Software Requirements Specification

Dustin Ingram
dsi23@drexel.edu

Aaron Rosenfeld
mjk75@drexel.edu

Maria Kolakowska
fjc28@drexel.edu

Frank Clark
ar374@drexel.edu

November 1, 2010
Version 1

Abstract

STAGE provides the capability to test the veracity of system-level sensors and component-level algorithms in the laboratory against virtual operations using data from simulated sensors and nodes. These nodes will work with scripted or real time data, depending on the user's need. This network and sensor testbed allows users to independently vary its topology, application suites, and the environment. It also emulates the communication between these components. A visualization tool is provided to the user for demoing and analyzing.

Contents

1 Introduction

1.1 Purpose

This requirements document defines the functional and non-functional requirements for the STAGE project. These requirements will include, but are not limited to, the performance, interfaces, and hardware goals and limitations. The information presented here is intended for the development team and the external stakeholders, currently Dr. William Regli, Mr. Joseph B. Kopena, Mr. Joeseph P. Macker, and The U.S. Naval Research Laboratory.

1.2 Scope

The goal of the ——— project is to automate the passive discovery, recognition, and consumption of semantic web services. ——— does not require any central registry of services but will examine network traffic patterns to identify specific services, how to invoke them and report specific, user chosen information. As well as running with known service definitions, the system will identify new services and analyze relevant network traffic to determine how to consume the service. This application will be constructed of an extensible core framework upon which future modules can be developed and integrated.

——— is split into 3 general pieces: the data input, data processing, and user interface modules. The data input module is responsible for collecting traffic off the network or using a Pcap file as a source instead of a live network. The data processing module, which is constructed from various filters and processors (which will be described in more depth later), applies filters to collected information and transforms it into various and configurable reports and views depending on what the user wants to see. Some examples of these reports will be described later. Lastly, ——— has both a CLI and a GUI for viewing the various data returned by the data processors and filters. These interfaces will allow users to interact with the data returned. For example, a user could apply additional display filters, such as filtering by a specific destination IP. Both interfaces will offer to users a report of what web services appear to be available on the network and supply a report - an ad-hoc registry of sorts - which users can use to consume the web service. Users may specify a additional types of filters or processors to generate different types of reports via an extension system.

Users of ——— are chunked into 3 loose groups: system/network administrators who wish to control all of the ——— pieces (data input, data processing, and the user interface), developers who wish to or are tasked with developing custom data filters and/or processors so users can have new/custom reports, and UI users who only wish to analyze the data collected and processed by ———.

1.3 Definitions, Acronyms, and Abbreviations

Foobar A foobar

1.4 References

These documents explain in-depth about the various technologies involved in this project.

- WSDL Specification: http://www.w3.org/TR/wsdl20/
- SOAP Specification: http://www.w3.org/TR/soap/
- libpcap Homepage: http://www.tcpdump.org/
- MVP-E Project (Team 10): http://www.cs.drexel.edu/SeniorDesign/2010Material/Projects2010.html

1.5 Overview

The rest of this document is organized as follows:

- Section 2 gives a high level overview of the project requirements.
- Section 3 gives more detailed project requirements.
- Sections 3.3 and 3.4 provides detail in the inputs and outputs of the system,
- Section 3.1 specifies what the software will do,
- Section 4.1 gives performance requirements for the underlying system.

The rest of Section 3 explains other minor requirements.

2 Overall Description

2.1 Product Perspective

STAGE can be used as a testbed for sensors and respective algorithms in the laboratory against virtual operations using data from simulated sensors and nodes. The ability to independently and quickly vary the network topology, application suites, environment, or any other variable will be essential to collect data which would be cost-prohitive to produce in a real-world scenario. Furthermore, the visualization tool can be used for conceptual demonstrations.

2.1.1 System Interfaces

STAGE combines:

- Scenario Language - Defining the entire scenario in a very simple language
- Agent Framework - Supporting the customization of semi-intelligent agents
- Topology Creator - Creating a virtual topology of nodes defined in the Scenario Language
- Distribution Framework - Distributing a large-scale emulation across numerous physical machines
- Data-Collection API - Collecting all relevant data through a simple API
- Visualization Engine - Conceptually visualizing terrain, node movement, link status, etc.

2.1.2 User Interfaces

STAGE's user interface is two-fold, with each interface for different tasks and uses. The basic interface allows a user to simply and quickly define a scenario with the Scenario Language and begin an experiment, using the API to collect data in a way that is defined by the user. The Visualization Engine allows the user to interpret the experiment visually, without a need for data-collection or the API.

2.1.3 Hardware Interfaces

STAGE requires that the machine running the system has one to many network interfaces to support distribution.

2.1.4 Software Interfaces

STAGE requires the use of Linux Namespaces' kernel virtualization as well as OpenGL in order to use the visualizer.

2.1.5 Memory Constraints

STAGE requires a machine with at least two gigabytes of RAM.

2.1.6 Site Adaptation Requirements

STAGE requires that the system is configured per-site to create a distributed emulation, or to interact on-the-fly.

2.2 Product Functions

This software performs two major functions: collecting network traffic related to web services, and anyalizing collected data in a user specified way using built-in or user-defined filters and processors. In performing these actions separately, either piece may be substituted by third-party software which performs the same actions and communicates over the same interface described in this document.

Network administrators, among other technical people, want to know what web services are being used on their network. Some users may not report that they're using specific web services on a network when they are, which may conflict with other services. By being able to see all of the web service invocations on a network, an administrator can determine where rogue web services are being used and better determine whether they offer something useful on the network.

Also, not all networks have a central registry of web services offered on the network, and most do not have a record of what web services from outside the network are consumed by users on the network. More, many web services aren't clearly defined in WSDL or an equivalent format, so it is unclear how to consume the web service without a formal definition. ——— constructs an ad-hoc registry of web services is observes being consumed in a defined way. This registry can then be used to let users consume these previously undefined web services.

Some networks have a requirement that all services being used on them be encrypted as the network itself cannot be trusted. ——— is able to show which services being used on a network are secure (either fully encrypted traffic or encrypted data with cleartext headers) and which are insecure. By looking at the specific traffic an administrator can learn what pieces of the network are vulnerable.

Questions commonly asked consist of: whether a web service is actively being used, if there's a pattern to who uses a specific web service, or whether there's a pattern to when a specific web service is used. Web service developers, as well as network administrators, frequently want to know what users and/or servers are using specific web services and when. This allows them to determine to whom a specific service is useful, who uses an unauthorized web service, and other similar situations.

In debugging web services, developers need to see what data is being sent to the web service during invocation and usage. Instead of using a program like Wireshark to have to look at the packets of a web service invocation, developers can use ——— to see where, if anywhere, the data actually sent differs from what should be sent.

2.3 User Characteristics

The system has three different types of users: administrators, developers and traditional users. Each type of user has different goals and needs to use different parts of the system. These user roles, while they may be performed by the same user, are here split into the three distinct roles for clairty:

2.3.1 Users: Jamie, Junior Data Analyst

Jamie solely uses the GUI to run/view web service analyst reports. Jamie wants to complete the tasks below on a network in real-time as allowed by the administrator, or based on a saved file of a previous capture in the GUI. Tasks:

- Get the full list of all services invoked
- See how to invoke a specific service
- View common services
- View rare services
- Finding patterns in service invocations

- Finding exceptions (anti-patterns) to the previous patterns
- Run reports and filtering by any combination of:
  - specific services
  - service specific criteria (e.g. Google Maps search vs. Google Maps display)
  - source IP address
  - destination IP address
  - protocol
  - protocol specific criteria (e.g. filtering SOAP content by Request Time)
  - date/time
  - secure services
  - insecure services
- Run reports with custom filters and processors setup by Susan

2.3.2 Developers: Susan; Senior Applications Developer

Susan needs to add more functionality to the application. Susan wants to complete the following tasks on a network in real-time as allowed by the administrator, or based on a saved file of a previous capture in either CLI or GUI mode. Tasks:

- Use an API
- Run application in CLI mode and/or GUI mode for testing
- Inspect their own services
- Add new protocols and/or services to be reported on

2.3.3 Administrators: Zed; Systems Administrator

Zed needs to capture traffic coming over the network card to store it in Pcap format and/or allow Susan and Jamie to run analysis against the live network stream. Zed needs to do this, as capturing from the network card requires root level access on the computer the system is being run on. Zed also needs to be able to:

- Deploy the software
  - on a single machine for full network scanning
  - on a single machine for scanning that same single machine
  - on a single machine for analysis of previously saved captures
  - with the GUI frontend (and associated files)
  - with the Data Processing layer (and associated files)
- Capture from the network in real time
- Store the capture in a Pcap file

2.4 Constraints

The system must be able to interpret and/or process any agent or topology it is given. If this isn't possible for any reason, the system should inform the user of this. No piece of this system requires elevated privileges.

2.5 Assumptions and Dependencies

It is assumed that a user, administrator, or developer has the ability to install required libraries which will named in the future as a result of the development process.

3 Specific Requirements

3.1 Functional Requirements

3.1.1 Data Input

3.1.1.1 Live Capture The system shall be able to read live traffic from a network device connected to the system. The user must be able to gain the necessary permissions in order to read data from the network device. If the system cannot read from the network device for any reason, the user must be notified with an error, and the system shall exit.

3.1.1.2 Offline Capture The system shall be able to read stored traffic from a Pcap formatted file. The file must exist and be readable by the user running the system. If the file cannot be read, or is of an incorrect format, the user must be notified with an error, and the system shall exit.

3.1.2 Filters

The system shall use filters in order to reduce the data into specific data subsets. These datasets will then be passed into other filters or processors for analysis.

3.1.2.1 Secure Connections The system shall provide a filter that takes as input a set of service invocations and will only return secured (encrypted) service invocations. If the input is malformed, the filter shall return nothing.

3.1.2.2 Unsecure Connections The system shall provide a filter that takes as input a set of service invocations and will only return unsecured (unencrypted) service invocations. If the input is malformed, the filter shall return nothing.

3.1.2.3 Source IP The system shall provide a filter that takes as input a set of available services and returns a list of available services from a specified source IP address, if any. The source IP address is a required parameter for this filter, if not present the filter shall return all services inputted. If the input is malformed, the filter shall return nothing.

3.1.2.4 Destination IP The system shall provide a filter that takes as input a set of available services and returns a list of available services which have been invoked by a particular user's IP address. The user IP address is a required parameter for this filter, if not present the filter shall return all services inputted. If the input is malformed, the filter shall return nothing.

3.1.3 Processors

The system shall use processors in order to transform data from one form to another. The output data will then be passed into other filters or processors for analysis.

3.1.3.1 Service Listing The system shall provide a processor which will take as input a set of TCP packets and return a list of service tuples. These service tuples shall contain:

- server name (obtained from DNS)
- server IP address
- type of service (SOAP, etc)
- service definition (WSDL, if possible)

If the input is malformed, the processor should return nothing.

3.1.3.2 Service Invocations

The system shall provide a processor which will take as input a set of TCP packets and returns a list of invocation tuples. The processor shall take as a required parameter the IP address and service name of the service in question. The invocation tuples represent a single service invocation, and shall contain:

- client IP address
- timestamp of invocation
- data sent (arguments) to service

If the input is malformed, or the required parameters are missing or invalid, the processor shall return nothing.

3.1.3.3 Useful Services

Within a network, some services are more useful to other users on the network than others. The usefulness of a service can be determined based on the number of invocations that are made to it over a given period of time. Using the information regarding service invocation, the system shall be able to determine usefulness of a service relative to the number of invocations.

The output of this type of analysis shall be a list of services containing the following information:

- server name (obtained from DNS)
- server IP address
- name of service
- number of invocations
- most recent invocation (YYYYMMDDHHMMSS)

This list shall be sorted in descending order by the number of invocations.

3.1.3.4 Abnormal Services

With in a network system, there may be a pattern to which services are accessed. This information is obtained based on what services different users connect to. After determining a pattern for which users are connecting to a set of services, a set of services and users that don't follow the pattern can be abstracted. The system shall be able to determine this list of abnormal services based on the invocation of different web services by users.

The output of this type of analysis shall be a list of abnormal services containing the following information:

- server name (obtained from DNS)
- server IP address
- name of service
- number of invocations
- most recent invocation (YYYYMMDDHHMMSS)
- list of IP making it abnormal

This list shall be sorted in order of number of invocations in ascending order. This will place the most abnormal services at the top of the list.

3.1.4 Data Storage

1. No outside database system will be used for data storage, unless it is introduced by the user via the API

2. Scenario definitions, including the topology and agents, will be stored and parsed by the system

3.2 Runtime Environment

- Operating Systems: Unix / Linux / Mac
- Processor: A modern 32- or 64-bit x86 processor

- Memory: 1024MB RAM minimum
- Network connection: Any recent network card which is currently connected to the emulation network (if the emulation is distributed)
- Visualizer: A graphics card which will support OpenGL or an equivalent

3.3 Input Formats

3.3.1 Packets

1. The system shall be able to read raw packets from a network device with libpcap.
2. The system shall be able to read raw packets from a file in the PCAP format.

3.3.2 Protocols

1. The system shall support HTTP as a transport protocol.
2. The system shall support SMTP as a transport protocol.
3. The system shall be able to recognize SOAP service invocations. An example is in Appendix **??**.
4. The system shall support extensions to recognize other protocols.
5. The system shall collate packets from recognized service invocations into single logical units.

3.4 Output Formats

1. The system shall support exporting a subset of captured packets into a user specified PCAP formatted file.
2. The system shall support exporting user defined PDF/HTML reports from data gathered by the system.
3. The system shall support generating WSDL for discovered web services.

3.5 User Interface

These requirements apply to both the Graphical User Interface (GUI) and the Command Line Interface (CLI).

1. The interface shall provide access to raw unfiltered captured web service information.
2. The interface shall provide the ability to filter web services based on user supplied criteria.
3. The interface shall provide the ability to process web service lists to gain more information.
4. The interface shall give the user the ability to produce reports based off the information in the interface.
5. The interface shall provide a method for providing a detailed inspection of web services.
6. The interface shall provide a method for viewing communication chain (Per user queries/responses from a server).

3.6 Extensibility

1. The system shall provide a means for users to create custom topologies.
2. The system shall provide a means for users to create custom agents.

3.7 Testing

All code with in the system must be tested. Core functionality of the software will have 100% code coverage with automated unit and integration tests. The visualizer testing will be done manually as it is a smaller component and would otherwise require a disproportionately large investment in initial setup. Unit tests will cover the algorithms used in our system while integration tests will be used to test our data-flow.

4 System Evolution

With more time, the STAGE team could create additional tools to aid in the the development of more complex agents or topologies, so that user customization does not become too granular. Furthermore, tools which can automate or assist in data collection and parsing could be implements. Even further development could lead to a platform that included Windows or other operating systems.

4.1 Performance

4.1.1 Real Time

The system shall have the ability to run in real time. Real time is defined as constantly processing the packets as they come off the wire. As the packets are processed the data will be presented to the user in either graphical form, or on a console. Much like Non-Real Time execution, all packets which have been determined to belong to a web service communication as defined in Section 1.3 will be stored in a Pcap file for later processing should the user choose to save this data.

4.1.2 Non-Real Time

The system shall have the ability to run in non-real time. Non-real time is defined as collecting the packets as they come off the wire and storing them in a data store for later processing and analysis. The format for storage of all web service packets is Pcap.

4.2 Documentation

1. The software shall provide a User Manual containing step-by-step instructions for user perspective in Section 2.3.
2. The software shall provide Developer Documentation containing:
    (a) Agent Framework definitions and example agents
    (b) Scenario Language syntax and usage