagements.

4.2.1.2.3. Data Volume Analysis

Based on the results of Engagement #2, we produced a preliminary prediction of the data storage and transmission needs of a TC system working in an enterprise scale environment. This was done to support transition partners with their planning for any eventual deployments and also to inform future system engineering efforts based on TC technologies.

4.2.1.2.3.1. Data Description

Data was taken by selecting a graph of the last two days of the nine-day Pandex data collection period from the monitoring server. The graphs allow downloading in a csv format, which returns a set of data points for each graphed line. In our case, this level of granularity in the graph returned a data point for every four minutes. We extracted data for the total number of records produced within a topic and the total size of a topic. In addition, we extracted the data from the graph of the averaged publishing rates over five minutes only to serve as a sanity check on the calculated values.

Data was fetched from Grafana server as csv files containing the time series data used to generate existing graphs. The first line of the file was then removed for better compatibility with existing R csv import functions. For the purposes of this analysis, we chose to use the MARPLE data set.

To calculate publishing rates, we took the deltas between sequential elements in the topic record count time series data and divided that by four (the granularity of our data points in minutes). This gives us an estimate of the number of records per minute. We also extracted the deltas between sequential elements in the topic size time series data. Finally, we divided the size deltas by the record deltas for each time step, and this gave us an average record size for each time step.

Unfortunately, the FAROS data could not be included in the data set. For these calculations, we rely on the incoming publishing rates, but for FAROS those values did not correspond to real time events on the system. The best we could do for that data was look at the total growth of the topic and note that the data collection ran for around 75 minutes, which only gave us a single averaged data point.

4.2.1.2.3.2. Modeling Methodology

To create projections of what a TA1 might publish, we assumed that the data generated during the time window we analyzed was representative of a real machine in an enterprise environment. Given that we have a data set for each TA1 (other than FAROS) that includes a set of average record sizes and average publishing rates, we can model a window of time by sampling from those data sets at random with replacement. If the data set can be trusted, then this creates a representative model for each TA1 that is better than a simple average rate for projection. The tighter the distribution of record sizes and publishing rates, the less variance there will be in each projection.

4.2.1.2.3.3. Known Issues with this Approach

- The background activity running on the systems may not be representative of actual workflows we are modeling.
- THEIA data may or may not be representative. While the system was up and running for the duration of the data sampling window, the data generation rates seem to be fairly low. This could be an issue with background activity generation, or it may be related to filters that the THEIA team applied as part of the Pandex scenario.
- The sample size is only over two days due to issues with data retention in our monitoring setup. A larger sample size would be ideal.
- TRACE data seemed to be artificially high for a roughly 14 hour period of time overnight. This was likely due to an issue with the system or the background activity generation. The model for the publishing rate seemed to have a spike in its distribution

due to this operational issue. To account for this, we purged that section of the data. This resulted in a smaller data set, but likely a more realistic set of distributions for publishing rates and record sizes.

4.2.1.2.3.4. Result – Enterprise Model A

In this model, we created an enterprise mostly made of user end systems with 75 users who were active during an eight hour period in the day and mostly idle during other hours. Users all had phones, but we modeled only 10 users at a time using them, with 5 of those users also actively using their workstations at the same time, and 5 users strictly using their phones. 60 of the users are running Windows, and 10 are running Linux. We split the Linux users between THEIA and TRACE. Figure 31 shows the data storage requirements to retain the data over periods from one-half to five years.

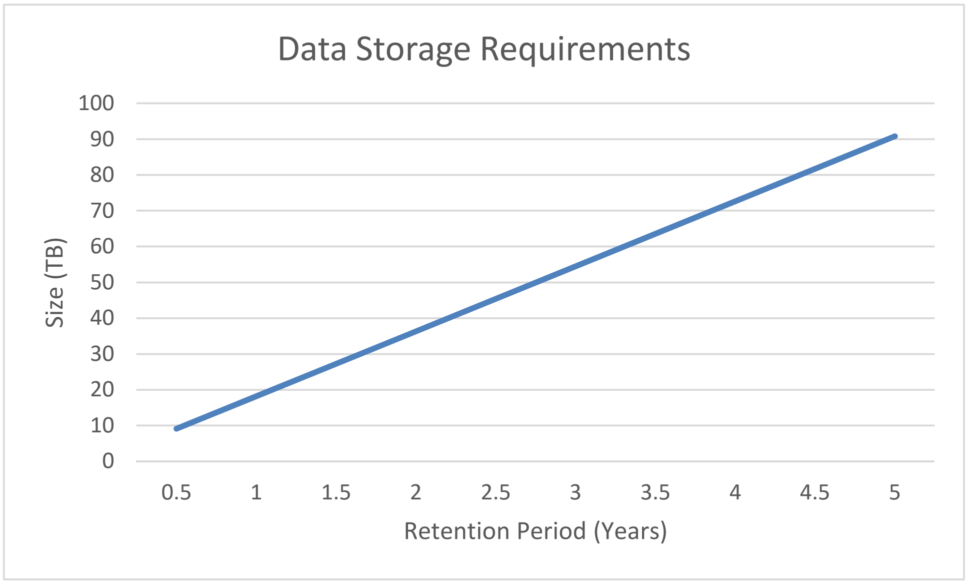


Figure 31. Enterprise Model A Storage Requirements Over Time

4.2.1.2.3.5. Result – Enterprise Model B

For enterprise model B, we imagined a data center centric model with 50 to 500 servers that are constantly active 24 hours per day, 7 days per week. Servers are split evenly between the four performers running server OSes. The data center operators have data retention policies that vary between 1 and 10 years for forensic analysis. Figure 32shows the resulting data storage requirements over varying retention periods for a 250 server data center, and Figure 33 shows the data storage requirements over a five year period for a varying number of server hosts.

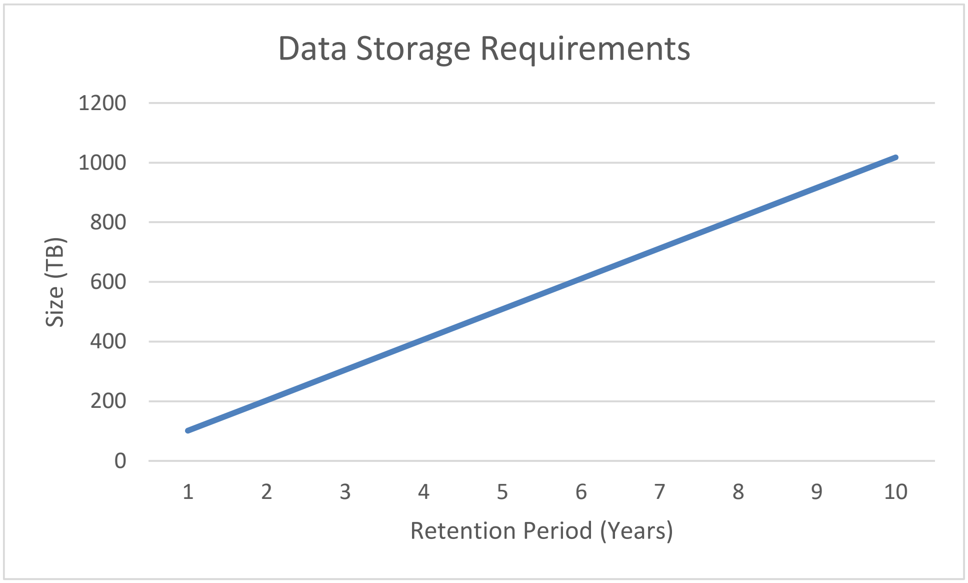


Figure 32. Enterprise Model B Storage Requirements Over Time (250 Servers)

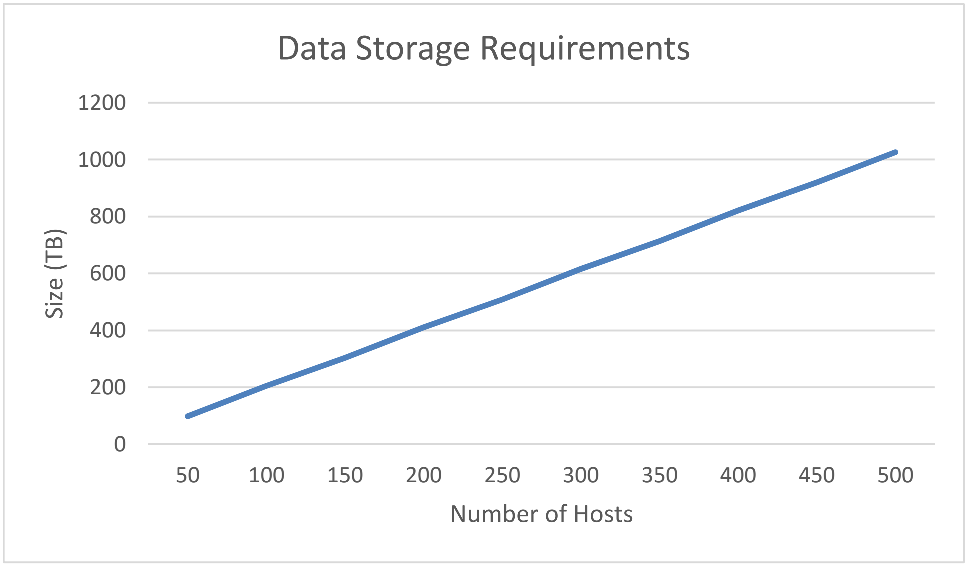


Figure 33. Enterprise Model B Storage Requirements Over Five Years for Varying Numbers of Servers

4.2.1.3. Engagement #3

4.2.1.3.1. Pre-Engagement Performance Testing and Improvements

The largest change to the infrastructure for Engagement #3 was the bump-up of the Kafka version to version 1.0 and the introduction of SSL for the connections to Kafka. Repeating the consumer and consumer tests, we found that the version change and the addition of SSL to the connections had minimal impact to the throughput in the record size ranges we encountered in Engagement #2. This is shown in Tables 17 and 18.

Table 17. Producer Throughput Experiment, SSL vs Plaintext and Kafka 0.11 vs 1.0

Experiment	Size (bytes)	Kafka 0.11 Plaintext (MB/s)	Kafka 1.0 Plaintext (MB/s)	Kafka 0.11 SSL (MB/s)	Kafka 1.0 SSL (MB/s)
Avro, Random records	150	12.65	13.78	14.29	14.29
No Avro	150	54.60	56.35	19.87	22.83
Avro, Random Records	300	14.29	14.28	15.66	15.29
No Avro	300	77.20	61.35	24.20	23.45
Avro, Random Records	5000	21.34	21.12	19.67	21.44
No Avro	5000	80.20	63.33	25.70	23.96

Table 18. Consumer Throughput Experiments, SSL vs Plaintext and Kafka 0.11 vs 1.0

Experiment	Size (bytes)	Kafka 0.11 Plaintext (MB/s)	Kafka 1.0 Plaintext (MB/s)	Kafka 0.11 SSL (MB/s)	Kafka 1.0 SSL (MB/s)
No Avro	150	59.9	66.5	25.6	27.91
Avro, Minimal work	150	27.6	20.7	18.6	19.9
Avro, Busy work	150	4.28	4.21	3.80	3.81
No Avro	300	64.8	71.5	30.9	31.02
Avro, Minimal Work	300	31.9	26.77	23	22.4
Avro, Busy work	300	6.22	6.45	5.69	5.17
No Avro	5000	84.4	85.6	36.3	35.09
Avro, Minimal Work	5000	45.9	34.9	27.2	24.8
Avro, Busy Work	5000	22.15	21.31	16.11	16.35

4.2.1.3.2. TA2 Disk Performance Issues

We encountered an issue during Engagement #3 when TA2 performers reported slowness and periodic freeze-ups occurring on their VMs. The underlying issue was in how the TA2 applications interacted with the hard disk. Examining the disk and buffer metrics reported by the performance data collection system, we determined that their workload saturated the disks with writes. The VMs themselves had too much memory allocated, allowing them to fill up the disk write cache and causing this memory to never be freed. Eventually the hypervisor started swapping as it competed for saturated IO, leading to these periodic freezes. This condition wasn't detected during integration testing, since it only occurred with larger, long running workloads.

We were able to reconfigure the VMs and tune kernel parameters in order to eliminate this issue prior to Engagement #4.

4.2.1.3.3. Results

Table 19 shows the TA1 (producer) metrics collected during Engagement #3.

Table 19. Producer Data Metrics for Engagement #3

TA1	Records Produced	Data Size (GB)	Avg Record Size (bytes)
CADETS	44,404,339	11.2	252
CLEARSCOPE	488,027,243	121.0	249
FAROS	58,944,195	6.4	110
AIA	310,748,711	64.1	206
THEIA	118,102,141	17.5	148
TRACE	1,049,370,350	146.0	139

We had made significant improvements to the Prometheus-based monitoring infrastructure, which required us to perform some tuning on the Prometheus server on the second day of testing. We did this maintenance outside of the 'daily normal working hours' used by the test, and no data was lost as a result of the tuning.

We also had to restart several of the TA1 systems due to various issues, none initially related to the infrastructure. The problems were either outright crashes or that system performance would slow to a crawl. In one instance the restart process was not assiduously followed by the TA3 team, and this caused subsequent issues with data collection that required an additional restart. Every TA1 performer experienced at least one issue requiring a restart or dropping functionality from the test plan.

In some cases, we allowed a TA1 to patch their systems under test. In general, the patches were not effective and/or the changes made introduced additional instability to their systems.

4.2.1.4. Engagement #4

There were a few major changes for Engagement #4 for the purposes of the program:

- Two identical copies of each TA1 were instantiated within the range, allowing the Kudu red team to pivot from host to host and test whether performers would be able to track and handle this case
- Instead of testing being spread out among all hosts for two weeks, each TA1 was heavily tested for a single day or two, in order to provide an in-depth look at capabilities
- The policy enforcement demonstration was expanded from the simple scenario in Engagement #3 to a full-scale set of tests and scenarios

The policy enforcement demo is discussed in detail in section 4.3.2.

For TA3, the major goals were:

- Install and populate a central build, test, and deployment system as part of TC test range
- Add in support for public/private key pairs and digital signatures, allowing TC hosts to authenticate themselves to the system
- Perform an initial test of running Kafka servers on a TA1 platform, allowing TC to perform self-defense and introspection

- Support two copies of each TA1 system on test range, allowing pivoting (establishing malware on one host, then using that to attack additional hosts within the network) across host systems
- Improve monitoring capabilities of the range

Starting in Engagement #4, with the concurrence of the DARPA PM, BBN stood up a Jenkins build and test system to accompany its existing Git repository and required that all performers use this continuous integration system to submit and prepare code for the engagement. This was prompted by multiple regressions in several of the technology performers in Engagement #3 preparation and execution. An advance deadline for submissions was established before the risk reduction and final evaluation phases to ensure that final versions of software would be tested and installed with enough time to insure a full test cycle.

BBN also upgraded the version of Kafka to enable native support of host authentication. BBN instantiated a simple public-key infrastructure with a key/certificate generator to provide unique identities to each participating TC host, providing security against all outsider and limited insider attacks. This system worked reasonably well except for one TA1 due to integration issues with the Kafka client.

For a detailed analysis of running Kafka on a TA1 platform, see the next subsection. Supporting two copies of each TA1 was an increased support burden that caused more difficulties than we expected. While planning, we had hoped that the hosts would be able to use nearly identical configuration and setup processes, but inevitably minor differences would crop up and cause issues. BBN continued to press for the use of central build and integration systems to alleviate future problems.

Monitoring capabilities on the test range continued to expand and improve. It was generally remarked that the monitoring support was greatly appreciated by all participants in the post engagement lessons learned call.

4.2.1.4.1. Kafka on CADETS

For Engagement #4, we attempted to run the TA3 infrastructure hosts on a TA1 CADETS enabled system. This would allow the TC infrastructure to also be protected by the TC technology, and would expand the attack surface presented to TA5.1 to include the TC infrastructure itself. We designed a hybrid cluster wherein three Kafka brokers would run on the original Ubuntu VMs (without CADETS), and an additional three brokers would run on CADETS enabled VMs. The three CADETS-protected brokers would house the leader replica for each TA1 data stream, with follower replicas housed on the Ubuntu VMs. Finally, the internal CADETS TA1 CDM data generated by the Kafka broker CADETS VMs would be published to the Ubuntu VMs only. This setup ensures that one replica of each data topic is present on an Ubuntu VM Kafka broker, so if the CADETS Kafka brokers had a catastrophic failure, no data would be lost. This design is shown in Figure 34.

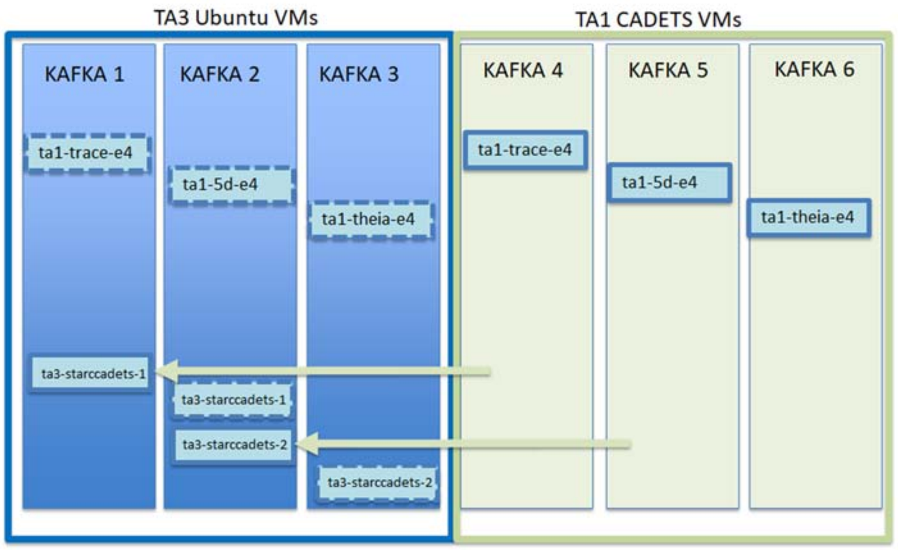


Figure 34. Proposed Kafka Setup for Engagement #4

The CADETS team, however, made major changes to their internal architecture for Engagement #4, and thus were not able to provide a working version of CADETS until very late in the process, too close to the planned start to complete adequate testing. We instead tested a TA3 on CADETS system using a Kafka mirror cluster. We ran six Kafka brokers on Ubuntu VMs without CADETS for the main engagement cluster, and then setup a three broker Kafka mirror cluster on CADETS VMs. We developed and ran a mirror publisher, which consumed some TA1 CDM data topics from the main cluster and published them to the CADETS cluster, and measured the performance and overhead. In addition, we asked some TA2 performers that had time and resources to investigate running their analysis by consuming data from the CADETS-protected cluster.

We found that Kafka Brokers on the CADETS cluster was not able to keep up with engagement data rates and load. The CADETS cluster was stable (in terms of system load and latency) with two topics being published at real time data rates, but any large spikes in traffic rate, or any additional traffic, even including a TA2 consumer, would cause the infrastructure services to start timing out. The consumer data rates became highly variable and throughput dropped significantly. This result shows that the then current version of CADETS was unusable for infrastructure services such as Kafka at live engagement stability and data rates. Services like Kafka receive many requests per second, and perform many disk writes to the Kafka log directory. This traffic generates too much CDM data for the Engagement #4 version of CADETS with a default configuration to process in real-time, leading to delays and instability.

We believe it would be possible to carefully tailor a configuration of CADETS that limits the type of CDM data being produced, but this would require careful analysis of which events to summarize, which to discard, and which to publish. This analysis would be a significant additional workload for the CADETS team.

Error! Reference source not found. below shows the throughput (left) and system load (right) for the Kafka on CADETS test cluster with two mirror publishers and one consumer running. The max consumer data rate recorded was 15 MB/s and that was highly variable (blue line) and decreased over time. In contrast, the data rate for a consumer running on the main Ubuntu Kafka cluster is shown by the red line, and that throughput rate is limited by the CPU on the consumer end, not the broker.

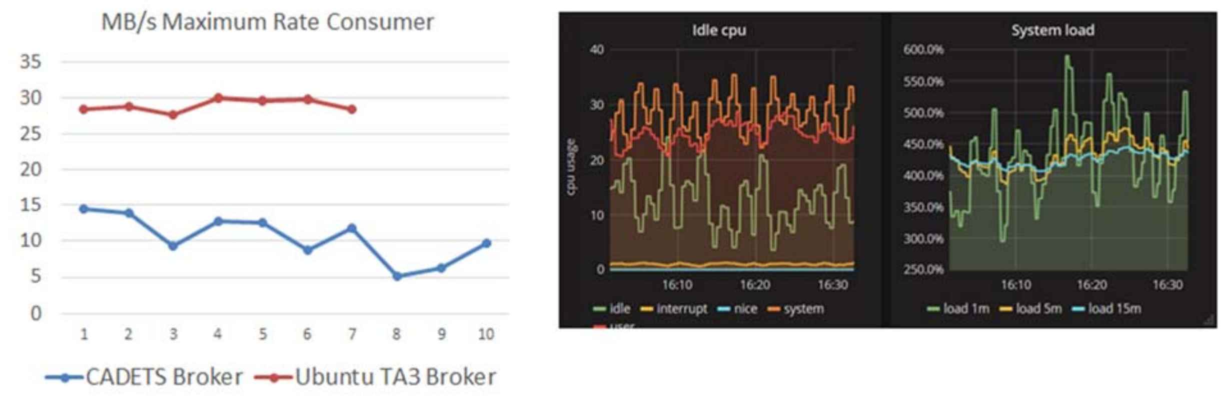


Figure 35. Maximum Data Rates, CADETS vs Normal Kafka Brokers

4.2.1.4.2. Publishing Delays

We used our CDM semantic checkers to uncover numerous errors in the CDM sample data provided by TA1s, and in all cases were able to identify the issue, present it in sufficient detail to the TA1s, and import and test a fixed version. One issue that arose in multiple Engagement #4 TA1 systems that was not a semantic issue was the problem of publishing delays. Our monitoring prior to Engagement #4 showed publishing rates and throughput, but couldn't tell if a TA1 publisher was slowly falling behind real-time. Some TA1s were generating more data than in prior engagements due to increased capabilities, and ran into the condition where the rate of data being collected was higher than the rate at which they could translate the collected data into CDM and publish it. In all cases, the slowdown was due to the TA1 translation software. Determining that this publishing delay was happening required custom investigation over the published data using manual methods.

We developed publishing delay checkers and monitoring software to eliminate this potential source of problems for Engagement #5.

4.2.1.4.3. Results

For Engagement #4, we ran each TA1 separately on its own day with two TA1 instrumented hosts and cross-host traffic going between them. CADETS also uses the TA3 infrastructure to publish internal intermediate data that is not intended for TA2 to analyze. A CADETS correlator process would then consume this internal data, correlate cross-host traffic, and publish the final CDM stream. The numbers provided in the table below are smaller than previous engagements because these represent only a single day of data collection.

Table 20. Engagement #4 TA1 Publishing Volumes (One Day)

TA1	Records Generated (Host A) (Millions)	Records Generated (Host B) (Millions)	Data Size (Host A) (GB)	Data Size (Host B) (GB)	Total Records (Millions)	Total Size (GB)	Averege Record Size (bytes)
CADETS (CDM)	23.80	24.30	7.30	7.40	48.00	14.70	307.00
CADETS (Internal json)	23.60	24.00	0.53	0.51	47.60	1.00	22.00
CADETS (Internal trace)	0.53	0.54	2.80	2.90	1.06	5.70	5,410.00
ClearScope	32.10	21.90	20.10	14.10	54.00	34.20	642.00
AIA	45.20	49.60	9.60	10.60	94.70	20.20	212.00
MARPLE	91.60	105.00	14.40	16.20	196.60	30.60	155.00
THEIA	29.80	246.00	6.40	53.30	275.80	59.70	214.00
TRACE	39.10	48.90	6.20	8.60	88.00	14.80	158.00

Once we had collected this data, we then paired different TA1 systems with each other to produce heterogeneous cross-host traffic. The results of this experiment are shown in Table 21.

Table 21. Engagement #4 Cross-Host Publishing Volumes (Three Day)

Host A TA1	Host B TA1	Records Generated A (Millions)	Records Generated B (Millions)	Data Size A (GB)	Data Size B (GB)
CADETS	THEIA	53.3	12.8	15.8	2.8
AIA	ClearScope	34.6	43.0	7.9	22.5
MARPLE TA1	TRACE	110.8	29.7	16.8	4.6

4.2.1.5. Engagement #5

For Engagement #5, the overall goal of the program was to allow each performer to fully demonstrate their full and unique capabilities and to provide some insight into the operation of a full TC system. In addition, a revised policy enforcement demonstration was implemented to test whether an attacker could circumvent the security policies informed by provenance information. The results of this demonstration are provided in 4.3.3.

For TA3, the specific goals were:

- Expand the test range to three simultaneous copies of each TA1
- Expand monitoring of the test range to collect TC program metrics as well as metrics to support OpTC transition
- Support differing levels of TA2 participation, due to commitments the participants had to support the OpTC program

The test range architecture was essentially the same as it was in Engagement #4, with the addition of additional machines to support the expanded TA1 range. Coordination with TA5.1 to only perform relevant tests for particular TA2 platforms was straightforward. The Git/Jenkins infrastructure continued to be used with improved process results in this engagement.

Kafka statistics were gathered along with Prometheus data to allow analysis of data throughput, delay monitoring, and other statistics. Our results are presented below.

4.2.1.5.1. Publishing Delay Monitoring

We developed a new monitoring feature for Engagement #5 that allowed us to check every TA1's publishing delay in real time. As in Engagement #4, the publishing delay was defined as the time between the most recent CDM event published by the TA1 (included in each TA1 CDM data record), and the current wall clock time from our monitoring client. Therefore, this publishing delay includes the latency for the CDM record as it propagated through the TA3 infrastructure. An increasing publishing delay indicates that the TA1 is falling behind, which may be caused by a misconfiguration, an error in the TA5.1 benign traffic scripts, or a failure in the TA1 VM (likely caused through interaction with the TA1 software). Publishing delays cause TA2 systems to be unable to detect events in real time, since the data they are ingesting can be minutes or hours old.

Using this metric, we were able to detect multiple cases where TA1s (CADETS and THEIA) had configuration errors or needed to be restarted. These issues were caught early and we were able to reconfigure/restart the TA1 systems and get the publishing delay back to real time. Other TA1s, such as AIA, had periods of time where their system ran up to 25 minutes behind, but eventually caught back up without outside intervention. This issue was likely due to increased traffic on the monitored host and is an issue for future investigation by the TA1 team. The publishing delay monitor historical data allows us to check what the publishing delay was for an AIA host at a specific point in time, so when the red team caused an event on the AIA host, we can more accurately determine the earliest time when the TA2 could have received the necessary CDM data in order to detect the host. An example of the publishing delay monitor is shown in the figure below.

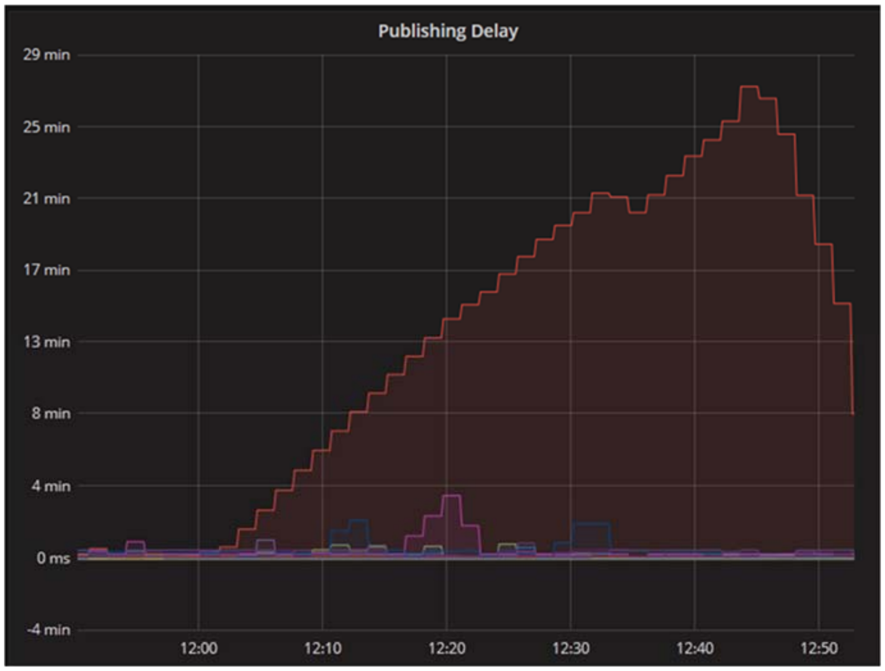


Figure 36. Publishing Delay Example Detail

The following figure shows the publishing delay experienced by all TA1 systems throughout the time period of the entire engagement.

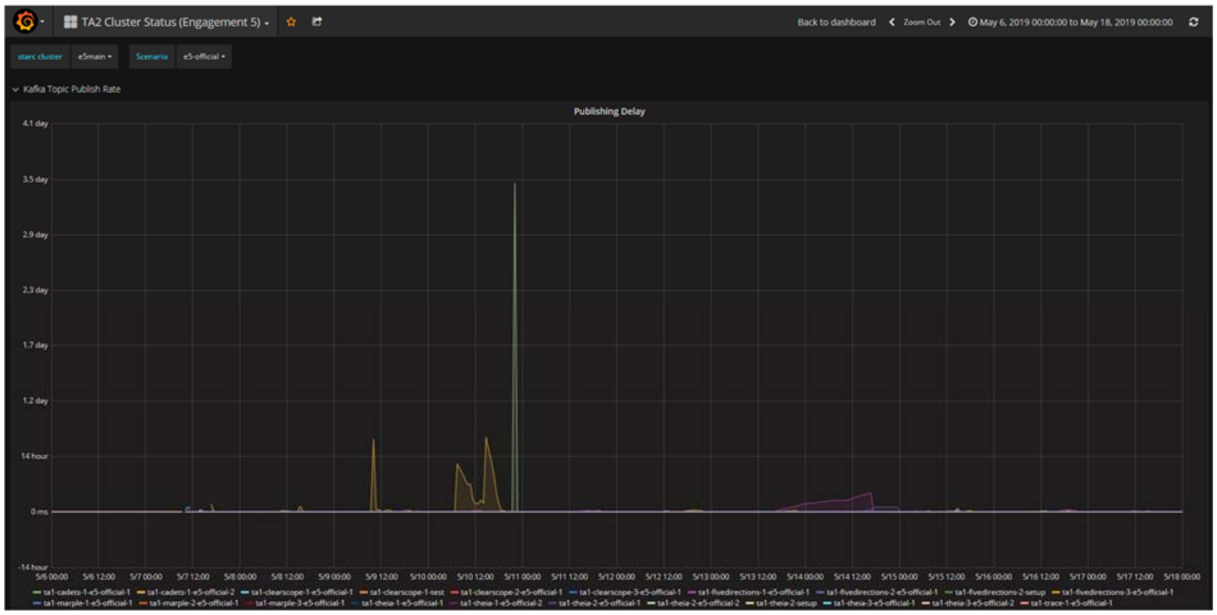


Figure 37. Publishing Delay Over Time, Engagement #5

The initial (orange) spikes on 5/9 and mid-day 5/10 were due to internal issues with the CADETS package that caused it to stop publishing data. Once restarted, it required several hours to 'catch up' to real time. The spike later on 5/10 was due to the fact that the THEIA team had attempted to repair an issue with their system and had stopped their endpoints in order to apply a patch. The slow growth of delay shown for 5/13 and 5/14 was due to the AIA agents appearing to suffer from resource exhaustion. We stopped the endpoints for a few hours to allow their pipeline to catch up and then restarted them. Finally, one Clearscope endpoint froze up on 5/15 and continued to have issues throughout the day. After being rebooted several times, the issue cleared up and the delay disappeared.

4.2.1.5.2. NTP Time Synchronization Monitoring

We also added monitoring for NTP time synchronization to look for the case where a TA1 host falls behind due to a clock synchronization error. We saw this case occur as a bug multiple times during previous engagements and developed the monitor to alert us if it occurred during the preparations for Engagement #5. The TA1 teams were able to fix the bugs causing this issue, and we did not see time synchronization errors occur during the engagement.

4.2.1.5.3. Network Overhead Metrics

During Engagement #5, we periodically ran a packet capture on all TA3 hosts to collect all incoming and outgoing network traffic for a period of two hours. Analysis of this captured data allows us to characterize the amount of overhead running the full TC system will add to existing network traffic. We organized this data into three components, TA1 to TA3, TA3 to TA3, and internal TA3. This collected packet capture data was cross-referenced with our internal monitoring data to ensure the data collection was comprehensive. For example, if a certain TA1 node published 1 GB of data during the captured two hours, we verified that at least that much traffic flowed from the TA1 to the proper TA3 Kafka Broker on port 9094.

4.2.1.5.3.1. TA1 to TA3 Traffic

This is the network traffic that the data collector TA1 components produce when writing all of their outgoing CDM data to the TA3 infrastructure. The actual traffic a deployed TA1 would generate was dependent on the TA1 configuration and the amount and characteristics of the activity on the TA1 system occurring during the collection period. The values in Table 22 were collected when we were running an estimate of "typical" traffic for a desktop computer in a basic office environment. Specific deployments of a TA1 may produce different traffic characteristics, and most TA1 systems can be specifically configured to produce more or less traffic, by whitelisting certain processes or types of records, however, these numbers can serve as a baseline estimate.

Table 22. Representative TA1 → TA3 Traffic

TA1	Network Traffic	Total Traffic	Kafka Traffic	Monitoring Traffic	Additional Traffic
Clearscope	237 MB	33.08 KB/s	33.03 KB/s	0.038 KB/s	Negligible
AIA	2.91 GB	404 KB/s	403.4 KB/s	0.59 KB/s	Negligible
MARPLE TA1	334.8 MB	46.5 KB/s	46.16 KB/s	0.32 KB/s	Negligible
THEIA	1.3 GB	185.2 KB/s	184.9 KB/s	0.16 KB/s	Negligible
TRACE	2.01 GB	290.7 KB/s	290.64 KB/s	0.16 KB/s	Negligible
CADETS*	2.49 GB	346.7 KB/s	346.5 KB/s	0.16 KB/s	Negligible
TA 5.2	8.5 MB	0.29 KB/s	0	0.29 KB/s	None

As expected, the primary contributor to the TA1 traffic load is the CDM traffic flowing from the TA1 to the TA3 Kafka cluster. The additional traffic is due to Kafka and Zookeeper management, and is negligible. The monitoring traffic is all optional, this allows for our monitoring infrastructure to gather real-time data about the performance of the hosts in the Engagement. Monitoring was far lower for Clearscope because we did not place a monitoring agent on the Clearscope Android phone itself. For the other TA1s, the Windows hosts (AIA and MARPLE) had larger monitoring overhead due to the fact that the Windows monitoring client produced more data than the Linux version. This is all configurable, so we could easily gather less information and create less network overhead. Since the total size of the monitoring network traffic was dwarfed by the Kafka CDM traffic, this sort of optimization would likely be irrelevant in operation.

The baseline US Army Cyber Protection Brigade software produced no traffic to our TA3 infrastructure apart from the monitoring clients. Since half were Linux and half were Windows VMs, the monitoring traffic values shown are an average of the two amounts of data. The TA5.2 systems do write information to their EndGame system, that amount of traffic would need to be measured separately.

We had to estimate the CADETS traffic from the monitoring data instead of using the packet capture due to issues with CADETS during the engagement. During the times we ran the packet capture, the CADETS system was not in normal operation, it was re-publishing previously captured data to fix observed issues. This meant that CADETS was publishing far more than normal during that time, in order to catch back up to real-time. In order to adjust for this, we estimated the network traffic by looking at a different time window in which CADETS was operating normally. CADETS also had a different network usage structure than the other performers in that the CADETS team took advantage of the TA3 services to support their internal processing, as shown in Figure 38.

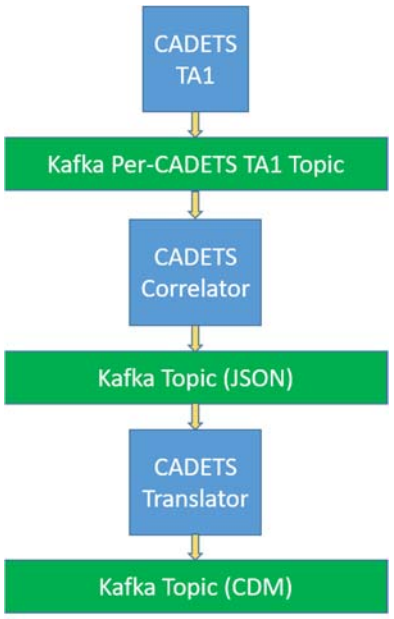


Figure 38. CADETS Data Flow Showing Internal Use of Kafka

CADETS TA1 endpoints published internal trace data to the Kafka infrastructure, then a CADETS correlator consumed that data, processed it, and published JSON back to a different Kafka topic. A CADETS translator process consumed the JSON data, processed it, and published CDM data needed by TA2 performers to a third topic. This results in many more TA1 to TA3 writes and reads than those exhibited by the other performers.

As an example, the following table shows the CADETS network usage breakdown. This CADETS overhead traffic could be reduced to just include trace data in one direction and the final CDM data, with a different (more complex) architecture. The CADETS team didn't choose this approach for Engagement 5 since the TA3 infrastructure was readily available, and easily able to handle the additional network bandwidth

Table 23. CADETS-Kafka Internal Traffic Volumes and Rates

CADETS Traffic	Direction	Traffic Volume	Traffic Rate
trace	TA1 → TA3	500 MB	69.4 KB/s
trace	TA3 → TA1	500 MB	69.4 KB/s
JSON	TA1 → TA3	210 MB	29.1 KB/s
JSON	TA3 → TA1	210 MB	29.1 KB/s
CDM	TA1 → TA3	1.07 GB	149.4 KB/s
TOTAL		2.49 GB	346.7 KB/s

4.2.1.5.3.2. TA3 to TA2 Traffic

TA3 to TA2 network overhead is more variable, since it is dependent on the analysis being run on the TA2 system at any given time. In general, a TA2 cluster that is keeping up to date with all TA1 produced traffic should consume the same amount of data that the TA1s produced, mirroring the numbers in the previous section. The TA2 data in the table below is an aggregate for the full 18 hosts of Engagement #5, since it is not possible to pull out exactly which TA1 topics were being consumed at the time the data was collected. In general, MARPLE consumed everything all the time and RIPE focused on specific TA1s (as instructed). We believe there was some data missing from the ADAPT numbers, accounting for the lower rate. In general, a TA2 will produce the same amount of network traffic as the sum of all TA1 traffic, since it would need to consume all of the produced data in order to analyze it in entirety.

Table 24. TA3 → TA2 Traffic Rates at Three Points in Time

TA2	Snapshot 1	Snapshot 2	Snapshot 3
ADAPT	66 KB/s	58 KB/s	62 KB/s
MARPLE	2.8 MB/s	1.8 MB/s	3.7 MB/s
RIPE	212 KB/s	132 KB/s	272 MB/s

4.2.1.5.3.3. TA3 to TA3 Traffic

TA3 internal traffic occurs between Kafka nodes and is nearly entirely due to keeping data replicas up to date. For Engagement #3, we used one replica for each CDM topic, so when CDM traffic arrived from TA1, we sent a copy of it to another Kafka node, incurring additional network overhead. This overhead is entirely configurable - since we used one replica per topic, we effectively doubled the TA1 → TA3 traffic.

All of the Kafka VMs were Linux hosts. The higher monitoring overhead (when compared to the TA1 Linux hosts) was due to the fact that we ran additional monitoring clients for the Kafka processes, which provided a great deal more data for monitoring and analysis.

Table 25. TA3 Internal Traffic over a Two Hour Period

TA3 Internal Traffic Type	Total	Rate
Kafka Replica	23.5 GB	3.27 MB/s
Monitoring	49 MB	0.35 KB/s
Additional (various)	1.8 MB	0.01 KB/s

4.2.1.5.4. TA1 Performance and Overhead

We computed the individual overhead imposed by each TA1 through the following experiment:

First, we ran the TA1 installed and configured on a host system with both publishing to the TA3 infrastructure and background traffic generation turned off. This provided a baseline environment.

Second, we then turned on TA1 publishing to TA3, and left off any additional background traffic generation.

Third, we turned off publishing and turned on background traffic generation,

Lastly, we ran with both publishing and background traffic enabled, which would be the standard operational environment.

Background traffic generation consisted of the scripts created by TA 5.1 to mimic regular usage of a target system. For each condition, we ran the system for at least 1 hour and recorded CPU usage, active memory, network data transfer, and disk usage metrics. The raw data recorded is given in the table below.

Comparing the metrics with background traffic on to the baseline gives us a measurement of the overhead incurred from just running the regular background traffic that we can take into consideration when analyzing the real operational environment with background traffic and publishing on. Likewise, comparing the metrics collected with only publishing on to the baseline, gives us a lower bound for the TA1 overhead, since TA1 clients provide no value without publishing data.

For the four classes of metrics we collected, CPU usage is likely the most straightforward to understand and report to a system administrator looking to deploy a TA1. For most TA1s there is a measurable, non-trivial CPU overhead associated with collecting and publishing their CDM data. Active memory usage is much harder to analyze, since operating system memory management is complex and it can be difficult to glean much from some aggregate statistics. In addition, most of the TA1s ran in a VM, which themselves ran on a host operating system, which further complicates memory management.

In some cases, active memory in use actually decreased from the baseline when publishing and traffic were enabled. The reasons for this can be complex - our initial analysis showed that the cached memory usage decreased, likely due to the fact that the operating system didn't need as much to devote to caching, since the system was far busier (consuming more CPU cycles). In more idle times, free memory can be allocated by the operating system proactively for things like caching frequently accessed data from the disk. In general, no TA1 exhibited a memory leak, which would manifest as steadily increasing usage or frequent garbage collection, and none had a memory related performance issue during the two week engagement.

The data transfer metrics correlated with our expectations. When publishing was enabled, data sent increases dramatically due to the TA1 writing out all the CDM traffic to the TA3 infrastructure. The disk usage metrics can again be challenging to analyze due to how the operating system and its memory management works with the disk controller. In general, there is a significant increase in disk usage for TA1 systems when publishing is on since many use the file system to transfer information across the internal components of their system.

Table 26. TA1 Metrics Collected Over One Hour - TRACE, THEIA, and CADETS

			CPU (% non-idle)			Active Memory (Total)			Data Transfer (Bytes/Sec)				Disk Usage			
TA1	Pub	BT	Min	Max	Avg	Min	Max	Avg	Recv Avg	Recv Max	Send Avg	Send Max	Read Avg	Read Max	Write Avg	Write Max
TRACE-1	No	No	4	12	5	13.2 GB	15.0 GB	13.3 GB	2.5 KiB	3.98 KiB	648 B	1.24 KiB	3 KiB	285 KiB	695 B	37 KiB
TRACE-1	Yes	No	4	12	9	13.2 GB	17.0 GB	15.7 GB	218 KiB	285 KiB	345 KiB	462 KiB	9 KiB	1.244 MiB	377 KiB	788 KiB
TRACE-1	No	Yes	1	27	5	16.4 GB	18.8 GB	17.4 GB	151 KiB	2.89 MiB	11 KiB	92 KiB	102 KiB	12.1 MiB	921 KiB	6.66 MiB
TRACE-1	Yes	Yes	20	45	27	10.5 GB	14.1 GB	11.3 GB	212 KiB	2.4 MiB	880 KiB	1.776 MiB	271 KiB	8.42 MiB	3.61 MiB	10.74 MiB
THEIA-1	No	No	1	48	8	1.23 GB	1.39 GB	1.25 GB	2 KiB	19 KiB	52 KiB	356 KiB	55 KiB	5.77 MiB	3 KiB	188 KiB
THEIA-1	No	Yes	1	88	32	1.3 GB	4.0 GB	3.6 GB	16.8 KiB	254 KiB	17.4 KiB	39.2 KiB	1.4 MiB	41.8 MiB	2.6 MiB	71 MiB
THEIA-1	Yes	No	1	49	18	248 MiB	260 MiB	255 MiB	2 KiB	4 KiB	110 KiB	310 KiB	4 KiB	250 KiB	233 KiB	706 KiB
THEIA-1	Yes	Yes	1	69	16	3.5 GB	3.85 GB	3.67 GB	4 KiB	287 KiB	87 KiB	479 KiB	29 B	1 KiB	303 KiB	2.61 MiB
THEIA-Analysis	No	No	1	1	1	661 MiB	672 MiB	664 MiB	840 B	1.77 KiB	469 B	3.29 KiB	116 B	11.6 KiB	1.4 KiB	49.6 KiB
THEIA-Analysis	No	Yes	1	1	1	663 MiB	664 MiB	664 MiB	799 B	1.1 MiB	403 B	770 B	0	0	134 B	4 MiB
THEIA-Analysis	Yes	No	1	6	3	655 MiB	3.8 GB	2.3 GB	219 KiB	619 KiB	204 KiB	578 KiB	83 B	819 B	2 MiB	6.0 MiB
THEIA-Analysis	Yes	Yes	5	25	12	11.26 GB	13.03 GB	12.23 GB	558 KiB	2.22 MiB	507 KiB	2.44 MiB	0	0	216 KiB	2 MiB
CADETS-2	No	No	1	1	1	49.93 MiB	63.62 MiB	55.66 MiB	1.07 KiB	2.25 KiB	241 B	789 B				
CADETS-2	Yes	No	50	100	53	54.3 MiB	89.4 MiB	70.1 MiB	1.8 KiB	8.3 KiB	6.0 KiB	29.9 KiB				
CADETS-2	No	Yes	1	22	5	12.2 MiB	32.7 MiB	16.3 MiB	54 KiB	529 KiB	300 KiB	833 KiB				
CADETS-2	Yes	Yes	51	99	61	63.2 MiB	138.9 MiB	104.2 MiB	5 KiB	25 KiB	50 KiB	328 KiB				
CADETS-Host	No	No	1	1	1	502 MiB	646 MiB	628 MiB	4.99 KiB	9.34 KiB	3.02 KiB	5.86 KiB				
CADETS-Host	Yes	No	13	26	14	787 MiB	855 MiB	824 MiB	115 KiB	459 KiB	129 KiB	644 MiB				
CADETS-Host	No	Yes	1	7	2	1.99 GB	2.1 GB	2.2 GB	763 KiB	3.15 MiB	1 MiB	3.45 MiB				
CADETS-Host	Yes	Yes	13	41	18	1 GB	2.19 GB	1.6 GB	679 KiB	4.05 MiB	917 KiB	5.6 MiB				

Table 27. TA1 Metrics Collected Over One Hour – MARPLE, AIA and Clearscope

			CPU (% non-idle)			Active Memory (Total)			Data Transfer (Bytes/Sec)				Disk Usage			
TA1	Pub	BT	Min	Max	Avg	Min	Max	Avg	Recv Avg	Recv Max	Send Avg	Send Max	Read Avg	Read Max	Write Avg	Write Max
MARPLE	No	No	1	2	1	28.1 GB	28.2 GB	28.1 GB	783 B	5.7 K	738 B	1.6 K	17 Kb	757 Kb	6 Kb	189 Kb
MARPLE	Yes	No	2	20	8	28.0 GB	28.1 GB	28.1 GB	1.0 Kb	2.8 K	6.6K	91.9K	4.7 Kb	213 Kb	8.9 Kb	64.7 Kb
MARPLE	No	Yes	1	22	4	27.7 GB	28.1 GB	27.8 GB	4.4 Kb	1 MB	3.5 K	202 K	39 Kb	3.5 MB	22 Kb	2.5 Mb
MARPLE	Yes	Yes	8	26	9	27.8 GB	27.9 GB	27.9 GB	1 K	7 K	10 K	524 K	82 B	6.7 Kb	8.6 Kb	60.2 Kb
AIA	Yes	No	12	20	14	26.3 Gb	26.5 GB	26.5 GB	5.8 K	775 K	1.27 K	19.6 K	1 kB	34 kB	162 kB	5.7 Mb
AIA	Yes	Yes	15	98	42	26.0 GB	27.3 GB	26.1 GB	21 K	418 K	209K	1.2 M	2 kB	143 Kb	324 Kb	1.6 Mb
AIA-Translate	Yes	No	24	35	28	13.3 GB	23.1 GB	18.3 GB	105 KiB	191 KiB	250 KiB	599 KiB	211 KiB	10.6 MiB	3.58 MiB	17.4 MiB
AIA-Translate	Yes	Yes	24	52	32	14.4 GB	21.0 GB	17.1 GB	99 KiB	570 KiB	646 KiB	5.85 MiB	351 KiB	8.2 MiB	16.1 MiB	30.7 MiB
Clearscope Host	Yes	n/a	1	6	1	14.5 GB	15.6 GB	15.4 GB	72 KiB	208 KiB	115 KiB	376 KiB	0	0	47.3 KiB	121 KiB
Clearscope Host	No	n/a	1	1	1	15.2 GB	16.2 GB	15.8 GB	4.8 KiB	7.5 KiB	2.08 KiB	3.42 KiB	5 KiB	282 KiB	46 KiB	97 KiB

For TA1 systems that included a target and an off host analysis VM, we recorded the overhead metrics for both the target and translator/analysis VMs independently (THEIA, CADETS, AIA). In general, the off host translator or analysis machine is less important for overhead calculations, since it is a single purpose VM that won't affect operational users.

CADETS is a different case, since the CADETS host functions as the hypervisor for the CADETS VMs, so an overloaded host could affect the performance of its guest VMs. CADETS did exhibit the largest overhead, with the target VMs using 50% of the CPU on average, spiking up to 100% at times. This would cause a noticeable effect to any user of the target system. CADETS may benefit from better hardware, but likely needs to be configured, analyzed, and potentially re-engineered.

TRACE showed a noticeable CPU overhead of around 22% over the baseline, which should still leave the target system quite usable and may be unnoticed by a user. THEIA was similar, with a smaller average CPU overhead, but a much higher maximum spike, which could be more problematic, depending on how often the spikes occur. In general, these CPU spikes correlate heavily to the type and amount of background traffic, so it is hard to give a general impression, and the system would need to be evaluated with the type of traffic and usage patterns for a given deployment.

MARPLE TA1 exhibited the lowest amount of CPU overhead, small enough that it should go unnoticed by a user. AIA is a special case, since AIA is never not publishing data to its off host translator. The AIA system was designed to be "tamper proof", so it can't be set to not publish. Therefore, there were no “publishing off” cases to compare with. This makes it more difficult to compare, and the AIA overhead is highly dependent on the amount and type of background traffic.

Finally, for Clearscope, we did not collect any metrics on the Android phones themselves, and will need to refer to MIT's own evaluation for overhead numbers for the phones themselves. The Clearscope off host translators were never stressed in terms of CPU usage, memory, network, or disk usage.

Table 28. Engagement #5 TA1 Data Production Results

TA1	Record Count A (Mil)	Record Count B (Mil)	Record Count C (Mil)	Total Size A (GB)	Total Size B (GB)	Total Size C (GB)	Total Records (Mil)	Total Size (GB)	Average Record Size
CADETS							1274	325	255.10
CADETS Internal	7.8	6.3	12.8	32	25	46	26.9	103	3,829.00
Clearscope	361	340	130	200.5	180.3	73	831	453.8	546.09
AIA	1220	1013	1110	257	227.7	237.6	3343	722.3	216.06
MARPLE	163	135	233	25.9	20.5	39.6	531	86	161.96
THEIA	414.6	446.2	319.6	84.2	89.8	65.6	1180.4	239.6	202.98
TRACE	1848	1909	1781	259	267	242.9	5538	768.9	138.84
Total							12724.3	2698.6	

4.2.2. Issues with Mobile Handsets

The Android mobile handsets used in the test range failed due to battery aging. We used simple low-cost powered USB hubs to connect the handsets to the servers that controlled them and were unable to monitor battery charge level or control the power supply to the handsets. As a result, the batteries were always charged to 100%, drastically shortening their lifespan. In addition, because of space and clearance limitations in our data center we had built our 'mobile phone test rigs' inside our equipment racks towards the top – where the heated airflow from the other systems in the rack would encounter them as it exited the rack (see Figure 39). We neglected to consider the effect of the elevated temperatures on the batteries, which also had a negative effect on them.

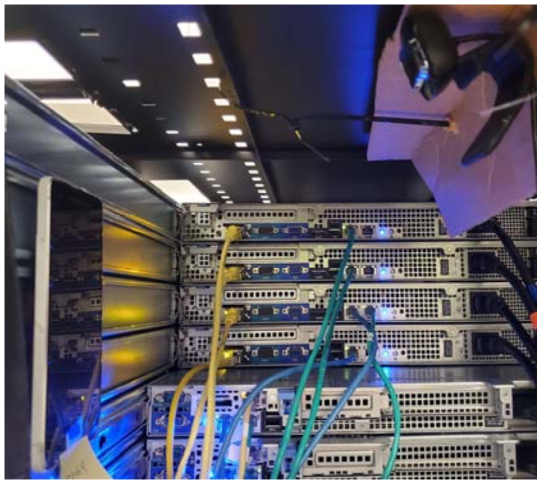


Figure 39. Mobile Phone Harness with Handset and WebCam

Should we have to support handsets in a future deployment, we should consider using more expensive industrial grade USB hubs that allow for power monitoring and control, allowing us to keep the batteries between a 50% and 80% charge level and extending their lives.

4.3 Policy Enforcement Demonstrations

As mentioned in 3.5.4.1, we proposed to create a demonstration that illustrated how the trace data created by the TA1 clients could be used to enforce security policies on the network. We developed a set of policies we wanted to enforce, worked with the TA1 and TA2 participants to identify the events leading up to the critical decision for each policy, then added a Policy Enforcement Service that queries the TA2 services to determine if an operation should be permitted or not.

This demonstration evolved over engagements 3, 4 & 5. In the first, a simple set of policies were defined and we were able to demonstrate the basic functionality and identify the importance of the "TA1 delay" and its effect on the overall system. In Engagement #4, we added scenarios, we expanded the policies to require the involvement of multiple TA2s, cross-host parameters, and additional query parameters. For Engagement #5, we kept the same structure, policies, server application, and Policy Enforcement Module as we had in Engagement #4, reduced the number of scenarios from 12 to 8, and allowed for more requests per scenario.

4.3.1. Engagement #3

We ran 7 requests for each scenario, for each TA2. Each request was either blocked or allowed, and we recorded whether the TA2 took the appropriate action.

Ground truth (explanations of how we generated each request, and whether it should be blocked or allowed) is provided in 0 below.

4.3.1.1. MARPLE

MARPLE provided very detailed analysis of the CDM for the AIA and THEIA errors, showing exactly what their system extracted from the CDM. The errors for AIA and THEIA appear to be a TA1 issue, where the proper linkage is missing from the TA1 data. The policy demonstration request exercises a different portion of the space of CDM records and linkages, so issues like this were uncovered only in the policy demonstration results and not in the main Engagement results.

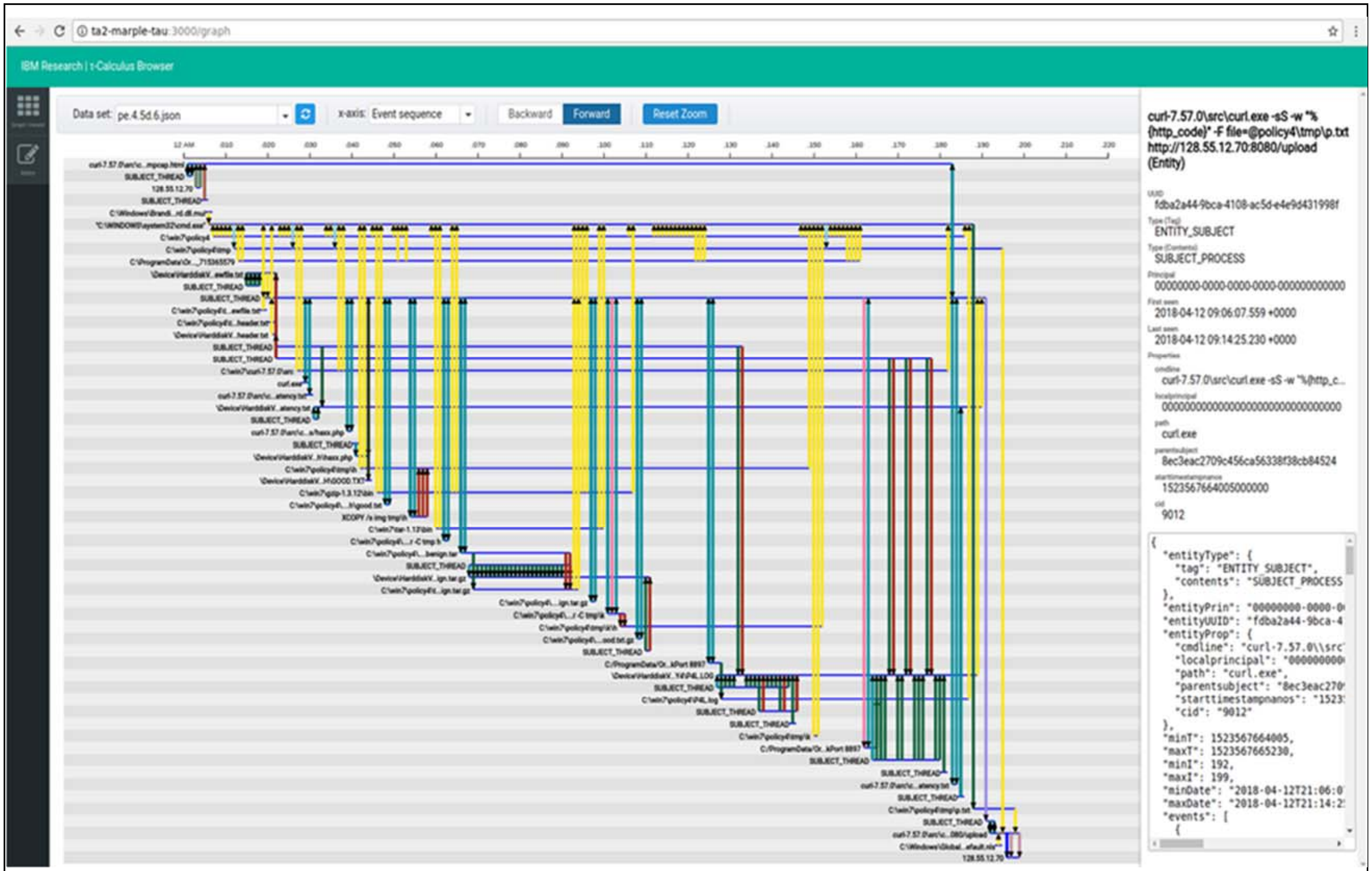
Table 29. MARPLE Engagement #3 Policy Demonstration Results

TA1 Client	Policy	Expected Results
CADETS	(1) Originating User	**All Correct**
TRACE	(4) Net Data	**All Correct**
5D	(4) Net Data	Three errors appear to be data/linkages missing from TA1 CDM
FAROS (PSA)	(2) Communication	No block, unable to link PSA records
THEIA	(2) Communication	Two errors appear to be TA1 CDM data/linkage errors
Clearscope	(3) UI	**All Correct**

MARPLE had multiple modules in their TA2 server for some policies and a voting mechanism for the final result. If any one out of two modules returns BLOCK, then the final result was a BLOCK, else PASS. They included optional explanations for BLOCK actions that linked the CDM record at the end of the casual chain, and expect to be able to provide entire causal chain if desired.

For FAROS, it appeared to be technically possible to get the right answer with FAROS PSA, but would require custom code just for this purpose. FAROS PSA data was a new form of data that was not used in the main engagement and was provided very late, due to being a late addition to FAROS. The main issue was that there were no EVENTs to link objects, requiring very different analysis methods, since the information needed to link objects was present only in the properties map.

Figure 40 below is an example of the type of analysis the MARPLE TA2 system performs to track a file upload (Policy 4). This example was from AIA, Policy 4, request # 4 where the file "**haxx.php**" was downloaded from a remote server onto the AIA host, then a series of file operations were completed in an attempt to hide the file (zip, rename, unzip, etc.). This graph from MARPLE shows how they were able to link the curl command that sent the file upload request to the policy protected server to the original **haxx.php** file download event from earlier, through all of the file operations.



curl-7.57.0\src\curl.exe -sS -w "%{http_code}" -F file=@policy4\tmp\p.txt http://128.55.12.70:8080/upload (Entity)

UUID
fdba2a44-9bca-4108-ac5d-e4e9d431998f
Type (Tag)
ENTITY_SUBJECT
Type (Contents)
SUBJECT_PROCESS
Principal
00000000-0000-0000-0000-000000000000
First seen
2018-04-12 09:06:07.559 +0000
Last seen
2018-04-12 09:14:25.230 +0000
Properties
cmdline
curl-7.57.0\src\curl.exe -sS -w "%{http_c...
localprincipal
00000000000000000000000000000000
path
curl.exe
parentsubject
8ec3eac2709c456ca56338f38cb84524
starttimestampnanos
1523567664005000000
cid
9012

```
{
  "entityType": {
    "tag": "ENTITY_SUBJECT",
    "contents": "SUBJECT_PROCESS
  },
  "entityPrin": "00000000-0000-0
  "entityUUID": "fdba2a44-9bca-4
  "entityProp": {
    "cmdline": "curl-7.57.0\\src
    "localprincipal": "000000000
    "path": "curl.exe",
    "parentsubject": "8ec3eac270
    "starttimestampnanos": "1523
    "cid": "9012"
  },
  "minT": 1523567664005,
  "maxT": 1523567665230,
  "minI": 192,
  "maxI": 199,
  "minDate": "2018-04-12T21:06:0
  "maxDate": "2018-04-12T21:14:2
  "events": [
    {
```

Figure 40. MARPLE Event Flow Graph

4.3.1.2. ADAPT

ADAPT requested the filename be provided by the PEM for Policy 4, which we offered as an option for this first iteration of the policy demonstration. The CDM data linkage errors for AIA were the same TA1 issues that MARPLE found, where the TA1 CDM data didn't link events properly. The differences in the THEIA results were due to how ADAPT interpreted the policy, specifically how to handle subprocesses. They were able to see the call chain and just interpreted the policy differently. If interpreted the same way, ADAPT would have likely encountered the same linkage issues as MARPLE for 2 of the 3 tests. ADAPT was also not able to process anything from FAROS data, since it was provided so late and was so differently structured than any other CDM data.

Table 30. ADAPT Engagement #3 Policy Demonstration Results

TA1 Client	Policy	Expected Results
CADETS	(1) Originating User	**All Correct**
TRACE	(4) Net Data	**All Correct**
AIA	(4) Net Data	*Three errors appear to be data/linkages missing from TA1 CDM*
FAROS (PSA)	(2) Communication	*No block, unable to do anything with PSA*
THEIA	(2) Communication	*Three differences due to interpreting the policy*
Clearscope	(3) UI	**All Correct**

ADAPT provided English explanations for their policy BLOCKs, some examples are provided below:

- “No network source data for the file: newupload.txt”
- 128.55.12.71:8080. This is the IP/Port of a NetFlow that violated the restricted server policy
- username used to make the request was: root. This query was testing for: admin

This graph below depicts the type of analysis that ADAPT performed. This shows them tracing a received NetFlow object back to the parent process that generated it, and to its subprocesses. These subprocesses then link to other earlier NetFlows, showing which other remote servers this process has communicated with in the past (bottom right of the figure). This analysis and traversal of this graph allowed ADAPT to answer the remote communication policy checks. If they interpreted the policy with regards to subprocesses in the intended way, this graph shows that they had the correct information traced and could have returned the expected result.

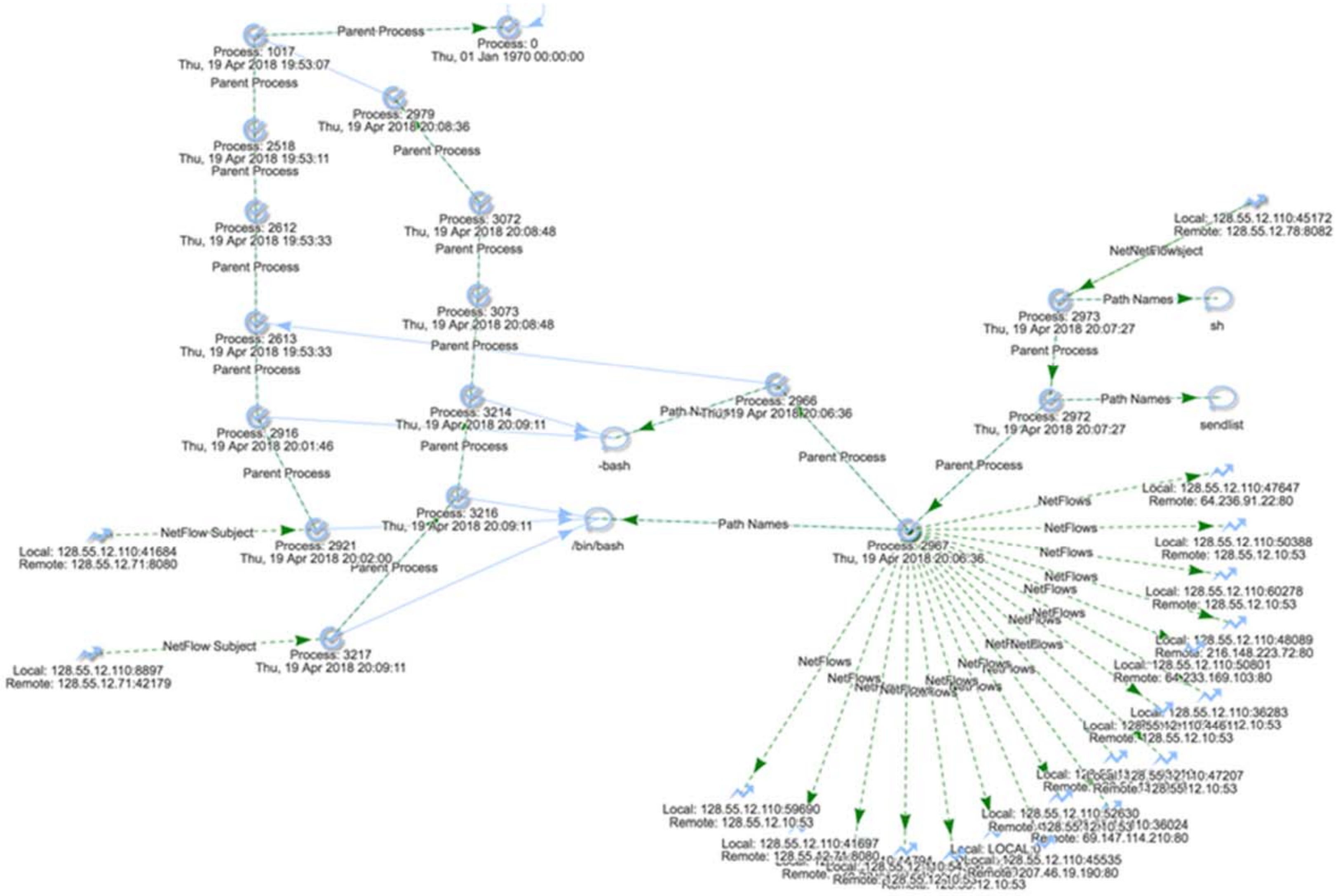

Process: 0
Thu, 01 Jan 1970 00:00:00
Process: 1017
Thu, 19 Apr 2018 19:53:07
Process: 2979
Thu, 19 Apr 2018 20:08:36
Process: 2518
Thu, 19 Apr 2018 19:53:11
Process: 2612
Thu, 19 Apr 2018 19:53:33
Process: 3072
Thu, 19 Apr 2018 20:08:48
Process: 3073
Thu, 19 Apr 2018 20:08:48
Process: 2613
Thu, 19 Apr 2018 19:53:33
Process: 3214
Thu, 19 Apr 2018 20:09:11
Process: 2916
Thu, 19 Apr 2018 20:01:46
Process: 2966
Thu, 19 Apr 2018 20:06:36
-bash
/bin/bash
Process: 3216
Thu, 19 Apr 2018 20:09:11
Process: 2921
Thu, 19 Apr 2018 20:02:00
Process: 3217
Thu, 19 Apr 2018 20:09:11
Parent Process
Path Names
NetFlow Subject
NetFlows
Local: 128.55.12.110:41684
Remote: 128.55.12.71:8080
Local: 128.55.12.110:8897
Remote: 128.55.12.71:42179
Local: 128.55.12.110:45172
Remote: 128.55.12.78:8082
Process: 2973
Thu, 19 Apr 2018 20:07:27
sh
Process: 2972
Thu, 19 Apr 2018 20:07:27
sendlist
Process: 2967
Thu, 19 Apr 2018 20:06:36
Local: 128.55.12.110:47647
Remote: 64.236.91.22:80
Local: 128.55.12.110:50388
Remote: 128.55.12.10:53
Local: 128.55.12.110:60278
Remote: 128.55.12.10:53
Local: 128.55.12.110:48089
Remote: 216.148.223.72:80
Local: 128.55.12.110:50801
Remote: 64.233.169.103:80
Local: 128.55.12.110:36283
Remote: 128.55.12.10:53
Local: 128.55.12.110:47207
Remote: 128.55.12.10:53
Local: 128.55.12.110:59690
Remote: 128.55.12.10:53
Local: 128.55.12.110:41697
Remote: 128.55.12.71:8080
Local: 128.55.12.110:52630
Remote: 128.55.12.10:53
Remote: 69.147.114.210:80
Local: LOCAL:0
Local: 128.55.12.110:45535
Remote: 207.46.19.190:80

Figure 41. ADEPT Event Flow Graph

4.3.1.3. RIPE

RIPE encountered similar issues with missing data linkages from the TA1 CDM (AIA) and differences in interpreting policy details. In addition, RIPE (like the other TA2s) required custom code to handle some of the TA1 data, since the policy demonstration exercised different aspects of the TA1 data from those used in normal event processing. They ran out of time to analyze the TA1 sample data and implement much of the custom methods (e.g. FAROS, Clearscope).

- **CADETS**: Requests involving a C program using the **setuid** system call were not handled correctly, RIPE needs to update code to handle this condition
- **TRACE**: Two expected block were allowed, RIPE needs to investigate further
- **AIA**: Same issue as other TA2s, missing linkages in the AIA data, confirmed as something that needs to be fixed by Five Directions
- **THEIA**: As with ADAPT, requests involving child processes were interpreted differently (ignoring parent process communication with the restricted host). The policy was ambiguous.
- **FAROS**: Analysis of CDM determined that new code would be required that they didn't have time to implement
- **CLEARSCOPE**: Ran out of time to implement.

4.3.1.4. Engagement #3 Results

We showed that the Transparent Computing infrastructure, including TA1, TA2s and the TA3 services, could be used to enforce policies in real time. Proper implementation of a policy enforcing TA2 server requires careful analysis of the TA1 data stream, and can require different analysis routines that look at different aspects of the TA1 data. Thus, time and engineering effort is needed to implement a new policy and tailor it to the specific TA1 providing the data. For future experiments, we planned to gather sample TA1 data as early as possible. This became a consistent challenge in the program since TA1s were constantly updating their software, invalidating the old sample data and requiring new sample data to be generated. TA2s needed to example and work with the latest TA1 sample data in order to automate their analysis. Automation is necessary here to be able to return a policy check result in real time. However, with enough engineering effort, we were able to produce correct policy check results with an overall latency in the seconds to tens of seconds range. This can be sufficient to implement a provenance policy protected server application, provided that the policy check is only triggered occasionally on specific requests. A multi-second latency can work on a highly protected application, for certain types of occasional requests, such as file uploading or downloading from a secure server.

4.3.2. Engagement #4

For Engagement #4, the policy demo was substantially increased in scope and complexity, with server side policies being tested as well as client side policies. The server side policies were more complex due to having multiple processes involved (web server with multiple threads and a PHP interpreter) as well as having cross host traffic from the web server to a data base server. The database server was run on a separate different TA1, forcing the TA2 analysis to first trace the NetFlows to the second server and second cross-reference data between two different TA1 CDM streams. In addition, we ran client side policies for each TA1 separately. This resulted in

12 scenarios (6 client side policies, and 6 server side policies, with each TA1 performing each role). This increased complexity, combined with TA1s being delivered late and with bugs that needed to be fixed (and therefore sample data that needed to be regenerated) resulted in a shortened time for TA2s to analyze the policy demo sample data. Regardless of this added complexity and challenges, at least one TA2 was able to properly respond to policy requests for each condition, with few exceptions.

We provided sample data with ground truth to all TA2s for each scenario months to weeks in advance, depending on the TA1s involved and when they were ready. This sample data had to be regenerated multiple times as TA1s fixed bugs, leading to some of the original data being less useful overall due to TA1 changes. For Clearscope, we designed custom Android server applications. We configure an existing app store web application for each TA1 environment, requiring us to port the web application to Apache2, nginx, php and postgres, on Windows, Linux, and FreeBSD environments. We designed request scripts that challenge TA2 analysis procedures by mimicking attacker attempts to bypass a provenance based policy.

For the live policy demonstration, we generated fresh CDM data sets from running the TA1s, web applications, and requests live. Our policy enforcement module sent requests to TA2 servers in real-time. We waited on average 20 seconds for the CDM data to propagate through the system before sending the requests to the TA2 server. This delay could be significantly reduced with more engineering time, we expect to be able to get this delay to around 5 seconds or so. The TA2 systems answered the query in ~1-30 seconds, depending on the complexity of the application, policy, and TA1 system. For the live demonstration, we generated similar requests to the sample data, varying order and timing. In addition, we generated bonus requests not in the sample data if the sample requests were handled correctly.

4.3.2.1. MARPLE

Table 31. MARPLE Engagement #4 Policy Enforcement Results

TA1 A	TA1 B	Policy	# Correct	Notes
TRACE		UI	3/4	Incorrectly blocked one UI request. Regenerated sample data often before the demo, updating the service startup sequence
MARPLE		UI	2/4	TA2 internal error on UI requests
AIA		UI	5/5	Bonus request completed correctly
THEIA		UI	2/4	All blocked, no UI events found
CADETS		UI	SKIP	TA2 Not ready to process
CLEARSCOPE		UI	SKIP	TA2 Not ready to process
TRACE	MARPLE	User	6/6	All correct, plus bonus
MARPLE	THEIA	Comms	3/5	Cross host tracking failed, no database server results returned. Perfect results for MARPLE server
THEIA	AIA	User	3/5	Backtracking broken for AIA (TA2 issue), cross host links found, but unable to return results.
AIA	CLEARSCOPE	Files	SKIP	TA2 Not ready to process
CLEARSCOPE	AIA	Comms	5/5	All correct

4.3.2.2. RIPE

Table 32. RIPE Engagement #4 Policy Enforcement Results

TA1 A	TA1 B	Policy	# Correct	Notes
AIA		UI	SKIP	TA2 Not ready to process
CADETS		UI	SKIP	TA2 Not ready to process
CLEARSCOPE		UI	SKIP	TA2 Not ready to process
AIA	CLEARSCOPE	Files	0/4	Can't do cross-host tracking in real-time, only analyze Server 1. Expected FILE_READ events, only found FILE_OPEN events
CLEARSCOPE	AIA	Comms	3/5	IP Addresses encoded differently in CS data, had to manually modify PEM to TA2 request. Tried additional Policy One (User) request, which was also correct.
CADETS	TRACE	User	3/5	Failed to trace back to the originating user, always returned the admin user. TA2 needed more time with the CADETS data.

4.3.2.3. ADAPT

Table 33. ADAPT Engagement #4 Policy Enforcement Results

TA1 A	TA1 B	Policy	# Correct	Notes
TRACE		UI	5/5	Treat anything that can be traced back to xvnc as a UI action
MARPLE		UI	4/4	All correct
5D		UI	5/5	Bonus request completed correctly
THEIA		UI	2/4	All blocked, insufficient data result returned from ADAPT
CADETS		UI	3/4	One UI request was blocked, unclear why, offline debugging needed
CS		UI	2/4	All blocked, no UI data found. Timestamp issue, offline forensic analysis found the expected UI data
TRACE	MARPLE	User	2.5/5	Netcat process not found for Request Three, Wrong user returned for requests Four and Five (user currently running the process, not user that started it)
MARPLE	THEIA	Comms	SKIP	Ran out of time
THEIA	AIA	User	2.5/5	Netcat process not found, wrong user returned
AIA	Clearscope	Files	1/3	Trouble finding files read due to timestamp mismatch, queried for file origination directly, failed to find network origination
Clearscope	AIA	Comms	3/5	Unable to find cross host traffic due to IP address formatting
CADETS	TRACE	Files	3/5	TA2 Internal server errors, re-ran forensically afterwards. Didn't find network origination for one request, didn't find SQL

4.3.2.4. Engagement #4 Results

As expected, we observed a strong correlation between data being ready early and good results. The TRACE/MARPLE pair and AIA data were ready the earliest, so the TA2 performers had longer to analyze and engineer their servers to process this data. CADETS, on the other hand, was not ready to produce data until very late, leading to most TA2s being not ready to properly analyze it.

All TA2s were able to answer some queries in real-time, depending on which policies and TA1s they focused their effort on. RIPE, for example, has a nice web service interface to their TA2 server that lets the experimenter see what they're tracking. In general, most TA2s were too busy with engagement preparation to devote the necessary time to analyze all 12 scenarios. The complexity of the policy demonstration resulted in a larger engineering challenge for TA2s due to the necessity for full automation to handle the real-time requests. TA2 analysts were not able to fall back to manual or forensic analysis if something went wrong. In general, the TA2s needed more time to study the data and implement different analysis routines than what was needed for Engagement #4.

4.3.3. Engagement #5

We only had a single TA2 participant who could support the policy enforcement demonstration for Engagement #5, the MARPLE team.

For Engagement #5, we switched the web server to FastCGI mode where the PHP interpreter ran as a separate, long lived process. This meant that TA2s would need to track the **php-cgi** process and trace inter-process communications between it and the web server. In previous engagements, with FastCGI mode off this wasn't necessary, and they could simply follow the chain of process creation to see the server fork-exec the PHP process. This FastCGI inter-process communication caused problems for many TA1s since they didn't have the proper provenance tags on interprocess communication events.

Figures 42 through 44 showing the internal representation of MARPLEs analysis. These graphs show that MARPLE can see many inter-process communication calls between **httpd** and **php-fpm**, but lacks the necessary provenance tags to trace an individual request from **httpd** to **php-fpm**, then from **php-fpm** cross-host to the database server, and then all the way back. Because the provenance information was not available in the TA1 data, MARPLE could not disambiguate between the separate calls. The only TA1 that potentially provided the proper provenance tags was Clearscope, but because Clearscope runs on a mobile handset and was not designed to run as a remotely accessible server it would have required significant additional development to make it useful for this purpose.

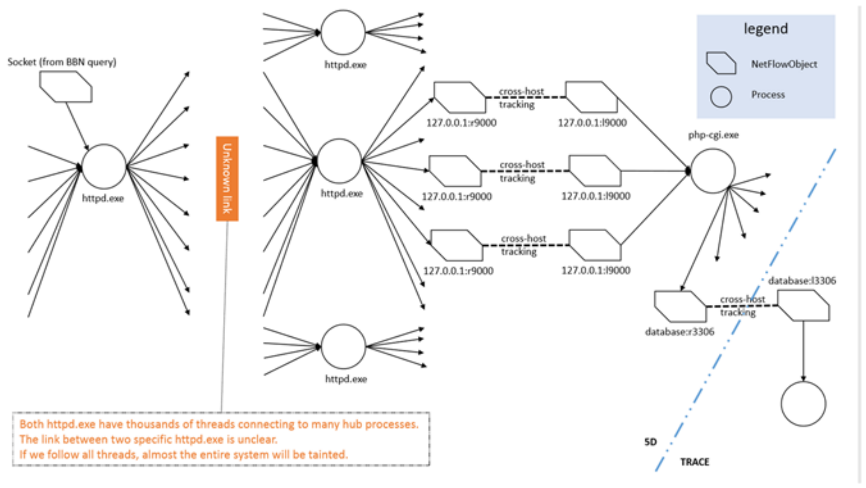

legend
NetFlowObject
Process
Socket (from BBN query)
httpd.exe
httpd.exe
httpd.exe
httpd.exe
Unknown link
cross-host tracking
127.0.0.1:r9000
127.0.0.1:l9000
php-cgi.exe
database:l3306
database:r3306
Both httpd.exe have thousands of threads connecting to many hub processes.
The link between two specific httpd.exe is unclear.
If we follow all threads, almost the entire system will be tainted.
5D
TRACE

Figure 42. Event/Flows for Policy Demo Showing Missing IPC Provenance Data for AIA and TRACE

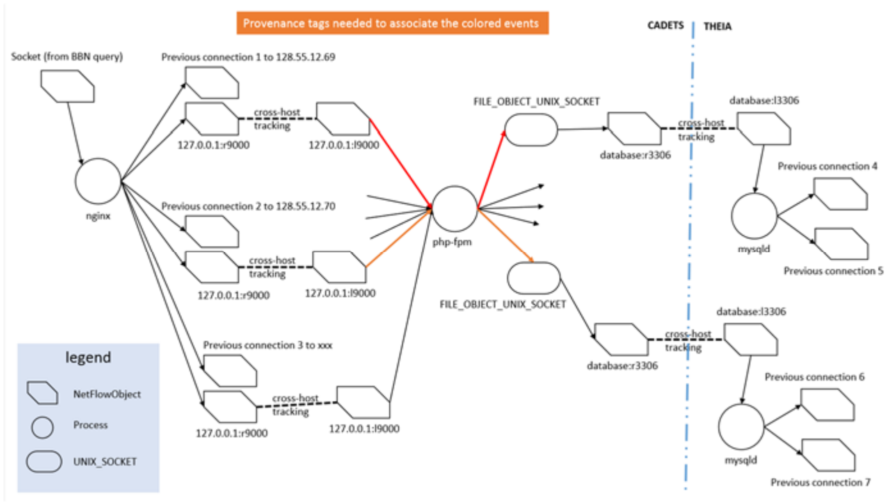


Figure 43. Event/Flows for Policy Demo Showing Missing IPC Provenance Data for CADETS and THEIA

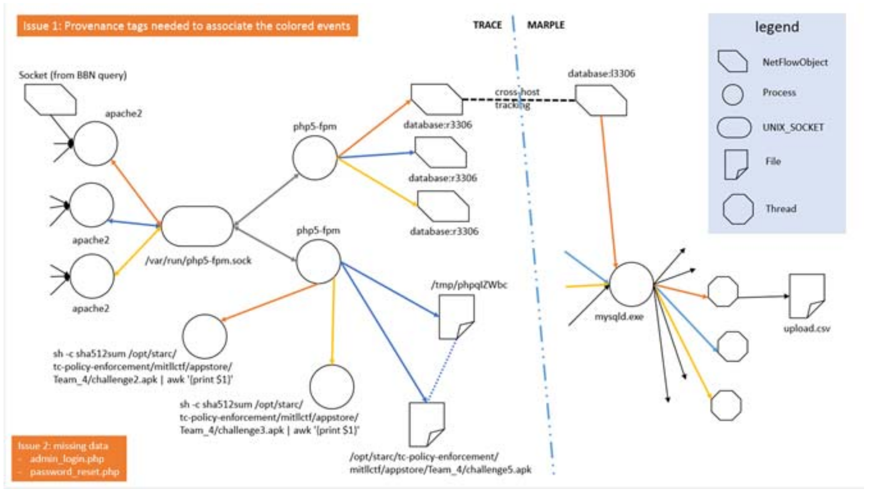

Issue 1: Provenance tags needed to associate the colored events
TRACE
MARPLE
legend
NetFlowObject
Process
UNIX_SOCKET
File
Thread
Socket (from BBN query)
apache2
apache2
apache2
/var/run/php5-fpm.sock
php5-fpm
php5-fpm
database:r3306
database:r3306
database:r3306
cross-host tracking
database:l3306
mysqld.exe
upload.csv
/tmp/phpqlZWbc
sh -c sha512sum /opt/starc/ tc-policy-enforcement/mitllctf/appstore/ Team_4/challenge2.apk | awk '{print $1}'
sh -c sha512sum /opt/starc/ tc-policy-enforcement/mitllctf/appstore/ Team_4/challenge3.apk | awk '{print $1}'
/opt/starc/tc-policy-enforcement/ mitllctf/appstore/Team_4/challenge5.apk
Issue 2: missing data
- admin_login.php
- password_reset.php

Figure 44. Event/Flows for Policy Demo Showing Missing IPC Provenance Data for TRACE and MARPLE

For requests where this interprocess tracking was not necessary, MARPLE returned the expected result with no false positives. However, most of the requests did require this sort of tracking. These graphs show how MARPLE was able to analyze what happened and that the issue was a lack of necessary information in the TA1 data. Since this demonstration and analysis occurred at the end of the program, there was no time to iterate with the TA1s, leaving providing this level of provenance tagging in the TA1 data an open area of research.

5.0 CONCLUSIONS

5.1 Architecture

The core goals of the TC architecture were to:

- Provide for a multi-layer data collection architecture;
- Support overall system design and development, including security, APIs, and data format specifications; and
- Allow for adaptation to new findings as the program progressed

Based on the extensive performance testing that BBN conducted before each engagement, combined with the results from each engagement, our conclusion is that the collection architecture successfully met the program goals. The Kafka system provided sufficient throughput for TA2 needs and 100% data reliability for all data collected in all five engagements.

Security was addressed primarily via Kafka support for encryption and authentication. Attempts to run Kafka on TC instrumented systems such as CADETS were not successful, however, due to the level of overhead imposed on the target system.

The APIs and data format specification as captured in the CDM were sufficient to allow accurate event reconstruction and APT detection across multiple platforms. However, full semantic integration was never truly achieved due to underlying native platforms differences and differing priorities among TA1 and TA2 performers.

As can be seen from the architecture diagrams from each engagement in section 3.4, the architecture did evolve over time as subsequent engagements revealed new requirements or issues. The flexibility of the architecture was critical to the success of this effort overall.

5.2 Common Data Model

The CDM was a core component of the TA3 effort and probably the most visible element of BBN's work across the collaboration. While CDM development continued throughout the program, the bulk of the development was completed by Engagement #3 with the release of v0.19. At that point, the CDM was sufficient to provide a common format for all relevant provenance data across the TC platforms (Linux, OpenBSD, Windows 7/10, and Android).

The main limitation of CDM was always the relative inefficiency of the format. Due to the limited scale of TC testing and evaluations, this inefficiency was never a limiting factor and the format allowed easier debugging and testing than a fully optimized and packed binary format. However, as was apparent in the first phase of the OpTC effort, the CDM would require an optimization rewrite to allow TC to scale to thousands or more hosts. BBN's conclusion is that the CDM served the DARPA development phase adequately, and a transition CDM or equivalent would likely be needed when scaling and application to a real domain were required.

In addition, TA2 performers generally felt that the CDM semantics were never truly converged among the TA1 performers, requiring them to create separate input handling routines for each TA1 regardless of the CDM standard. BBN's conclusion is that a tighter process on semantic agreement among teams and platforms starting in Phase One would have benefited the overall collaboration greatly, although it would have required more work on the part of TA1s to pay attention to the CDM standard development process details.

5.3 Infrastructure

The BBN test infrastructure proved to be sufficient to provide the necessary capabilities (processing, throughput, storage) for TC integration and evaluations. As BBN was able to utilize a combination of its own equipment plus some GFE (primarily for TA2 high capacity processing), we had the convenience of being able to run the test range as much as we desired over the course of the program, without worrying about cloud computing overhead costs. We did not evaluate the comparative cost of such an infrastructure against a cloud-based setup to provide us the same capabilities; see the section on the ORange process and evaluation to see an examination of feasibility and capabilities for a cloud-based alternative to the TC infrastructure.

5.4 Engagements

The main conclusion that BBN derived from the five TC engagements was the necessity for creating a strong and well defined process to provide repository, build, test, and installation functionality that all performers are required to use. In addition, good software practices such as unit tests and true code freezes providing adequate time for both risk reduction and engagements are a necessity to allow proper evaluation of technology strengths and capabilities without requiring heroic effort on the part of all parties. Direct communications such as live chat and blogs provide essential communications between geographically separated parties participating in test run up and live test periods. Finally, post-test lessons learned for all involved parties are very useful for evaluating issues and improving processes for future iterations.

5.5 Policy Enforcement Demonstration

The Policy Enforcement Demonstration was an interesting alternative to incorporate policy enforcement directly within the TC engagement scenarios, which seemed to be the initial intent expressed in the BAA. This method of testing enforcement meant that blocking actions would not impact engagement scenarios, allowing those events to continue without interruption and potentially time-consuming resets of mechanisms and policies. However, the separate demonstration did require separate preparation, especially among TA2 performers, who needed to examine very different data sets after the main engagement ended.

Overall, the Policy Enforcement Demonstration did allow a focused examination of security enforcement processes and mechanisms that could be enabled via TC-enabled systems. This focus could have been lost or missed due to conflation with missed detections and other issues that cropped up during the actual engagements, so the isolation of variables certainly helped create a more clear and interpretable situation with regards to true effectiveness of the mechanisms under test.

6.0 RECOMMENDATIONS

Overall, the base TC architecture worked well. We would recommend examination of the Kappa and Lambda architecture patterns as well as any updated work in this area for future use in any program of this type.

CDM exhibited a great deal of value in that it enabled a set of different analysis tools sourced from different vendors to identify and detect behaviors in a heterogeneous operating environment. The single model allows for more than just malware detection, but other types of failures as well. The existence of such a capability in both military and commercial environments would greatly enhance the ability to create heterogeneous system monitoring and administration tools in such environments.

While the CDM fulfilled its purpose for TC, the process used for its development and results could have been improved. We have two main recommendations if CDM itself or a CDM-like product is created again:

- Force teams to agree much earlier about semantics for items in a CDM
- Support development by both client and analysis tool vendors by developing searchable and comprehensive documentation on the meaning of events and how they should be represented in CDM
- Be prepared to create/use a much more efficient version of a CDM if transition is envisioned/needed soon after the program ends

TC worked well in a stable networked environment, but the need to configure each client to connect to a fixed address Kafka server would not work well in a more dynamic networking situation. It could be worthwhile to take a page from the 'Smart Cities' and Internet of Things initiatives where a CDM source could be envisioned as a data stream source, and use a more dynamic networking infrastructure such as the Distributed Data Service (DDS) for a least the first link in the collection of CDM data from endpoint hosts. DDS offers a minimal configuration, self-adapting data transmission subsystem designed and developed for real-time data exchange in dynamic environments.

Tasks like the Policy Enforcement Demonstration, where a separable element/environment/goal set based on the mechanisms under development are established, should be incorporated into future programs in this area of study to allow greater isolation of variables and more focused examination and consideration of results. Such tasks serve to explore and demonstrate capabilities and features not normally associated with those immediately obvious to those focusing on the primary program goals and metrics.

For engagement management, our chief recommendation here is to require the integrator to create adequate and easily accessible code submission and testing facilities. The current industry term for such capabilities is "Development Operations" (DevOps) and/or "Continuous Integration" (CI). Performers should be encouraged to incorporate the DevOps system into their development process, and submission to the test range should require the use of the DevOps system. If possible, the DevOps system should be containerized so that it can be replicated to some degree at each performer site to support local development and pre-testing before submission to the shared repository.

The integrator should also aggressively enforce a code submission/freeze schedule for performers to ensure that their code is working on the test/integration/evaluation range ahead of time. A code freeze of one month before integration is not excessive in a 4 year program with 5 engagements as inevitable problems will arise and require time to address.

Due to the potential issues of running APTs or software with APT-like behaviors within a commercial cloud infrastructure, we recommend that DARPA continue funding GFE for local test clusters that can be run in isolation and without potential legal complications of employing commercial or operational cloud resources.

Given that, our recommendations for a potential infrastructure setup are contained in B.5.

7.0 REFERENCES

[1] N. H. M. e. a. Marz, "Lambda Architecture," 2017. [Online]. Available: http://www.lambda-architecture.net.

[2] W3C, "PROV-DM: The PROV Data Model: W3C Recommendation 30 April 2013," 30 April 2013. [Online]. Available: https://www.w3.org/TR/prov-dm/.

[3] N. Marz and J. Warren, Big Data: Principles and best practices of scalable realtime data systems, Greenwich, CT: Manning Publications Co., 2015.

[4] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker and S. A. Haghighat, "Practical domain and type enforcement for unix," *Proceedings of the IEEE Symposium on Security and Privacy,* pp. 66-77, 1995.

[5] A. Keromytis, "Transparent Computing," 5 December 2014. [Online]. Available: https://beta.sam.gov/api/prod/opps/v3/opportunities/resources/files/9fe73ed863c590d63b0a503e21bee06d/download?api_key=undefined&status=archived&token=. [Accessed 19 December 2019].

[6] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang and D. Xu, "MPI: Multiple Perspective Attack Investigation with Semantics Aware Execution Partitioning," in *Proceedings of the 26th USENIX Security Symposium*, Vancouver, 2017.

[7] Y. Kwon , F. Wang , W. Wang , K. H. Lee , W.-C. Lee , S. Ma, X. Zhang, D. Xu, S. Jha, G. F. Cretu-Ciocarlie, A. Gehani and V. Yegneswaran, "MCI : Modeling-based Causality Inference in Audit Logging for Attack Investigation," in *NDSS*, San Diego, 2018.

[8] W. Enck, P. Gilbert, B.-g. Chun, L. P. Cox, J. Jung, P. McDaniel and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime PrivacyMonitoring on Smartphones," in *OSDI 10*, Vancouver, 2010.

[9] H. Yin, D. Song, M. Egele, C. Kruegel and E. Kirda, "Panorama: Capturing System-wide Information Flow forMalware Detection and Analysis," in *CCS 07*, Alexandria, 2007.

[10] L. M. Rossey, R. Cunningham, D. J. Fried, J. C. Rabek, R. P. Lippmann, J. W. Haines and M. A. Zissman, "LARIAT: Lincoln adaptable real-time information assurance testbed," in *Proceedings, IEEE Aerospace Conference*, Big Sky, 2001.

[11] M. Navaki Arefi, G. Alexander, H. Rokham, A. Chen, M. Faloutsos, X. Wei, D. S. Oliveira and J. R. Crandall, "FAROS: Illuminating In-memory Injection Attacks via Provenance-Based Whole-System Dynamic Information Flow Tracking," in *2018 48th Annual*

IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxembourg City, 2018.

[12] influxData, Inc, "InfluxDB: Purpose-Built Open Source Time Series Database | InfluxData," 06 01 2020. [Online]. Available: https://www.influxdata.com/. [Accessed 06 01 2020].

[13] Prometheus.io, "Prometheus - Monitoring system & time series database," 06 01 2020. [Online]. Available: https://prometheus.io/. [Accessed 03 01 2020].

[14] The OpenTDSB Authors, "OpenTSDB - A Distributed, Scalable Monitoring System," 03 01 2020. [Online]. Available: http://opentsdb.net/. [Accessed 03 01 2020].

[15] "quickavro · PyPI," 11 01 2020. [Online]. Available: https://pypi.org/project/quickavro/. [Accessed 11 01 2020].

APPENDIX A Common Data Model Version 0.5 Schema Highlights

```
// fixed field types
fixed bits16(2); // generic 16-bit type
fixed uuid(16); // 128-bit UUID
enum SubjectType {
    SUBJECT_PROCESS,
    SUBJECT_THREAD,
    SUBJECT_FUNCTION,
    SUBJECT_BLOCK
}

enum SourceType {
    SOURCE_LINUX_AUDIT,
    SOURCE_OPEN_BSM
    // TODO: add others
}

record Subject {
    uuid id;
    int pid;
    string cmdLine;
    string envVars;
    SubjectType type;
    SourceType source;
}

record HasParent{
    Subject child;
    Subject parent;
}

record Host {
    string ipAddr;
    union {null, string} optionalHostname;
}

record RunsOn {
    Subject subject;
    Host host;
}

record Timestamp {
    long sec;
    long nsec;
}

enum EventType {
    EVENT_ACCEPT,
    EVENT_BIND,
    EVENT_CHANGE_PRINCIPAL,
    EVENT_CHECK_FILE_ATTRIBUTES,
    EVENT_CLOSE,
    EVENT_CONNECT,
    EVENT_CREATE_OBJECT,
    EVENT_CREATE_THREAD,
    EVENT_EXECUTE,
    EVENT_FORK,
    EVENT_LINK,
    EVENT_MMAP,
    EVENT_MODIFY_FILE_ATTRIBUTES,
    EVENT_MPROTECT,
    EVENT_OPEN,
    EVENT_READ,
    EVENT_SIGNAL,
    EVENT_TRUNCATE,
    EVENT_UNLINK,
    EVENT_WAIT,
    EVENT_WRITE
}

record Event {
    uuid id;
    Timestamp timestamp;
    EventType type;
    SourceType source;
    long sequence;
    union {null, long} optionalLocation;
    union {null, long} optionalSize;
}

record IsGeneratedBy {
    Event event;
    Subject subject;
}

record Memory {
    long pageNum;
    long address;
}

record File {
    string url;
    long version;
    long size;
}

record NetFlow {
    Host src;
    int srcPort;
    Host dest;
    int destPort;
}

record Object {
    uuid id;
    bits16 permission;
    Timestamp timestamp;
    SourceType source;
    union {Memory, File, NetFlow} subclass;
}

record ResidesOn {
    Object object;
    Host host;
}

record IsPartOf {
    Object subObject;
    Object superObject;
}
```

```
record Value {
    bytes value;
    long size;
}

record EventAffects {
    Event event;
    union {Subject, Object, Value} entity;
}

record AffectsEvent {
    union {Subject, Object, Value} entity;
    Event event;

}
```

APPENDIX B ORange – the Infrastructure Framework Decision Revisited

B.1 Preamble

The TC Open Range (ORange) effort was the final of a series of analyses performed by BBN in its role as integrator (TA3) for the TC project. It is a trade-off analysis between a home-grown **libvirt**-based system and toolsets, such as OpenStack, which provide management and oversight tools for test ranges such as TC's experimental range. Since TC was an active and ongoing effort, the earlier analyses had to consider not only the value of introducing a new toolset, but also the disruptive impact of changing tools, procedures, etc., in the context of the ongoing TC experiments.

Being the final evaluation, the ORange effort ignores the impact of disrupting ongoing experiments and instead focuses on two questions:

- If one were to implement the TC range in 2019, what approach would be the approach to take? E.g., home-grown or off the shelf (commercial or open source) toolset?
- As a result of that analysis, what guidance would the ORange team provide to any other project seeking to set up a test range today?

The first question considers the same concerns as the earlier analyses, specifically does a toolset now exist that would be a better choice than TC's home-grown system, but does not consider the cost of moving from one to the other.

The second question is meant to provide a more general roadmap for any project considering building in-house range. Its answer should provide information on the toolsets that were surveyed, pointers and considerations to include should a new toolset arise, which should aid that team in choosing a toolset. These findings are summarized in the "Getting Started" appendix (APPENDIX D).

Both assume a fresh start and do not consider the cost of transitioning. Both consider life cycle costs, e.g., the initial cost of learning the toolset, the cost of introducing new team members to the toolset, the cost of moving the toolset to its next version, etc.

The following philosophy statement highlights both the TC and general requirements and criteria that guided the effort.

B.2 Philosophy

The primary goal of the ORange analysis was to select the best toolset for setting up any test range, using two use cases. One very similar to the TC range and the other a more general test range. The team assumed that the test range should allow for multiple, concurrent experiments operating at different access levels, e.g., fundamental research with non-US persons vs. ITAR research, without allowing leakage between them. The test range can either be in-house, in the cloud (off-site), or a mix of the two (hybrid cloud). The team was looking for a toolset that had a vibrant user community and active development/support teams, since it is understood that complexity and advanced capability often go hand in hand. The team was looking for something easy for an incoming team that was new-to-range-setup to learn and use.

Ongoing ease of use and toolset life-cycle, e.g., ease of adding new capabilities or ease of updating to a new toolset version, were also considered. The overall assumption is that there is no team dedicated to designing, setting up, and running the test range, so the toolset should minimized those costs (and learning curve), so the team has more time available for their primary responsibilities.

Since there are many options for building a test range hosted off-site by an off the shelf vendor (i.e., in the cloud), why even consider building the in-house capability? This analysis of in-house tools is driven by the belief that several aspects of our research necessitate an in-house solution, even if it, in part, interacts with pieces in the cloud (off-site), i.e., a hybrid solution.

Opting for an in-house solution may be a result of aspects of running the experiment that would violate the cloud's terms of service (TOS), i.e., running malware (even if "defanged"), or running software that is too sensitive to risk pushing outside of the enterprise, i.e., ITAR software.

An oft-cited special capability, such as guaranteed separation of experiments due to IP or ITAR considerations, is easily addressed by off the shelf solutions that currently can guarantee separation or even specific locations for hosts, The in-house solution comes into play when the incremental cost of those guarantees exceed the budget.

While cost and ease of use are often cited as reasons for in-house solutions, these arguments are of diminishing strength due to free tier pricing or when considering the true cost of in-house set up.

See the Experimentation and Results section of this appendix for more details. The team did much of its experimentation within the design and architecture of the existing TC test range, which limited some of the experiments the team was able to perform.

B.3 Requirements

While the requirements provide the goal, answering the "recreating the TC range" question should be constrained by only considering solutions that would work for the systems, resources, and limitations that were in place when TC started and should highlight issues and constraints faced in the initial range implementation. Alongside the cognizance of what technology was available at the time of the proposal submission (2015), implementation requirements served as the reason for selecting one software or architecture over another. This section will explore those constraints as based on their impacts in hardware, software, networking, ease of use, experiment isolation, and logistical objectives.

B.3.1. Range Management Software

Ideally, range management software allows the range owner free reign in setting up an experiment and then turning that experiment over to the performers who can run it with minimal or no impact from the management artifacts. In most cases, this results in test machines with at least two network interfaces, one for management, the rest to serve the experiment. Management software can be agent-based or agentless. Agent-based solutions have an agent on each test host, either directly on the host or in a hypervisor (or other overlay) which allows management of the host without impacting the experiment. Examples would be hypervisors, PXE-based controllers, and agents embedded in the OS under test. Agents are usually quite feature-full. Agentless solutions are based on very lightweight presence on the system under test, sometimes as light as an available **ssh** port.

Ideally, the range management should allow for multiple, concurrent experiments, which are isolated from one another.

The requirements of a software and hardware coupling required to service the TC range are outlined in the goals of the TC program. In serving as TA3 performer on the program, responsibilities largely included the facilitation of cross-TA collaboration. The TA3 performer also had the responsibility of managing individual TA experiments and providing access to relevant information required for evaluating TA1 and TA2 software. As a result, software selected for the TC range must support the ability to work with a virtual machine hypervisor, or other agent, for managing performer hosts and running experiments. Additionally, a system configuration management tool for managing a number of running hosted systems with specific software and data is necessary. A way of configuring the network topology dynamically to adapt to a changing number of hosts with specific network access requirements will enable networking and connectivity, which is discussed in a later section.

In the original TC range implementation, hosts were provisioned using the SaltStack (Salt) platform, a Python-based open source configuration management and remote execution application. The procedure for executing code and thus configuring the range is a multi-step process. It consists of defining an ‘experiment’ file from the Salt “Master” with a set of machines to operate on, and then executing a script on the "Master" which runs code on each system (called a Salt “Minion”).

This approach required that each host run the Salt Minion agent software, such that the system is provisioned to receive instructions from the Salt Master. This adds a level of complexity in managing hosts, as it requires all hosts to have a working Salt Minion agent software installed. This worked well for our “base case” of a specific system or host, but introduced challenges when provisioning additional systems or platforms. Salt was the ideal tool for configuration management and remote execution at scale, but only shows its strength in purpose built systems and workflows.

In TC’s use of Salt, a number of different hosts were running scripts determined by the Master. This meant that on occasion, the resulting behavior would differ from Minion to Minion. This required attention to each run of Master scripts, so as to catch any errors and output for debugging runtime errors.

B.3.2. Range Hardware

As proposed, the TC architecture necessitated the ability to install hosts on both virtualized hosts and “bare metal” hardware – a distinct, single piece of computing equipment. Performers on the program presented work on Linux-based, Windows, and even mobile operating systems. Some of the hosts could run in a virtualized environment via the KVM virtualization technology. Others, like one performer’s Android OS, required that the software run directly on the target device without virtualization. Across multiple engagements, a performer might need to have access to many different hosts – or just a few.

In order to provide access to both virtualized and physical hosts, two separate networks were configured for controlling and for running experiment hosts. Using **libvirt** and KVM, the performers' experimental hosts were able to be run on the TC systems as virtual hosts. The key advantage of this functionality is that it allowed BBN to maintain control of the hardware while allowing individuals to manage hosts as needed.

The ranges consisted of several types of components:

- The bulk of the range consisted of commodity rackmount Dell R210 or R230 servers with 32GB memory, up to 5TB (4+1) HDD, and 2 or 3 network interfaces; these served as TA1 and TA3 (Kafka and test servers) hosts;
- A few Dell 620 or 710 servers; these served as build and service hosts, such storage, git/wiki, and Jenkins;
- Four Dell C6000 servers, each with 4 internal blades; these served as TA2 hosts and occasionally as TA1 hosts when desired by TA1 performers for higher performance platforms;
- A few Cisco routers and switches; these handled the necessary internal routing architecture.

In addition to the above, Kudu Dynamics (TA5.2) provided a complete LARIAT setup to provide traffic generation. This setup consisted of several machines plus routers. As TA5.2 configured and operated LARIAT, that portion of the test network was treated as a black box by BBN.

B.3.3. Network

From the point of a DARPA experiment and in particular from TC, there were a few requirements that the network needed to support. First, the range needed to support separation (physical or logical or both) between range control and experimental data collection, i.e., these two streams could not interfere with each other. While TC used VLANs to separate control from experimental data, the two streams did use the same physical switch, and this possibly ran the risk of having large loads cause non-purely experimental overload effects on the hardware. The overall load in TC did not approach this level, so it was not deemed necessary to provide complete physical separation of flows, but in some experiments, this may be a concern.

Second, the network needed to be able to isolate portions of itself for TOS or safety reasons. In the case of TC, the TA5.2 performer ran artificial (in that the 'exploits' used as the initial entry vector were old, known vulnerabilities fixed in current versions of the software or vulnerabilities intentionally added to existing software) attacks upon the target systems, but other DARPA experiments could easily require attacks or activities that would be imprudent to run on a network connected live to the Internet or would violate the terms of service of the portion of a hybrid cloud setup. Isolation could allow such experiments to be conducted in greater safety and adherence to usage agreements.

Finally, the TC network in particular provided a simulated Internet with local copies of well-known sites (such as Google or YouTube) in order to provide realistic appearing network traffic without having to impact or reveal the experiment to the actual real-world sites. In order to do this, the network was configured to be able to switch from a local DNS server with the default route pointing to the LARIAT gateway server (for experiments) over to the actual Internet during setup and configuration time, allowing external performers to be able to access and test on the range from their remote sites and also access public repositories and software sites and vice versa. This sort of flexibility greatly increased the usability of the range for program collaborators.

B.3.4. Ease of Use

A major consideration of any tool set/system choice is the set of life cycle costs, including the initial cost of learning the toolset, the cost of introducing new team members to the toolset, the cost of moving the toolset to its next version, etc. Some costs may be more important than others due to local conditions (e.g., a rapid initial setup may be required due program schedule), but otherwise, the cost of ownership – including the operations and maintenance phase - is likely to be the driver for this factor.

Designing, setting up, managing, and updating the test range should be accessible to most engineers with minimal training. Having only a few or even one engineer with the required background and knowledge can be a major blow to a project if a disaster strikes. As a corollary, the toolset should include resources (FAQ, user community, etc.) to handle both initial and more complex questions, as well as reference resources for all phases of setup and usage. A toolset that has a vibrant user community and active development/support teams is highly desirable, as development tasks can be shared with that community rather than all falling to the in-house team.

B.3.5. Experiment Isolation

While TC was classified as fundamental research, thus allowing relatively unfettered access for non-US persons from TC collaborators to access the range with only minimal hindrances and requiring no isolation, it is likely that future types of these experiments may have EAR or ITAR restrictions placed upon some or all of the software or components. Thus, the test range should allow for multiple, concurrent experiments, operating at different access levels without allowing leakage between them.

The operation of a multilevel classified network is beyond the scope of this work as the tools and requirements are circumscribed by a different set of security processes and authorities, so the only consideration for the purposes of this work are restrictions on EAR and ITAR separation. Thus, the particular separation mechanisms may need to be documented and certified in some respect, they do not necessarily adhere to the high levels of assurance in classified settings.

B.3.6. Experiment Logistics

As was seen in TC, experiment logistics are important to experiment-based testing with required metrics. Luckily, however, provision of tools and mechanisms to test ranges is a relatively well-known and solved problem. Among these requirements are:

- Mechanisms for range users to reserve subsets or the entire range for independent testing; this may involve changing from test to development mode as well;
- Control and management of logs and logging parameters, including downloads, log rotation and purging;
- Monitoring and management of host and range infrastructure components, including live performance data as well as records for post-test analysis.

B.4 Research and Experimentation Results

The ORange effort started with the list of TC-focused requirements for the range, as well as a more general set of requirements, captured in the "Requirements" section of this appendix. The team then surveyed Internet for the existing toolsets, and chose the most promising to experiment with.

B.4.1. Preparations and Toolset Selection

After researching a wide range of toolsets, the team chose to focus on tools that provide Infrastructure as a Service (IaaS)[1] , as they seemed most suited for creating general purpose in-house ranges.

To start, the team started with the following characteristics of ideal IaaS toolset:

- High-level APIs dereference low-level details of machine networking and infrastructure
- Configurable network topology with management plane for starting, stopping, logs, etc.
- A hypervisor runs virtual machines as guests
- Alternatively, services are containerized eliminating overhead of a hypervisor
- Capacity of a container scales with computing load
- Capability for bare metal host while preserving access by the management plane
- Provides additional services such as
  - Virtual-machine disk image library
  - File or object storage
  - Software bundles
- A vibrant user community and active development/support teams
- Easy to learn; easy to maintain; (if not home-grown) easy to upgrade when the product evolved

The team also looked at the components of TC's home-grown **libvirt**- and Salt-based solution. While Salt provides the configuration management tools to create simple cloud quickly, it lacks the tools and capabilities to orchestrate the individually configured hosts into a homogeneous single cloud. That responsibility fell to the home-grown scripts. Every new feature required extensive work to integrate it into the existing tool suite. It lacked the polish and consistency of a more holistically designed and implemented suite.

After a brief survey the team chose three promising suites to experiment with: OpenStack, CloudStack, and OpenNebula. We quickly ran into issues with OpenNebula, however and stopped working on it due to time and resource constraints. This was not surprising since OpenNebula is the least mature of the three technologies.

[1] NIST defines IaaS as "where the consumer is able to **deploy and run arbitrary software**, which can **include operating systems and applications**. The consumer does not manage or control the underlying cloud infrastructure but **has control over operating systems, storage, and deployed applications;** and possibly **limited control of select networking components** (e.g., host firewalls)."

The experimentation approach was straightforward. For each toolset, we performed the following operations:

- Install the minimum toolset required for experimentation.
- Recreate the TC CADETS experiments using the toolset. CADETS is a single host experiment, but required a bare metal install.
- Recreate the TC TRACE experiments using the toolset. TRACE requires a complicated host setup, in multiple steps, for its execution.
- Document everything.

For the sake of expediency, the team chose to stick with TC's existing hypervisor/guest OS model, rather than redesign and rewire the range. The primary issue was how the interfaces were configured in the range and the inability of the existing management tools to accommodate any changes. In the TC range, the management interface was connected to the hypervisor and the test range interface was connected to guest OS. While many of the IDRAC interfaces were wired up, they were not used.

The next sections provide an overview of the experimentation with each solution and describe the results of the research and experimentation process anecdotally for each of the solutions. The final section provides a summary and recommendations.

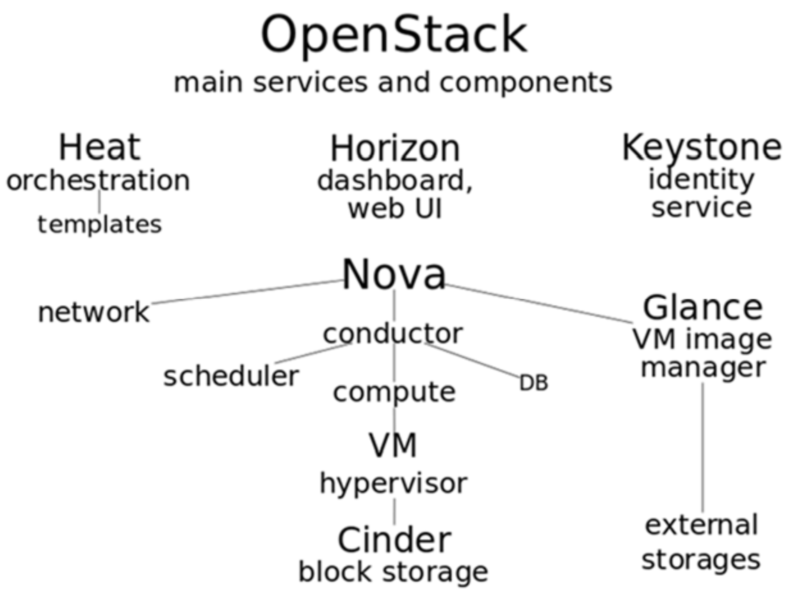
B.4.2. OpenStack

Introduction

OpenStack (https://www.openstack.org/) is an open-source software platform for cloud computing, offering a broad suite of modular service components covering compute, storage, network and management functions:

- Bare metal, virtual machine, containerized and lambda compute resources, plus elastic map reduce
- Physical, VLAN, and virtualized network segments, virtual or (some models of) physical switches and routers plus DNS
- Block, image, object and file storage plus database
- Management dashboard, telemetry/usage metering, user identity, key management. A dashboard is also available to the users of the range.
- Messaging, workflow, application and container orchestration, alarm/event analysis and rule-based actions
- Drivers to manage Amazon EC2 resources (instances, images, volumes and networks) for building hybrid cloud environments
- Plugins to integrate other services with the OpenStack compute, networking and storage services, such as VMware and Hadoop

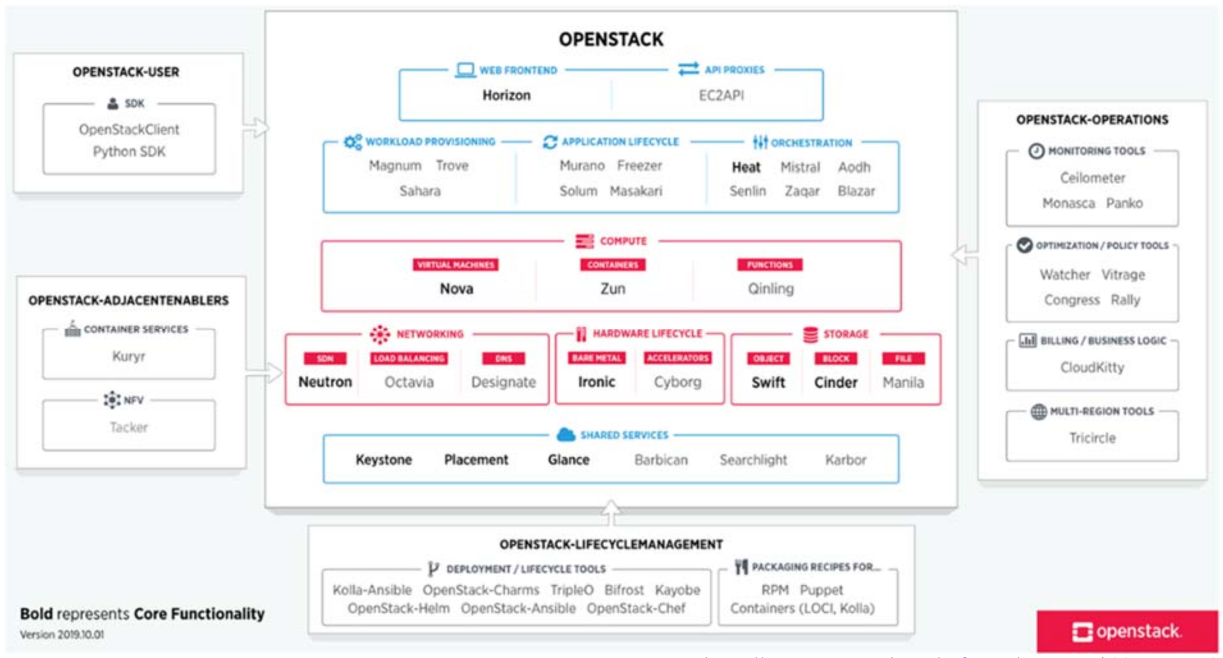
Unlike the other candidates, OpenStack is a federation of separately developed components, sharing common services and capabilities. This would allow selecting a subset of components, then pick up others as requirements change. OpenStack releases are coordinated across the components, in the hopes of eliminating most integration issues in the field. Figure B-1 shows the main components and each name (e.g., Heat, Nova, Cinder, and Keystone) is a component maintained by its own development team.



Source: Conan at English Wikipedia, downloaded from https://en.wikipedia.org/wiki/OpenStack

Figure B-1. OpenStack Main Components

Figure B-2 shows a more complete set of components, circa 2019, showing the wide range of management capabilities available.



Source: https://www.openstack.org/software/ Accessed 30 January 2020

Figure B-2. A Feature-full OpenStack Instance

In addition, several different toolsets are available for orchestrating deployment of an OpenStack range onto new hardware. These are termed "deployment tools" and often include a means to describe the desired range. These would be used to setup the initial OpenStack management tools and could be used to replicate the existing OpenStack and range to a new set of hardware.

OpenStack launched in 2010 and is under active development.

Initial impressions

OpenStack's main advantage, as of late 2019, as a range management toolset is that the components cover a wide range of cloud environment functionality that includes functions that were specified as requirements of the TC range at the outset of the program, functions that emerged during the course of the program, and functions that could prove useful in the future. In particular, OpenStack services include:

- Common mechanisms for managing the software delivered by performers whether the delivery is bare metal machine images, virtual machine images, or containers
- Dynamic network configuration, management and virtualization
- Granting performers privileges to reinstall or restart their services through a common dashboard
- Range-wide management and telemetry

The requirements for managing a range for all of the TC performers, with their different timetables, requirements and development methodologies, meant that the management scripts BBN wrote had to be augmented and modified throughout the program. With the benefit of hindsight, much of that work appears to have been implemented in OpenStack in the last five years, and so the decision of which platform to choose will likely be different from that made at the outset of the TC program.

OpenStack's main disadvantages are a consequence of that wide coverage and flexibility. Common mechanisms are used for configuration and management of each service, and components make use of services provided by other components. However, with such a diverse set of partially optional services that can be separately configured, the range maintenance staff will have a great deal of material to absorb, and to keep current on as the project evolves. Given OpenStack's development history, each release is maintained for 18 months, so plans to incorporate regular updates to the platform and tools into the range would be an important aspect to consider when planning the project and the engagements.

Vision

Here is a concept of how an experimentation range shared by multiple performers could make use of OpenStack.

- The test range administrators install OpenStack on the range and assign machine roles, comprising a management host, a large number of compute hosts, one or more storage service hosts, and several switches. Few manual steps are involved at machine consoles.
- The test range administrators create users and permissions, assign keying, and establish resource policies and limits. They configure the overall network including access to the outside internet.
- Each performer is given access to a slice of the range: a number of compute hosts, a number of virtual network segments, and a slice of block storage. Each performer

receives pointers to appropriate OpenStack documents and forums, which augment the test range administrator's support.

- Performers are able to allocate virtual and bare metal hosts, machine images, virtual disks, and assign network addresses within their allocated spaces. They can connect to the overall network and also have private network segments. Permissions prevent performers from altering resources allocated to other performers.
- When the time comes for an engagement, the test team ensures that the systems under test are properly configured, installed with specified images from the image storage service, and are accessible on the network. OpenStack telemetry services can provide range-wide metrics about machine and network usage.
- Test performers can also use the same configuration files as the shared range to create their own matching OpenStack test ranges at their facilities.

Experiments

We conducted a series of experiments installing OpenStack to test our assumptions about its suitability.

DevStack

The initial approach for installing OpenStack was to use the DevStack[2] deployment or bootstrapping toolkit. We decided to use this toolkit after an initial survey for OpenStack bootstrapping solutions. DevStack development started in 2011 and is considered feature rich, robust, and mature. We deployed OpenStack via DevStack on a newly installed Ubuntu VM (which was a guest VM of an existing hypervisor). Using DevStack to install OpenStack was very simple. We first cloned the GitHub repo and ran the main script (per the README). Once the script finished running, we were able to navigate to the Horizon UI.

Neither OpenStack nor DevStack provided "create a cloud" wizard, so we returned to the OpenStack pages to find it.

With rudimentary training, we tried probing the network to discover other hosts. Unfortunately, this is where progress stopped. The main issue was the the DevStack install was not able to probe the network for additional assets such as switches, iDRAC's, or even other reachable servers.

Returning to the OpenStack pages, we discovered that DevStack is more of an install acid test than a deployment tool. The network discovery problem coupled with the fact that DevStack is not mentioned in the list of deployment tools helped us reach the conclusion that another deployment tool set should be used.

Deployment and Lifecycle Management Tools

OpenStack hosts a variety of deployment tools[3] for deploying OpenStack onto fresh hardware and managing the deployment. Using such a framework, redeploying onto a fresh range or adding and deleting hosts can be automated. Different groups have assembled these frameworks based on existing tools they were familiar with such as Ansible, Chef or Helm.

[2] https://docs.openstack.org/devstack/latest/
[3] https://www.openstack.org/software/project-navigator/deployment-tools

Surveying the list of tools, the two we examined more closely were TripleO and Kayobe.

TripleO

TripleO[4] (or OpenStack On OpenStack) installs a minimal OpenStack configuration and uses that to deploy and manage the full desired OpenStack deployment of machines and other resources that users will see. These two configurations are independent of one another.

The main advantage we see of this approach is that that minimal management cloud uses the same scripting languages and configuration options as the full OpenStack deployment, which could reduce the learning curve for initial deployment.

Kayobe

Kayobe[5] offers deployment of a containerized OpenStack service onto bare metal hosts. In contrast with TripleO's use of a complete, minimal OpenStack cloud as a deployment tool, Kayobe uses a small selection of OpenStack components for its deployment environment. Specifically, OpenStack **bifrost** discovers and provisions the cloud, OpenStack **kolla** builds container images for OpenStack services, and OpenStack **kolla-ansible** provides the deployment and upgrade of containerized OpenStack services. This dependence on Kolla and Bifrost is the basis for the tool's name.

All configuration takes place in a small tree of files for the Ansible agentless management tool. These files describe the initial Kayobe deployment machines, the initial physical and virtual network configurations, the broad categories of machines that Kayobe will install onto, and host discovery.

The main advantages we saw of this approach are that there are fewer OpenStack services installed as the deployment tool, potentially requiring less overhead and providing more flexibility.

Bare Metal

The step-by-step walkthrough[6] for installing Kayobe indicated that it would work best as a bare metal install rather than inside a VM. Late in our experimentation, the team performed a bare metal OpenStack instance with about a half of day extra engineering to get network access for the installation and again for the OpenStack instance. We attempted an initial experiment of installing the seed VM directly onto one of the hosts inside the TC range. However, configuring the network interfaces of the bare host to match the VLAN configurations used by the rest of the TC range was challenging, and it was decided that to make changing environments easier as we tested it would be best to switch to a virtualized installation.

While the bare metal OpenStack installation was successful, we backed away from the bare metal approach to avoid additional networking surprises.

[4] https://docs.openstack.org/tripleo-docs/latest/
[5] https://docs.openstack.org/kayobe/latest/
[6] https://github.com/stackhpc/a-universe-from-nothing/

In VM

After restoring the TC range machine under test to its default, VM host configuration and reinstalling the stock virtual machine, we went through the steps to install Kayobe on this virtual host. The initial step of installing the seed went smoothly. At that point our task was to specify the range configuration we wanted, enumerating the hosts or initiating a hardware discovery stage making use of the OpenStack Ironic service. At this step, two factors presented challenges: attempting to automatically discover the TC VMs running on other TC hosts, and specifying the existing TC network configuration rather than giving control of the network to Kayobe. Both of these would likely be smoother in a case where the Kayobe seed host had direct access to the physical network interfaces and managed DHCP network address assignments on the test range itself.

At this point we decided that we had learned enough about how deployment tools worked through reading and inspection to describe how another project starting fresh might use them.

Return to DevStack

We returned to the DevStack toolkit and installed and manually configured it on four Ubuntu virtual hosts on the TC range.

We were able to use the Horizon dashboard to start and run multiple virtual machines hosted by OpenStack's Nova compute service. (Note that the virtualization service KVM used by OpenStack and other tools supports management of nested virtual machines without additional overhead: see Nested Guests[7])

We were also able to take advantage of the Neutron networking service to configure multiple VXLAN[8] segments over the virtual machines' VLAN-attached virtual network interfaces. One complication was that configuring the same interface for both the external gateway for the OpenStack-host environments and for the Ubuntu virtual host that DevStack ran the management services on meant that we needed to configure two IP addresses on the same interface. While OpenStack supported this, the TC range management environment did not support that configuration directly, and so manually entering the second IP address into the ARP cache on the TC range gateway host was necessary to forward external traffic back to the OpenStack environment.

Finally, we were able to create multiple users with different permissions for the Nova and Neutron services, verifying that we could give different levels of access to different users of the OpenStack range.

DevStack is intended for initial experimentation. Any changes to the host or network configuration of a range require uninstalling and reinstalling the software on all of the hosts. While this does not take much time, it would not be appropriate for a large, shared test range used by multiple performers for extended tests. For this reason, we explored the more complex deployment methods described in the previous section.

[7] https://www.linux-kvm.org/page/Nested_Guests
[8] https://tools.ietf.org/html/rfc7348

Conclusions

OpenStack is a featureful and useful environment for creating and managing a test range, with great potential for cleanly handling the challenges of a large shared test environment with evolving requirements. We were able to demonstrate multiple services such as creating users, setting permissions, defining virtual and physical networks and running VMs within it.

OpenStack is also complex and layered, and it will require substantial training and set up time to create a test range useful for a TC follow-on project. The effort to create a durable and repeatable range will need to be put in at the beginning of a project, with a certain amount of learning and testing for any new feature and some overhead for keeping the range current as OpenStack evolves.

The good news is that resources exist which reduce this effort. No only OpenStack's online documentation and tutorials, but also O'Reilly such as "Learning OpenStack", Alok Shrivastwa, Sunil Sarat, Packt Publishing, 2015.

Overall, OpenStack could be more flexible and powerful at lower overall cost than project-created scripts for a project requiring a multiple-access test range over several engagements over several years.

B.4.3. CloudStack

Apache CloudStack is a self-contained Infrastructure-as-a-Service (IaaS) offering from Apache. CloudStack can be used to turn virtual infrastructure into hosts, networks, and other components that can be managed on-demand through the IaaS's interface. Frequently compared to (and not to be confused with) OpenStack, CloudStack is distinct for its fixed set of functionalities. Rather than do many different things with varying levels of performance, CloudStack aims to do just a few things for its users - and do them well.

Initial Impressions

CloudStack has a few features and traits that made it an attractive option for looking at how one would manage the TC range with the technologies available today. First released as a stable software in 2013, CloudStack is considered mature and fully functioned for its intended use case of enabling users to host a private managed cloud. Development of CloudStack is maintained by the Apache Software Foundation. CloudStack offers straightforward services for managing virtualized hosts, orchestrating network topologies, allocating resources, managing users, and even billing for the amount of time and resources used. The web-based UI is a nice consideration for ease-of-use without the technical learning curves of running specific scripts or executables in a CLI. An attractive and noteworthy feature of CloudStack is its support of multiple hypervisor technologies - meaning that not all hosts in the CloudStack range need to be running KVM.

As shown in Figure B-3, the CloudStack instance can be implemented in such a way that is both simple and has a minimized host requirement.

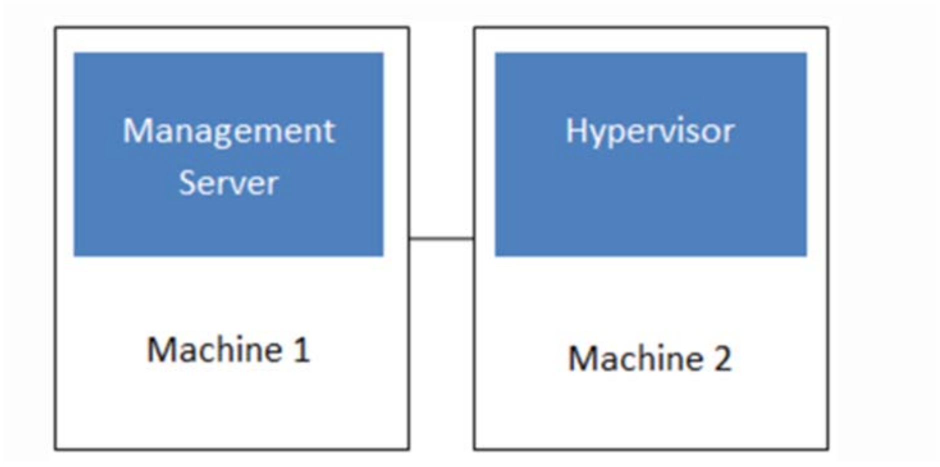


Figure B-3. A Basic CloudStack Deployment

Getting CloudStack setup seemed straightforward, as all necessary software - the CloudStack Management Server, Hypervisor, and CloudStack agent - can be installed on a single host in a basic deployment. We took this approach for our experiment with setting up and using CloudStack on the TC range. A Feature-full OpenStack Instance

Setup Process

The CloudStack documentation offers a quick installation guide that walks through the process to initialize two main components - the CloudStack Management Server and the KVM hypervisor setup. This process is ultimately broken down into the steps of preparing the host operating system with additional configuration and other environment setup. In evaluating CloudStack, we attempted to follow the quick install steps and did not have any issues accessing the web interface.

Barriers

After trying to use the CloudStack user interface to begin testing with the range, it became clear that CloudStack was intended to be used in a very specific way. In order to use platform, you must populate a few CloudStack specific elements. This includes creating a zone, pod, and cluster. In CloudStack infrastructure terminology, a zone is analogous to a single data center. A pod typically represents a rack or row of racks, and a cluster is one or more hosts paired with primary storage. Not understanding the conceptual relationship between these elements in a CloudStack-based range made it difficult to get started with the platform and implementing our ideal infrastructure setup.

Even while navigating some of the CloudStack-specific language, adding the KVM virtual hosts still presented a challenge. Error messages presented in the UI were extremely non-descriptive, and simply indicated some sort of failure occurred in adding a host to be managed. This meant that debugging on the application level had to take place on both the CloudStack Management Server and the virtualized host. While errors like these are expected of any first-time deployment of an application, it was still troubling to try and add the first host and have little context for the failures that we were experiencing.

The failed attempt to integrate CloudStack into the existing TC architecture is a result of both the aforementioned points. The tightly defined architecture combined with shallow error reporting added friction to the installation and setup process. In attempting to leverage CloudStack, levels of reachability from the hosts to the Management Server were limited.

Considerations for Use

A side effect of the simplification introduced by CloudStack is that it requires additional platform-specific knowledge to properly deploy the cloud. For a researcher charged with the task of installing and provisioning CloudStack, there is considerable lead time involved to best understand how one would architect a range to fit the boundaries of CloudStack. Users of the installation, however, should require less training thanks to the simplified and straightforward user interface[9].

The ability to customize CloudStack's internal functionalities is limited. If a project requires an especially unique setup for managing and installing hosts, it may be worth looking at a different IaaS provider which offers the ability to stack custom components together. Thankfully, however, there is generous support for multiple hypervisor platforms and network topologies - assuming you are familiar with the language used by CloudStack.

Conclusions

CloudStack fills the niche of an easily deployed IaaS offering that one can "set and forget." This assumes, however, that the researcher's intended system architecture matches that of CloudStack's prescribed layering of zones, pods, and clusters. The installation is fairly lightweight, requiring just one host running the self-contained Management Server application and the infrastructure itself. Though there is some lead-time required to understand CloudStack, it will still take less time to understand orchestrate than a platform like OpenStack, where there are endless available combinations of features. CloudStack does offer an integrated API, which may prove useful for organizations that are looking to provide custom integrations in the cloud management tool.

B.5 Conclusion and Recommendations

Overall, our recommendation is that more research and effort is needed, but in general, taking the leap and starting with an existing toolset seems to be a better way to handle range management over time. Also, our experience with the as-yet immature OpenNebula echoed earlier issues evaluating OpenStack. It is important to use current tool instances and current front runners in any toolset research.

The home-brew setup is always easier than the learning curve of a new toolset, but the ability to leverage an existing, possibly large, well tested suite of tools will quickly outperform a research team forced to go outside its comfort zone to write and maintain range management tools.

In a short time, we were able to install the toolset, get a range discovered, and begin running on the range using two different toolsets. While we did not reach the stage of replicating any TC test

[9] After our experiments, we discovered "OpenStack for Architects - Second Edition", Michael Solberg and Ben Silverman, Packt Publishing, 2018, which may have simplified things still further.

setup, we were limited by our self-imposed limits and our lack of knowledge. The experiments that did succeed leave little doubt that we could have achieved replication of all TC configurations, as well as some that are beyond TC. For example, bare metal test machine installation and management, accounting systems, enforceable access control, or side by side concurrent test ranges.

Even our multiple restarts provided value. While the restarts meant that we didn't progress as far against our goals, we did increase our knowledge of the toolset under review, which allowed us to move quickly and effectively after restarting. And since we were working with a somewhat (or very) mature tool suite, not fresh code, we could focus in on mapping our requirements onto this tool suite and not contend with bugs and deficiencies in what would otherwise be newly minted code.

Except for the very simplest or short-lived range needs, considering an existing, mature, and well-supported toolset seems the better choice. This is also almost always true even if the toolset is later deprecated or abandoned. Assuming the toolset is up and running at the time of the deprecation or abandonment, our range remains usable as we look for another. In this scenario, if bugs arise as we wait, we can address them ourselves, as the source is available. Since we are part of a large community of teams affected by the abandonment, many people will be facing the same transition need and posting solutions that we can leverage.

APPENDIX C Re-implementing the TC Range, Guide and Instructions

Our preliminary results show that, with more understanding of the tools, either toolset, OpenStack or CloudStack would at least provide the hypervisor and guest OS approach that TC used successfully. After designing the approach, additional testing with a bare metal range (with any of these toolsets) is likely to show this too is possible.

Unanswered questions:

- Do we have a sense of which has the easiest onboarding ramp?
- If you had to pick one, which would it be?

While hand-crafting a solution is at first more accessible that adopting a new toolset, long-term range management and use require more support than a research-focused team is capable of. Taking the time to learn a new toolset and use it, is likely to improve the effectiveness of future TA3 roles as the TA3 team is freed from supporting both the range and the toolset.

C.1 Open Cloud Services and TC requirements

Open cloud suites such as OpenStack and CloudStack provide a wide range of services that can be individually used for cloud management and operation. For the TC range case, many of these open cloud services correspond with problems that the TC range designers solved using other techniques.

A future range management team could consider each of those problems and determine whether an internally developed solution would be more suitable for the purpose, more extensible, or faster to deploy than installing and learning the suite service in that area. Moreover, as the needs for a range evolve, adding new functionality and modifying the behavior of existing tools will require planning. Determining the tradeoffs between modifying an existing service and developing a new one will be an ongoing endeavor.

Here are a range of functions that the TC range implemented in tools or performed by hand to operate the range for testing and engagements:

- Allocating resources on the range: physical machines under test, physical and virtual machines performing analyses, machines for performer access, VLANs, network addresses, etc.
- Configuration management: manual steps, manual recordkeeping, Salt scripts
- Allocating services for support of TC data generation and analysis: event logging and distribution, instantiating the simulated internet
- Managing performer access: account and key distribution
- Managing images: storing and distributing images

Here are the functions of the existing TC infrastructure, as compared to CloudStack and OpenStack, which provide a subset or a superset of the functions for range operation and management on the TC range:

Range function	**TC solution**	**OpenStack service**	**CloudStack function**
Range resource tracking	Excel spreadsheet kept in the GitLab repository, plus the contents of scripts.	The OpenStack Placement service tracks resource allocations: compute hosts with available CPUs, memory, disk storage; network addresses; and so forth. Other services check with Placement to find available resources and register allocations as they occur.	CloudStack provides a list of all defined available resources in the landing page of the application. In addition to a graphical display of available allocations, the CloudStack Usage Engine provides user metering to give a real-time look at resource usage.
Compute instance (VM) provisioning	Manually with the Excel spreadsheet and scripts.	OpenStack Nova provisions and installs virtual machines on compute hosts, maps physical and virtual peripherals and devices to VMs, and tracks the state of virtual machines.	In CloudStack, there is strong support for VM templating and ISO management. Built in to the platform is the ability for a user to select from a number of pre-configured templates and system images. The setup wizard to add and provision a VM gives the user a number of options when instantiating a new or existing VM image.

Range function	**TC solution**	**OpenStack service**	**CloudStack function**
Network management	Embedded in scripts and data files used to configure the machines and the DHCP server, and in the fixed VLAN IDs configured within the scripts. The DHCP server also has some components which are managed by a web service which is accessible to all users of the range.	OpenStack Neutron manages allocation of network interfaces and VLANs and managing DHCP addresses. It can perform virtual switching and routing for the range. It also handles DNS resolution within the range. OpenStack cannot configure the current range's switches, so they would be managed as fixed resources by Neutron.	CloudStack natively provides support for software-based network management, as well as configuring VLANs. The "direct attached" IP functionality allows for further configuration of specific virtual architectures as needed.
Bare metal install (physical host) provisioning	TC range machines are installed using PXE boot directly or by manual installation from DVD-ROM.	OpenStack Ironic handles installing machine images on bare metal hosts. It wrangles PXE, DHCP, network bootstrap, IPMI, TFTP, etc. to install a fresh image on a machine and reboot with the fresh image. Ironic configuration files can either specify each machine or can invoke auto-discovery to catalog and populate the nodes.	There exists unofficial support for manually provisioning bare metal hosts. By installing a PXE server, DHCP server, and an instance of the additional lightweight Linux system PING, one can have CloudStack provision this host. This is not officially documented in the CloudStack manual, but is referenced in content from the Linux Foundation and in the CloudStack Confluence site.

Range function	TC solution	OpenStack service	CloudStack function
Resource reservation	Physical/VM allocation is currently tracked on an Excel spreadsheet.	OpenStack Blazar provides users the ability to allocate current and request future OpenStack resources. Currently these are limited to physical and virtual machines and floating IP addresses; plans are to extend that to Heat stacks, so that all resources requested by a particular Heat script are reserved for when the stack is run at a specified time.	CloudStack users have the ability to indicate use of a system out of systems available. Additionally, the usage metering features of CloudStack allow for tracking of an individual user's time.
Orchestration	TC uses Salt to run remote scripts and commands on physical and virtual machines, and resource provisioning and allocation is handled by configuration files and by referencing the Excel spreadsheet that defines the current state of the range.	OpenStack Heat orchestrates actions to take using the other OpenStack services. A Heat script defines a "stack" of resources, pre-existing or not, that a job needs and directs the other OpenStack services to provision, allocate and invoke them. It can also invoke shell, Ansible/Puppet/Salt scripts, etc. to handle on-machine configuration. Ansible and Heat are well integrated.	In the web-based user interface, CloudStack administrators can manage and control groups of hosts or individual hosts. There does not exist a native scripting or "bulk run" code execution feature in CloudStack, but one can still use traditional methods of such configuration such as with Salt.

Range function	**TC solution**	**OpenStack service**	**CloudStack function**
Metrics and monitoring	TC uses Apache Kafka, Prometheus and Grafana directly to gather metrics for analysis.	Openstack Monasca provides monitoring-as-a-service. It wraps an instance of Apache Kafka and Prometheus. Monasca ties into the Horizon web UI dashboard as well as supporting Grafana integration, and uses agents deployed on each compute instance as well as gathering metrics from other OpenStack components. A future range operator could use the existing metrics gathered by Monasca and also plug extra data gathering routines into the Monasca agents.	The CloudStack VM Sync technology provides notifications and monitoring of the state of all VMs. This is in addition to CloudStack's Alerts and Notifications feature set, which allows for notifications via the API or over email. Notifications can be sent when certain resource thresholds are met or if a machine failure occurs.

Range function	TC solution	OpenStack service	CloudStack function
Identity	TC uses access to the GitLab server and SSH keys for access on a per-user basis, and shared accounts on the virtual machines. LDAP was also used for credential management and access control for some services.	OpenStack Keystone provides user identification, authentication and authorization. Keystone allows users and applications to authenticate to keystone, and allows privileges to be delegated. Authentication can be connected to LDAP or other single sign-on services. User identification is used by the various OpenStack services to check for permissions on being able to see, modify or execute their various attributes and functions, and allows administration of access to shared resources to be handled throughout OpenStack and in client systems via the OpenStack Identity APIs.	LDAP integration is supported for authentication, in addition to roles defined for user access to APIs and the web interface. Account-based resource isolation is offered such that memory, CPU, network, and storage resources are available and offered to specific users only.
Key management	Keys are manually pushed to services as needed or as updated via scripts.	OpenStack Barbican provides storage, provisioning and management of secrets such as passwords, certificates and keys. Barbican provides keys to OpenStack services that need them for data encryption, key distribution to VMs, and data signing and signature verification.	There is native support for storing user data when deploying a new virtual machine. This can include adding keys to a template when deploying to the cloud.

Range function	**TC solution**	**OpenStack service**	**CloudStack function**
Network block storage	VMs do not mount disks from a central server.	OpenStack Cinder provides block storage for guest instances, allowing them to mount disks from a central store if so configured.	A central server can contain a collection of VM images which can then be used to provision new hosts. Primary and secondary storage can be accessed through the Network File System (NFS) protocol.
VM/physical machine image storage	GitLab acts as the software repository. Images are stored on the TC development host. The code for provisioning base images is stored in scripts which are in the repos.	OpenStack Glance stores machine images for use by Nova, Ironic, and other services, matches requests to images based on machine architectures and other parameters, and verifies signatures on signed images. Uploading images to Glance makes them available to install throughout the range, subject to permissions. Glance can be configured to store and pull images from Cinder block storage, Swift object storage, or a range of other options including the Ceph distributed storage service, HTTP, or local storage.	Templates are a key value-add provided by CloudStack. A template is a virtual disk image that includes an operating system, optional additional software, and settings such as access control. Users are able to upload and manage these templates in a central location. CloudStack handles ensuring that these images are made available on the server as needed.

Range function	TC solution	OpenStack service	CloudStack function
Web dashboard	Grafana displays the state of hosts and services based on metrics collection by Prometheus. Range control is done through command line scripts.	OpenStack Horizon provides a view of the current state of the OpenStack range and services, and controls for altering that state. Horizon respects permissions and allows both administrators to manage the entire range while users can see and affect only the sections that the user has permission to.	The web interface is the home of all CloudStack functionalities. Users can engage and use any and all of the CloudStack features through this portal. An HTTP API is also available to perform many of the same actions.

C.2 OpenStack Services for functions not provided by TC:

OpenStack service	Function	
Swift	Object store	Swift is a highly available, distributed, eventually consistent object/blob store. Swift storage can be distributed throughout the OpenStack cloud on any managed machine with disk space, and Swift will update and synchronize data as needed between repositories. Provides storage to services such as Glance (image repository).
Magnum	Container support (via an orchestration engine)	Magnum makes containers available to OpenStack through container orchestration engines such as Kubernetes or Docker Swarm. Installs instances of an orchestration engine on one or more Nova instances, and controls containers through commands to the orchestration engine.
Zun	Container support (individually managed)	Zun runs containers on OpenStack compute nodes directly, provisioning them on par with VM instances. Magnum and Zun are different approaches toward the same goal.

Tacker	Network function virtualization	Tacker implements the ETSI standards for NFV, a technology aimed at mobile network operators for virtualizing their switching and network management functions. Use in a test range could start with emulated network routers and go all the way to a virtual 4G or 5G mobile network.
Qinling	Lambda function support	Qinling provides function-as-a-service within OpenStack, enabling scalable serverless execution of code similar to the AWS Lambda service.
Octavia	Load balancing	Octavia provides automatic provisioning and scaling of services, allowing a large service or application to be configured to scale up or down its allocations of virtual machines and other resources according to the load on the service.
Congress	Policy enforcement	Congress provides a policy engine for specifying declarative rules regarding the correct state of the OpenStack services and actions to be taken if the system is in violation of those rules.
Freezer	Backup, restore, and disaster recovery	Freezer provides incremental, snapshotted and complete backup of OpenStack instances. Can be orchestrated to coordinate backups and restores of groups of machines.
Manila	File storage	Manila provides remote file access to instances, configurable within OpenStack with access control and snapshots, backed by a wide range of storage backend options.

APPENDIX D Getting Started – A General Guide for a Team Planning to Set Up a Test Range

The general insights and recommendations to a team getting started closely follow the conclusions shown in the Conclusions and Recommendations section, Appendix B.5.

Any team starting down this path must start with a careful and objective assessment of their range requirements, as did the TC team in 2016. Our observations, in hindsight, is that certain requirements should trigger a very strong bias towards researching and selecting an existing infrastructure toolset over a home-brew solution. Our recommendation is that except for the very simplest or short-lived range needs, considering an existing, mature, and well-supported toolset is the better choice. This is even true if some of the test range will be hosted in-house, as hybrid range management is part of all of the noteworthy toolsets available today.

Triggers for a strong bias for an existing infrastructure toolset include:

- Range will be up for more than six months
- Range will be used by collaborators from outside BBN
- Range will require per-experiment configuration and there will be three or more such experiments
- Range will require concurrently isolated experiments

If any of the listed triggers apply to your planned range setup, we recommend taking the leap and starting with an existing toolset. Do not be swayed by the ease of the first setup using a home-brew solution. Range management complexity does not arise from the first configuration, but the switch to the second or third does. Optimizations made for the earlier configurations will almost always come back to haunt (and restrict) later ones.

Toolset deprecation or abandonment should not be a deterrent, assuming the team selects a mature, actively used, and well-supported toolset. This implies that the toolset is successfully managing the range at the time of it abandonment. The range remains usable as you look for another. If bugs arise as you search for a replacement, fix them. This is no more costly than the home-brew alternative. Since there were many users at the time of abandonment, yours is not the only team in this predicament and fixes should appear from others as well.

The home-brew setup always seems easier than the learning curve of a new toolset, but taking the time to learn an existing toolset quickly returns the ability to leverage an existing, possibly large, well tested suite of tools that will quickly outperform a research team forced to go outside its comfort zone to write and maintain range management tools.

Each of toolsets has simple installation procedures and most have tutorials for installing the toolset and building a range.

In a short time, we were able to install the toolset, get a range discovered, and begin running on the range using three different toolsets, OpenStack, CloudStack, and OpenNebula. See APPENDIX B for our experiences and results. We could not have achieved this progress writing our own home-brew scripts. No doubt similar to your situation, our skills lie in designing, running, and evaluating experiments, not in building range management software.

Other advantages of choosing an existing toolset include:

- Inter-component communication is built into the toolset, using several well-known packages such as Kafka, Prometheus, and Grafana, as well as REST, etc.
- Support for virtual machine and bare metal installations
- Integrated dashboard for all aspects of range management
- Integrated range configuration (often called orchestration)
- User accounting, user- and role-based access control
- Machine image management system, including mechanisms for managing the software delivered by performers whether the delivery is bare metal machine images, virtual machine images, or containers
- Logging, metrics, monitoring, and reporting
- Test machine telemetry
- Dynamic network configuration, management, and virtualization
- Resource reservations
- Support for agent-light and agent-based host management
- Network block storage (so test systems can load remote disks from a central storage server)

While any of these can eventually be successfully home-brewed (as some were with TC), the need usually arises when there are experiments to be set up and run.

Another suggestion for future teams is to take full advantage of existing documentation, tutorials, and resources when considering each candidate toolset. Review the toolset's online documentation and tutorials. Check O'Reilly and other publishers for similar resources. For example, we would have had much better success had we started with "Learning OpenStack", Alok Shrivastwa, Sunil Sarat, Packt Publishing, 2015 or "OpenStack for Architects - Second Edition", Michael Solberg and Ben Silverman, Packt Publishing, 2018. These resources go a long way towards reducing what may appear to a daunting learning curve. An existing document set is another powerful advantage of an off the shelf tool, as the range management team is able to off-load user training to these documents.

As more teams consider and adopt off the shelf (commercial or open source) toolsets, our comfort will increase and our abilities to support complex test ranges will improve without requiring a huge staff to support them.

APPENDIX E Engagement #1 Infrastructure

Name	HW Type	Desired Physical Memory	Disk	Function
devel	R710	60 GiB	2 x 1 TB	TA3 development
mit-build	R710	12 GiB	2 x 1 TB	MIT #1 and #2
gw	R210 II	4 GiB	1 x 1 TB	Gateway
openstack	R210 II	8 GiB	1 x 4 TB, 1 x 1 TB	OpenStack controller
ch01	R210 II	32 GiB	2 x 4 TB	BAE (TA1) #1
ch02	R210 II	32 GiB	2 x 4 TB	BAE (TA1) #2
ch03	R210 II	32 GiB	2 x 4 TB	SRI #1
ch04	R210 II	32 GiB	2 x 4 TB	SRI #2
ch05	R210 II	32 GiB	2 x 4 TB	UNM #1
ch06	R210 II	32 GiB	2 x 4 TB	UNM #2
ch07	R210 II	32 GiB	2 x 4 TB	Five Directions #1
ch08	R210 II	32 GiB	2 x 4 TB	Five Directions #2
ch09	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #1
ch10	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #2
ch11	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #3
ch12	R210 II	16 GiB	2 x 4 TB	Kafka + Zookeeper #4
ch13	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #5
ch14	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #6
ch15	R210 II	32 GiB	2 x 4 TB	Hot spare for TA1s
ch16	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Hot spare for Kafka/ZK
ch17	R210 II	8 GiB	2 x 1 TB	
ch18	R210 II	8 GiB	2 x 1 TB	
ch19	R210 II	8 GiB	2 x 1 TB	Salt-Master
ch20	R210 II	8 GiB	2 x 1 TB	Salt-Master
ch21	R210 II	8 GiB	2 x 1 TB	Salt-Master
ch22	R210 II	32 GiB	2 x 1 TB	THEIA-K #1
ch23	R210 II	32 GiB	2 x 1 TB	THEIA-K #2
ch24	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #7
ch25	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #8
ch26	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #9
ch27	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #10
ch28	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #11
ch29	R210 II	16 GiB	1 x 4 TB, 1 x 1 TB	Kafka + Zookeeper #12
ch30	R210 II	8 GiB	2 x 1 TB	Test Clients
ch31	R210 II	8 GiB	2 x 1 TB	Test Clients
ch32	R210 II	8 GiB	2 x 1 TB	Test Clients
ch33	R210 II	8 GiB	2 x 1 TB	TA3 Services for Testing
ch34	R210 II	8 GiB	2 x 1 TB	Test Clients
ch35-38	C6000	4 x 192 GiB	TBD	GATech #1 and #2
ch39-42	C6000	4 x 192 GiB	TBD	BAE (TA2)
ch43-46	C6000	4 x 192 GiB	TBD	Galois
ch47-50	C6000	4 x 192 GiB	TBD	IBM

APPENDIX F Engagement #5 Infrastructure

Name	Type	Mem	Disk	Function
Infrastructure Machines				
devel	R710	58 GiB	2 x 4 TB, 1 x 1 TB	TA3 development
devel2	R710	58 GiB	2 x 4 TB	Load balance services
gw	R210 II	4 GiB	1 x 1 TB	Gateway
coverity	R210 II	8 GiB	1 x 1 TB	Coverity
Production (Engagement) Range				
TA1s				
CADETS				
ch01	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	CADETS #1
ch25	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	CADETS #2
ch56	R230	32 GiB	2 x 4 TB	CADETS #3
ch03	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	CADETS hypervisor testing
ch71	R620	16 GiB	1 x 500 GB, 1 x 1.5 TB	CADETS PVM
Clearscope				
ch64	R230	32 GiB	2 x 4 TB	Clearscope #1
ch04	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Clearscope #2
ch58	R230	32 GiB	2 x 4 TB	Clearscope #3
webcam1	R200	8 GiB	1 x 1 TB	Webcam for phone
FiveDirections				
ch05	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	FiveDirections #1
ch06	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	FiveDirections #2
ch59	R230	32 GiB	2 x 4 TB	FiveDirections #3
ch08	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	FiveDirections translation
ch73	R230	32 GiB	2 x 4 TB	FiveDirections translation #2
MARPLE				
ch16	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	MARPLE (TA1) #1
ch17	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	MARPLE (TA1) #2
ch74	R230	32 GiB	2 x 4 TB	MARPLE (TA1) #3
THEIA				
ch60	R230	32 GiB	2 x 4 TB	THEIA Target #1
ch61	R230	32 GiB	2 x 4 TB	THEIA Target #2
ch69	R230	32 GiB	2 x 4 TB	THEIA Target #3
ch62	R230	32 GiB	2 x 4 TB	THEIA Analysis
ch63	R230	32 GiB	2 x 4 TB	THEIA Replay ADAPT #1
ch65	R230	32 GiB	2 x 4 TB	THEIA Replay MARPLE #1

Name	Type	Mem	Disk	Function
ch66	R230	32 GiB	2 x 4 TB	THEIA Replay RIPE #1
ch35	C6000	192 GiB	1 x 4 TB	THEIA Database
TRACE				
ch67	R230	32 GiB	2 x 4 TB	TRACE #1
ch68	R230	32 GiB	2 x 4 TB	TRACE #2
ch76	R230	32 GiB	2 x 4 TB	TRACE #3
TA2s				
ADAPT				
ch43	C6000	192 GiB	1 x 4 TB	ADAPT E3 #1 (VM1)
ch44	C6000	192 GiB	1 x 4 TB	ADAPT E3 #2 (VM4, VM5)
ch45	C6000	192 GiB	1 x 4 TB	ADAPT E3 #3 (VM3, VM5)
ch46	C6000	192 GiB	1 x 4 TB	ADAPT E3 #4 (VM2, VM6)
ch77	R230	32 GiB	2 x 4 TB	ADAPT #5
ch78	R230	32 GiB	2 x 4 TB	ADAPT #6
ch75	R230	32 GiB	2 x 4 TB	ADAPT #7 (CADETS demux)
MARPLE				
ch47	C6000	192 GiB	1 x 4 TB	MARPLE E3 #1
ch48	C6000	192 GiB	1 x 4 TB	MARPLE E3 #2
ch49	C6000	192 GiB	1 x 4 TB	MARPLE E3 #3
ch50	C6000	192 GiB	1 x 4 TB	MARPLE E3 #4
ch82	R230	32 GiB	2 x 4 TB	MARPLE #5
ch94	R230	32 GiB	2 x 4 TB	MARPLE #6
RIPE				
ch39	C6000	192 GiB	1 x 4 TB	RIPE E3 #1 (VM1)
ch40	C6000	192 GiB	1 x 4 TB	RIPE E3 #2 (VM2, VM7)
ch41	C6000	192 GiB	1 x 4 TB	RIPE E3 #3 (VM3, VM4)
ch42	C6000	192 GiB	1 x 4 TB	RIPE E3 #4 (VM5, VM6)
TA3				
ch02	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA3 E3 dedicated test #1
ch09	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	E3 Kafka+ZK #1
ch10	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	E3 Kafka+ZK #2
ch11	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	E3 Kafka+ZK #3
ch12	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	E3 Kafka+ZK #4
ch13	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	E3 Kafka+ZK #5
ch14	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	E3 Kafka+ZK #6
ch23	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Prometheus #1
ch52	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA3 performance 1

Name	Type	Mem	Disk	Function
ch54	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA3 performance 2
ch57	R230	32 GiB	2 x 4 TB	TA3 performance 3
TA5.1				
ch15	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA5.1 Linux mail server
ch72	R620	16 GiB	1 x 500 GB, 1 x 1.5 TB	TA5.1 activity generation
ch99	R230	32 GiB	2 x 4 TB	Uninstrumented pivot hosts 1, 2, and 3
ch98	R230	32 GiB	2 x 4 TB	Uninstrumented pivot hosts 4 and 5
TA5.2				
ch26	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA5.2 Windows 10 #1
ch27	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA5.2 Windows 10 #2
ch28	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA5.2 Ubuntu #1
ch29	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA5.2 Ubuntu #2
Testing Range				
ch07	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TRACE testing
ch18	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	ADAPT IO testing #1
ch19	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Test cluster Kafka+ZK #1
ch20	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Test cluster Kafka+ZK #2
ch21	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Test cluster Kafka+ZK #3
ch22	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Test cluster Kafka+ZK #4
ch24	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	THEIA target testing
ch30	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Free
ch31	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Five Directions testing
ch32	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	MARPLE TA1 testing
ch33	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA3 internal testing33 VM
ch34	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Five Directions translator testing
ch38	C6000	192 GiB	1 x 4 TB	Free
ch51	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	TA3 test cluster client
ch53	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	THEIA analysis testing
ch55	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	CADETS testing
ch70	R620	16 GiB	1 x 500 GB, 1 x 1.5 TB	Free
ch79	R230	32 GiB	2 x 4 TB	Free
ch80	R230	32 GiB	2 x 4 TB	Free
ch81	R230	32 GiB	2 x 4 TB	Free
ch95	R230	32 GiB	2 x 4 TB	Free
ch96	R230	32 GiB	2 x 4 TB	Free
ch97	R230	32 GiB	2 x 4 TB	Free

Name	Type	Mem	Disk	Function
CI Range				
ch36	C6000	192 GiB	1 x 4 TB	CI Runner
ch37	C6000	192 GiB	1 x 4 TB	CI Runner
ch83	R230	32 GiB	2 x 4 TB	CI Runner
ch84	R230	32 GiB	2 x 4 TB	CI Runner
ch85	R230	32 GiB	2 x 4 TB	CI Runner
ch86	R230	32 GiB	2 x 4 TB	CI Runner
ch87	R230	32 GiB	2 x 4 TB	CI Runner
ch88	R230	32 GiB	2 x 4 TB	CI Runner
ch89	R230	32 GiB	2 x 4 TB	CI Runner
ch90	R230	32 GiB	2 x 4 TB	CI Runner
ch91	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	CI Runner
ch92	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	CI Runner
ci-runner-01	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Gitlab CI
ci-runner-02	R210 II	32 GiB	1 x 4 TB, 1 x 1 TB	Gitlab CI

APPENDIX G Engagement #3 Policy Demonstration Ground Truth Information

G.1 Policy One Ground Truth

Policy:

ALLOW only requests coming from the "admin" user (UID 1004)

Request One: Expected: PASS

Request generated by the admin user executing a curl call from a bash script

```
retval=$(curl -s -o /dev/null -w "%{http_code}" $url/upload)
```

Request Two: Expected: BLOCK

Request generated by the "user" user (UID 1005), executing curl from a bash script (same command as #1)

Request Three: Expected: BLOCK

"user" user executing a bash script that calls a C program (called 'program3') with the setuid sticky bit set, so it runs as the "admin" user. The program execs a call to curl to send an http get.

Request Four: Expected: PASS

"admin" user running a curl command through sudo -u user, so it runs as the "user" user. Expect this to be allowed, since the original user is “admin”.

```
retval=$(sudo -u user curl -s -o /dev/null -w "%{http_code}" $url/download)
```

Request Five: Expected: PASS

“admin” running 'program3' via exec from a bash script, this is the same program as in #3 where it sends an http get via an exec to curl, and has the sticky bit set so it runs as “admin”. Since “admin” is the original user, this should be allowed

Request Six: Expected: BLOCK

"user" user running 'program1' via exec from a bash script through sudo. This program originally runs as the superuser but makes a seteuid system call to switch to “admin” (1004). It then executes the same http get action via an exec of curl.

G.2 Policy Two Ground Truth

Policy:

BLOCK any request from a process where the process previously communicated with a blacklisted IP address.

Request One: Expected: PASS

Execute a bash script that iterates through a list of URLs, none of which resolve to the unallowed IP address, then requests a policy protected file (uploads/TCCDMDatum.avsc). Requests are generated via curl.

```
retval=$(curl -s -o /dev/null -w "%{http_code}" $url/uploads/TCCDMDatum.avsc)
```

Request Two: Expected: BLOCK

Execute a bash script that iterates through a list of URLs, the second of which resolves to 128.55.12.59:8080:/uploads/haxx.php. The final request is to download TCCDMDatum.avsc from the policy protected page. This request should block, since the parent calling process (bash script) communicated with 128.55.12.59.

Request Three: Expected: PASS

Execute a C program that iterates through a list of URLs and generates GET requests via C socket calls. None of the URLs in the list resolve to 128.55.12.59, so the final policy protected request should pass.

Request Four: Expected: BLOCK

Execute a C program that iterates through a list of URLs that contains one request to 128.55.12.59:8080:/uploads/haxx.php. The final policy protected request should thus block.

Request Five: Expected: PASS

Execute a C program that iterates through a list of URLs, none of which are policy protected. A separate process opens a listening UDP socket and eventually receives data from 128.55.12.59. Since this is a separate process, the C program should be allowed to access the policy protected page.

Request Six: Expected: PASS

Two separate programs execute at the same time, the first is a bash script that iterates through a list of URLs, none of which are 128.55.12.59, and finally accesses the policy protected page (should pass). The second script also is a bash script (separate process) that does communicate with 128.55.12.59.

Request Seven: Expected: BLOCK

Execute a Java program that spawns a thread to listen on a UDP socket, and a separate thread to generate a set of GET requests. The UDP socket eventually receives data from 128.55.12.59. Later, the other thread generates a request to the policy protected page, which should block.

G.3 Policy Two Ground Truth for FAROS

The FAROS solution required a separate procedure to support the Policy Enforcement Demonstration.

Policy:

BLOCK any request from a process where the process previously communicated with a blacklisted IP address.

Scenario One Procedure: Expected: PASS

- Open Firefox
- Browse to 128.55.12.70:8080/example
- Browse to www.cnn.com
- Browse to 128.55.12.70:8080/pages/wireshark.html

- Browse to 128.55.12.70:8080/download

Scenario Two Procedure: Expected: BLOCKED

Close any open Firefox window.

Open a new Firefox browser, browse to the restricted server, then hit the policy protected page. Should be blocked.

- Browse to www.e-corp-usa.com
- Browse to 128.55.12.71:8080/uploads/haxx.php (this is the restricted server)
- Browse to www.google.com
- Browse to www.amazon.com/robots.txt
- Policy request to 128.55.12.70:8080/download

G.4 Policy Three Ground Truth

Policy:

Block if no portion of the request originated from a definite User Interface action.

1. Scripted access to unprotected files: Expected: PASS
2. Scripted access to protected file: Expected: BLOCK
3. Manual access to unprotected files: Expected: PASS
4. Manual access to protected file. Expected: PASS

G.5 Policy Four Ground Truth

Policy:

Block uploads that contain data downloaded from a network connection.

Request One: Expected: PASS

Upload oreally.jpg. This file existed on the TA1 client before recording started in the ~ta3/scripts/policy4 directory.

Request Two: Expected: PASS

- On the TA1 client, copy the last 50 lines from dmesg into a new file in the temp directory, /tmp/newfile.txt
- Upload /tmp/newfile.txt

Request Three: Expected: BLOCK

- Download latency.txt from 128.55.12.71 to /tmp
- Rename latency.txt to TotallyLegitBenignFile.txt
- Upload TotallyLegitBenignFile.txt

Request Four: Expected: BLOCK

- Download haxx.php from 128.55.12.71 to /tmp
- Move haxx.php to /tmp/h/nothaxx.txt
- gzip nothaxx.txt

- Copy some other files from /etc/mail into /tmp/h
- tar czvf benign.tar.gz /tmp/h (this should include files from /etc/mail, and nothaxx.php.gz, which includes haxx.php as nothaxx.txt.
- copy benign.tar.gz to /home/ta3
- tar xzvf benign.tar.gz
- unzip h/nothaxx.gz
- Upload h/nothaxx.txt (which is really haxx.php)

Request Five: Expected: PASS

- Run a Java program that writes some log4j log messages to P4L.log
- Upload P4L.log

Request Six: Expected: BLOCK

- Run a Java program that writes some log4j log messages to P4L.log
- Download latency.txt from 128.55.12.71 to /tmp/latency.txt
- cat /tmp/latency.txt P4.Log >/tmp/p.txt
- Upload /tmp/p.txt

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

ADAPT	A Diagnostic Approach for Persistent Threat Detection
ADB	Android Debug Bridge
AIA	Acuity Intelligence Agent
API	Application Programming Interface
APT	Advanced Persistent Threat
BAA	Broad Agency Announcement
BGP	Border Gateway Protocol
CDM	Common Data Model
CI	Continuous Integration
CIDR	classless inter-domain routing, Internet Protocol v4 address
CPT	Cyber-Protection team
CPU	central Processing Unit
CRASH	Clean-Slate Design of Resilient, Adaptive, Secure Hosts
DARPA	Defense Advanced Research Projects Agency
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
Gb	gigabits
GB	gigabytes
Gbps	gigabits per second
GPS	Global Positioning System
HDFS	Hadoop Distributed File System
ID	identification, identity
IDL	interface definition language
IM	instant messaging
IO	input output
IT	Information Technology
JSON	Javascript Object Notation
KVM	Kernel-based Virtual Machine
LARIAT	Lincoln Adaptable Real-time Information Assurance Testbed
LDAP	Lightweight Directory Access Protocol
LTS	long-term support
MARPLE	Mitigating APT Damage by Reasoning with Provenance in Large Enterprise Networks
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
OpTC	Operational Transparent Computing
OS	Operating System
PEM	Policy Enforcement Module
PXE	Preboot Execution Environment
QPR	Quarterly Program Review

RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
RIPE	Rapid Identification and Prevention of Exfiltration
SASL	Simple Authentication and Security Layer
SQL	Structured Query Language
SSL	Secure Sockets Layer
STARC	Scalable Transparency Architecture for Research Collaboration
TB	terabytes
TC	Transparent Computing
THEIA	Tagging and Tracking of Multi-Level Host Events for Transparent Computing and Information Assurance
TRACE	Tracking and Analysis of Causality at Enterprise Level
TSDB	Time Series Database
UI	User Interface
URL	Universal Resource Locator
USB	Universal Serial Bus
UUID	Universal Unique Identifier
VLAN	Virtual Local Area Network
VM	Virtual Machine