Lincoln SAMPLE Phase 1

Challenge Problem Announcement

Background

Cybersecurity experts are called in after a vulnerability has been exploited. These experts are faced with a difficult task. They must wade through gigabytes of operational logs searching for the crucial clues that explain how the vulnerability was exploited and what data may have been compromised or exfiltrated. If the enterprise is still under attack, finding this information may be the key to ending the attack. Lincoln is performing important research to address this problem by developing tools to collect, analyze and visualize data provenance within the enterprise network.*Data provenance* is the history of ownership and modification of data in order to evaluate its authenticity and correctness. Data provenance provides the answers to several important cybersecurity questions:

- Where is my data?
- Where did it come from?
- Who made changes to it?
- Where is it vulnerable?
- What is the mission impact of an attack?
- What do I do after being attacked?

The system which Lincoln designed is capable of collecting data at different system scales. For example, the provenance data used in this application will be produced by one of the following mechanisms:

- Applications-level data provenance enabled by directly instrumenting applications
- System-level data provenance enabled by modifications to the operating system kernel

The amount of information collected at the system-level can be orders of magnitude greater than the collection for single application.Typically, once collected the provenance data is stored in one or more databases where it can b retrieved for analysis and visualization. Decisions about the type and number of backing databases can change fo each system instrumented to collect data provenance. Therefore, the analysis and visualization component of the toolchain must be easily adaptable to accommodate this situation.

NOTE*: Phase 1 will not support multiple databases or changes to data format and interfaces (i.e. SQL vs. NoSQL) Instead, we will provide a method of loading a single database with different providence data. Later phases of the program may explore more drastic changes to the database.*

Platform Overview

The Simple Adaptive Multi-Parameter Lincoln Example (SAMPLE) platform represents the analysis and visualizati components of a larger collection, monitoring and forensic analysis tool suite. The goal of the SAMPLE platform is provide a usable tool for cybersecurity professionals supporting both continual monitoring and forensic activities across the enterprise computing environment. The SAMPLE platform was designed with usability, in particular responsiveness, in mind. Specifically, does the response time for drawing a provenance graph meet usability standards? There have been a number of usability studies going back to the 1960s that provide advice that's been validated for the web:

- Up to 0.1 seconds users perceive the system's response to be instantaneous. Here, they feel that they are directly manipulating the user interface and data.

- Up to about 1 second, users feel that the system is responsive. Users will likely notice a delay, but it is smal enough for their thought processes to stay uninterrupted. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1 second, but the user does lose the feeling of operating directly on th data.
- Up to about 10 seconds, users clearly notice that the system is slow, and their mind is likely to wander, but they are capable of maintaining some amount of attention on the application. Providing a progress bar is critical here.
- For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what tc expect.

The challenge of displaying a directed graph scales with the size and complexity of the graph. The available provenance data files represent provenance graphs that have different amounts of data. The smaller graphs shoul meet the usability criteria, the larger graphs are likely to fail.Figure 1 represents the SAMPLE platform that allows one or more analysts to visualize and interact with provenance data graphs to provide a post incident assessment key enterprise systems. The graphs are constructed from data stored in the database and presented to the analyst(s) interacting with a web browser.

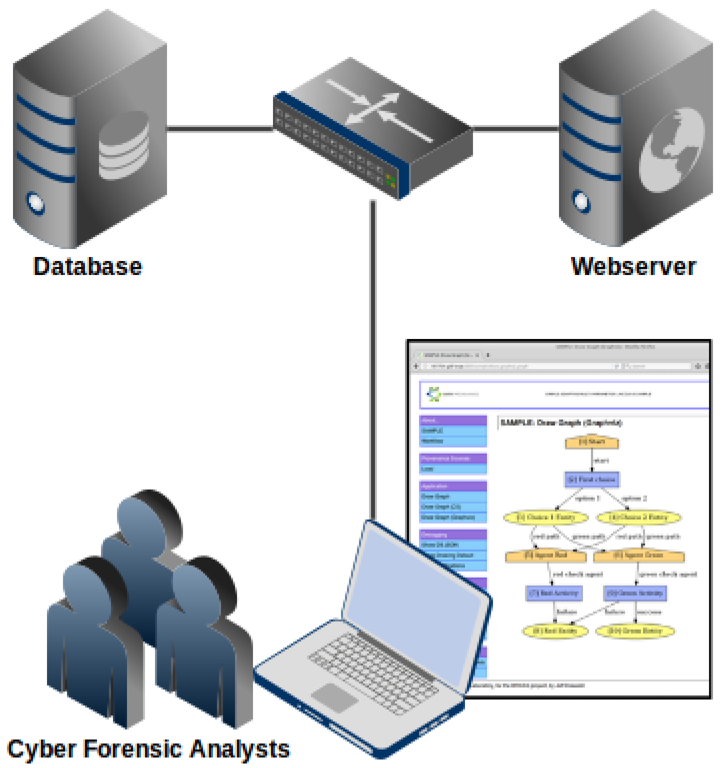


Figure 1: SAMPLE Platform Overview

Platform for Evaluation

Figure 2 further breaks out the components and their location relative to the system delivered for evaluation. It alsc highlights the interfaces between the Lincoln test harness and SAMPLE platform defined as follows:

1. Test Specification, Parameter Specification and Evaluation Specification
2. DAS State

3. DAS initialization configuration interface
4. Application and platform monitoring
    a. Collected by DAS and written to shared file system
    b. Collected directly by test harness
5. Ecosystem configuration (for changes which do not include specific API calls into the Platform, (i.e. changin
number of cores or amount of memory))
6. Client request/response interface (test harness will act as an analyst using the system)
7. API call to change the contents of the database
8. Response validation interface generated by the DAS and written to the shared file system

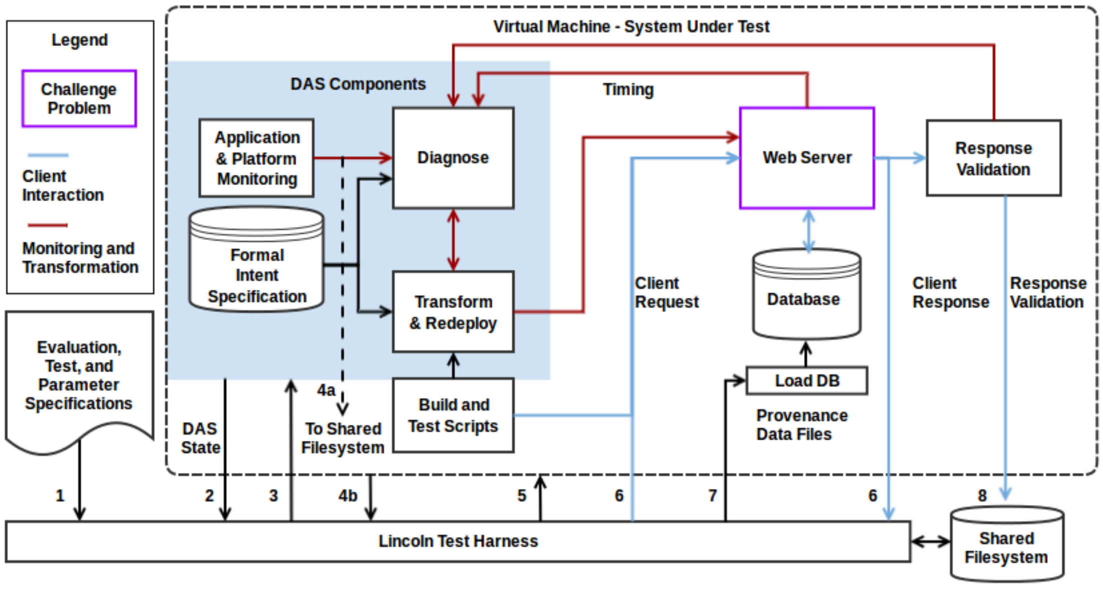


Figure 2: SAMPLE Platform for Evaluation

For the purposes of this design the “DAS” may provide some monitoring capabilities but will not provide any capabilities supporting diagnosis or transformation of the challenge problem application. At a minimum the “DAS” will be expected to communicate the current state of the system with the test harness.The web server will be writte in Ruby on Rails and backed by a SQLite3 database. The web app will have the ability to construct and display the provenience data in one of two ways:

- **Graphviz**, which requires that the graph is constructed by adding nodes and vertices to the graph in the Rul controller before the page is constructed. Graphviz then generates a static image file and image map file, which is referenced by the HTML page as it is rendered. There is no feedback to the user as the page loads
- **D3.js**, which requires that the data describing the graph is in JSON, constructed by the Ruby controller befo the page is referenced. This JSON is processed by Javascript on the page load, to construct the dynamic graph and render it. The result appears on the screen before the graph settles into its final state, which can give the user feedback that the application is at work.

Challenge Problem Descriptions

Challenge Problem 1: Supporting mission scope from continuous monitoring to forensic analysi (online)

Overview

This challenge problem explores the preservation of system responsiveness as the amount and type of data becomes available to the web application. Initially, analystsare likely to make decisions about the enterprise systen based on lower bandwidth logging information. However, the level of analysis (mission scope) changes when it is thought there may be a compromise of one or move systems on the enterprise network. The transition from continuous monitoring to forensic activities can instantaneously require orders of magnitude more data to be processed while still maintaining the responsiveness goals defined earlier.This challenge problem will also explore the preservation of system responsiveness while scaling the number of concurrent analysts accessing the system. As the number of analysts is increased it is expected that the rate at which queries

System Perturbations

- Provenance data size (number of nodes) and connectivity for both continuous and forensic graphs
- Number of concurrent analysts

Evaluation API

The SAMPLE team will provide definition of API calls enable the evaluator to loading different datasets. The team will provide a number of data sets for test. The format of these datasets will also be documented in the event the evaluator would like to generate data for use during system evaluation.

Evaluation Criteria

1. Correctness
2. Responsiveness
3. Availability

Challenge Problem 2: Platform upgrade (offline)

Overview

In order for the SAMPLE platform to be useful in a multitude of environments for which it may be deployed across the enterprise we would like to explore how changes to the physical processing capability of the system affect the user experience. The performance of the server to generate visualizations of large data sets is expected to be impacted by a reduction in one or more of the environment parameters listed below. This challenge problem represents an offline capability change to the processing system that could occur during a platform upgrade (or down grade) with the goal of maintaining a level of responsiveness called out earlier.

System Perturbations

- Number of cores
- Size of memory
- Disk space

Evaluation API

There are many ways to control the physical processing capabilities presented to an application under test. It is expected that none of methods used will require a SAMPLE specific API for affecting this change. However, in ord for the DAS to make well informed decisions about what system capabilities the challenge problem is actually operating with it may be necessary for the test harness to communicate the current capabilities of the system in some way. This will require more discussion between the SAMPLE and evaluation teams.

Evaluation Criteria

Same as CP1

Platform Integration

Virtual Machine Specification

Both CP1 and CP2 will use the same virtual machine specification and software build and deploy scripts.

1. What operating system(s) does your DAS and challenge problem application run on?

Ubuntu 14.04

2. What are the virtual hardware requirements for your DAS and challenge problem application (i.e. instruction set such as x86_64 or ARM, accelerator architectures such as GPU, etc.)?

The SAMPLE platform will target multi-core x86_64 processing architectures (we may want to allow for multi-socke architectures in the future), and needs ~2GB RAM, 1 GB hard disk space. The challenge problems will need at lease one network interface for each VM.

3. Describe any major or unique software dependencies necessary for building or running the DAS and challenge problem. This does not need to include common packages available directly from the OS repository but should include thing like Matlab, Intel Parallel Studio XE C++ compiler, Robot Operating System 2.x, etc.

- Ruby on Rails 2.3.0 (version available from the internet)

4. Describe how the DAS and challenge problem applications will be built, configured, and installed.

The SAMPLE challenge problem application can be built, configured, and installed using an Ansible provisioning script that we will provide. Most software dependencies (such as Ruby) will be downloaded from the web. In addition, the Ansible scripts will compile and start both the DAS and CP processes from code stored in the MIT GitHub repository. The Ansible script will be delivered via GitHub.

Interface API

Figure 2 identified all of the interfaces to/from the platform and test harness. This section will describe the API for each identified interface. We have chosen to use a RESTful API for all interactions with our DAS and challenge problem application. Use of this API to initialize and interact with the challenge problem will be described later. The API calls are described below and in the documentation in our Github repository.

Lincoln SAMPLE Challenge Problem API

```
// Interface(s) from DAS/CP to test harness
// -----------------------------------------

// DAS state (Fig. 2 interface 2)
POST /logs/status/DASSTATUS
 Parameter DASSTATUS = Starting | Repairing | Operational | ShuttingDown

// Interface(s) from test harness to DAS/CP
// -----------------------------------------

// Draw graph is the main interaction with the challenge problem
application (Fig. 2 interface 6)
GET /sample/draw_graph

// API call to perturb the dataset file (Fig. 2 interface 7)
POST /sample/load_provenance_tables?filename=FILENAME
 Parameter FILENAME = enumerated set of filenames described in parameter
language database:dataset


// Other DAS/Challenge Problem Configuration Options (not necessary for
tests)
//
---------------------------------------------------------------------------

// Change DAS adaptation settings (Fig. 2 interface 3)
POST
/adaptation/change_adaptation_settings?enable_adaptations=ADAPTIONS&minimum
_page_load_time_in_seconds=LTIME&timing_failures_before_adaption=FAILURES&s
imulated_adaption_time_in_seconds=ATIME
 Parameter ADAPTIONS = Boolean value for turning DAS adaption on or off
 Parameter LTIME = Integer number of seconds sets trigger for DAS adaption
threshold
 Parameter FAILURES = Integer number of responsiveness failures before DAS
takes action
 Parameter ATIME = Integer number of seconds to simulate a DAS repairing
event

// Change default method used to draw graphs (Fig. 2 interface 3)
POST /sample/change_drawing_default?graph_type=GTYPE
 Parameter GTYPE = d3 | graphviz | unknown

// Clear the table of notifications displaying DAS status (Fig. 2 interface
3)
GET /notifications/delete_all
```

The following table provides a description of what each DAS state means.

DAS State	Description
Starting	DAS is in the process of starting up, the status should switch to "Operational" once initialzation is complete

Operational	DAS is monitoring the challenge problem application
Repairing	DAS has detected a need for adaptation, and is going *ffline* for repairs
ShuttingDown	DAS is in the process of shutting down

Parameter Specification

This section describes the set of platform and mission parameters and describes their baseline state and constrain on configuration values for test and evaluation.

Lincoln SAMPLE Parameter Description File

```
// Paramter file for the Phase 1 Lincoln SAMPLE platform

// CP1 Parameters
// --------------

// Database selection is a runtime perturbation. The system should always
start in
// continuous monitoring mode by loading one of the c* files. It will then
transform
// into forensic analysis mode by swapping the dataset to one of the f*
files.
database : {
     available : True, always_true;
     dataset : continuous_00050_nodes_data, choose_one{
continuous_00050_nodes_data,

continuous_00100_nodes_data,

continuous_00150_nodes_data,

continuous_00200_nodes_data,

continuous_00250_nodes_data,

forensic_00300_nodes_data,

forensic_00400_nodes_data,

forensic_00500_nodes_data,

forensic_01000_nodes_data,

forensic_02000_nodes_data };
};

// The number of concurrent client interactions
numAnalysts : 1, range (1,20);


// CP2 Parameters
// --------------

// Number of processing cores
numCores : 1, num_processors (1,4);

// Amount of memory
sizeMem :is an  1024, ram_capacity (1024,2048);

// Amount of disk space
sizeDisk : 128, disk_capacity (128,256);
```

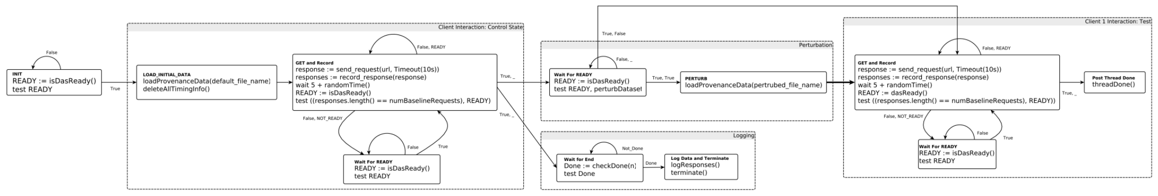
Test and Evaluation

Challenge Problem 1

Test Specification

TODO

Test Automation



TODO - Provide an overview description of the automation.

Figure 3 CP1 Test Automation

Intent Element Definition and Evaluation

The following questions pertaining to formal intent and DAS capabilities will help MIT LL better understand how to work with your system. Since the SAMPLE platform does not actually have a DAS capability these questions remain unanswered below.

1. Do you statically or dynamically validate the intents?
    a. If statically, do you have proofs? How are the proofs generated?
        i. How can LL verify the proofs? (if we can't you fail the CP)
    b. If dynamically, do you have intent checkers (i.e., runtime monitors)? Intent checkers/runtime monitors are executables that encode the intent.
    c. How is the intent checker implemented? Are the intent checkers online or offline checkers? (Online checkers check intents while SUT is executing; offline checkers check intents after SUT finishes executing.)
        i. If online, how can Lincoln gain access to the checkers' source code to verify that the checker i valid?
            - Do you provide APIs so Lincoln can get access to the events the online checker consumes?
            - Do you provide offline checkers that perform post processing on the events?
            - Do you provide a log of the events the checker requires?
        ii. If offline, are the checkers standalone programs?
            - How can LL verify that the checker is valid? (need the source code with some annotations)
            - Can LL execute the checkers?
            - How are the events needed by the checkers collected?
2. Do you have test cases for each intent? (one test for each intent, many tests for one intent, or one test for many intents?)

a. How should LL execute these tests? In sequence?
b. How do you deal with tests that become obsolete when the SUT adapts?

Intent Element 1: Accuracy

Informal Description: Server responses reflect the contents of the database at the time the request was received

Formal Description: After the perturbation, the percent of correct responses is eventually comparable to the baseline accuracy, for an adequate number of observations.

Test/Measurement Description: The server will hash the contents of the database values used to generate the graph image for each client request (represented by the *response validation* block in Figure 2) . Hash values durin test will be written to the shared file system. The SAMPLE platform will also provide the mapping from file name to expected hash value as a key, value pair stored on the shared file system.

Result Expression: {(request, response, status, time)}

For each *request, time* is the observed wall-clock time between the time the request is sent by the client and the time the response was received by the client. *response* and *status* values are defined as follows:

Response	Status	Definition
NULL	CORRECT	The response accurately matches the contents of the database at the time the request was received by the server
NULL	INCORRECT	The response did not accurately match the contents of the database at the time the request was received by the server
NULL	TIMEOUT	The server failed to respond in time
NULL	ERROR	The server reported an error

Verdict Expression: $\mathbf{A}_{\mathrm{C}_{\mathrm{baseline}}}$ is the average number of correct answers during the baseline interaction

$\#\{\text{response} \neq \text{NULL}\} < 200$	INCONCLUSIVE
$\text{eventually} \left(\begin{array}{l} \#\{\text{response} \neq \text{NULL}\} > 100 \\ \wedge \\ \frac{\#\{\text{status=CORRECT}\}}{\#\{\text{response} \neq \text{NULL}\}} > 0.97 * \mathbf{A}_{\mathrm{C}_{\mathrm{baseline}}} \end{array} \right)$	PASS
$\text{eventually} \left(\begin{array}{l} \#\{\text{response} \neq \text{NULL}\} > 20 \\ \wedge \\ \frac{\#\{\text{status=CORRECT}\}}{\#\{\text{response} \neq \text{NULL}\}} > 0.90 * \mathbf{A}_{\mathrm{C}_{\mathrm{baseline}}} \end{array} \right)$	DEGRADED
else	FAIL

Intent Element 2: Responsiveness

Informal Description: The client receives responses in a timely manner.

Formal Description: After the perturbation, the median response time is eventually comparable to the baseline responsiveness (median and median absolute deviation (MAD)), for an adequate number of observations.

Test/Measurement Description: The response time of each HTTP request should be directly measured by the te harness. Since the test harness is not actually rendering the page this measurement should be representative of th time it took the server to service each request (i.e. the observed wall-clock time between the time the request is received by the server and the time the response was sent by the server).

Result Expression: {(request, response, status, time)}

For each *request, time* is the observed wall-clock time between the time the request is sent by the client and the time the response was received by the client. *response* and *status* values are defined as follows:

Response	Status	Definition
NULL	CORRECT	The response accurately matches the contents of the database at the time the request was received by the server
NULL	INCORRECT	The response did not accurately match the contents of the database at the time the request was received by the server
NULL	TIMEOUT	The server failed to respond in time
NULL	ERROR	The server reported an error

Verdict Expression:

$\#\{\text{response} \neq \text{NULL}\} < 200$	INCONCLUSIVE
$\text{eventually} \left(\begin{array}{l} \#\{\text{response} \neq \text{NULL}\} > 100 \\ \wedge \\ \text{Median}\{\text{time} \mid \text{response} \neq \text{NULL}\} < \text{Median}_{\text{baseline}} + \text{MAD}_{\text{baseline}} \end{array} \right)$	PASS
$\text{eventually} \left(\begin{array}{l} \#\{\text{response} \neq \text{NULL}\} > 20 \\ \wedge \\ \text{Median}\{\text{time} \mid \text{response} \neq \text{NULL}\} < \text{Median}_{\text{baseline}} + (2 * \text{MAD}_{\text{baseline}}) \end{array} \right)$	DEGRADED
else	FAIL

Intent Element 3: Availability

Informal Description: The server responds to almost all requests.

Formal Description: After the perturbation, the percent of responses is eventually comparable to the baseline availability, for an adequate number of observations.

Test/Measurement Description: The test harness should log HTTP response status codes for each interaction which does not timeout; the test harness should also log a message for each request that times out.

Result Expression: {(request, response, status, time)}

For each *request*, *time* is the observed wall-clock time between the time the request is sent by the client and the time the response was received by the client. *response* and *status* values are defined as follows:

Response	Status	Definition
NULL	CORRECT	The response accurately matches the contents of the database at the time the request was received by the server
NULL	INCORRECT	The response did not accurately match the contents of the database at the time the request was received by the server
NULL	TIMEOUT	The server failed to respond in time
NULL	ERROR	The server reported an error

Verdict Expression: $\mathbf{A}_{\mathbf{V}_{\text{baseline}}}$ is the average availability during the baseline interaction

$\#\{\text{requests}\} < 200$	INCONCLUSIVE
$\text{eventually}\begin{pmatrix} \#\{\text{request}\} > 100 \\ \wedge \\ \frac{\#\{\text{response}\neq\text{NULL}\}}{\#\{\text{request}\}} > 0.97 * \mathbf{A}_{\mathbf{V}_{\text{baseline}}} \end{pmatrix}$	PASS
$\text{eventually}\begin{pmatrix} \#\{\text{response} \neq \text{NULL}\} > 20 \\ \wedge \\ \frac{\#\{\text{response}\neq\text{NULL}\}}{\#\{\text{request}\}} > 0.90 * \mathbf{A}_{\mathbf{V}_{\text{baseline}}} \end{pmatrix}$	DEGRADED
else	FAIL

Challenge Problem 2

Test Specification

TBD...

Test Automation

TBD...

Intent Element Definition and Evaluation

TBD...