

# Session Capture and Replay Test Infrastructure

Submitted to:

Project Advisor: Mark Claypool, WPI Professor Computer Science

Shape Security Advisor: Dan Moen, Senior Software Engineer at Shape Security

Submitted by:

Andrew Rottier

Oba Seward-Evans

Akshit (Axe) Soota

Date: March 4, 2018

Department of Computer Science

C 2018

MQP

Submitted in Partial Fulfillment of  
The Major Qualifying Project Requirement  
Worcester Polytechnic Institute  
Worcester, Massachusetts



## **Abstract**

Shape Security is a Silicon Valley startup that helps other companies secure their websites and mobile application servers against automated attacks. Shape required improved tooling to consistently reproduce HTTP traffic which was passing through their product, called the “ShapeShifter.” This would allow easier reproduction of normal and abnormal traffic seen in customer environments, and help ensure that all traffic was being handled properly. Our group was tasked with developing a Chrome extension and a Bulk Replay Script to assist in the capturing and replaying of session information through Shape’s systems. We developed a Session Capture and Replay (SCR) Chrome Extension using HTML, CSS, JavaScript and React. Our team also developed a Bulk SCR Script in Python. We tested and validated these tools by demonstrating the tool suite to various teams at Shape, running performance metrics and iterating over feedback received. These tools should help accelerate the development cycle within Shape Security, increase the bug resolution rate, and improve the accuracy of automated attack prevention for Shape’s clients.

## **Acknowledgements**

We would like to thank our sponsor, Shape Security, for the opportunity to contribute to an internal solution for many teams. We would also like to thank our mentor Dan Moen for guiding our team throughout the process. Lastly, we would like to thank our MQP advisor Mark Claypool for his patience and support throughout the project and WPI for giving us this amazing opportunity.

# Table Of Contents

<b>Session Capture and Replay Test Infrastructure</b>	<b>1</b>
Abstract	2
Acknowledgements	3
Table Of Contents	4
1. Introduction	6
2. Background	8
2.1. Shape	8
2.1.1. Pegasus	8
2.1.2. Telemetry	9
2.2. Technology Stack	9
2.2.1. Chrome Extension	9
2.2.2. React	9
2.2.3. JIRA	9
3. Design	11
3.1. Problems and Requirements	11
3.2. Design of System (System Overview)	11
3.3. Design of the Session Capture Replay (SCR) Extension	12
3.4. Design of the Bulk Session Capture Replay script (Bulk SCR script)	13
4. Implementation of the SCR extension	15
4.1. Axios	15
4.2. JSON Tree Viewer	15
4.2.1. Modifying Existing Module	15
4.2.2. Internal Node Package Registry	16
4.3. Validation	17
4.3.1. Hierarchical Validation States	19
4.3.2. Local Storage Interface	22
4.4. Policies Tab	22
4.5. Capture Tab	24
4.6. Replay Tab	26
4.7. JIRA Integration	27
4.7.1. Login	27
4.7.2. Upload	27
4.7.3. Download	28
4.8. Settings Tab	29
4.8.1. Default Validators	30

4.8.2. Namespaces to Objectify	31
4.8.3. Import Export Settings	32
4.8.4. Chrome Local Storage Interface	33
4.9. Testing	33
4.9.1. Mocha	33
4.9.2. Istanbul	33
4.9.3. Sinon	34
4.9.4. Chai	34
4.9.5. Enzyme	34
5. Implementation of Bulk SCR Script	35
5.1. Invocation	35
5.2. Replay Integration	36
5.3. Validation	36
5.4. Validation Results and Script Summary	36
6. Evaluation and Results	37
6.1. SCR Validation	37
6.2. Policy Effectiveness	37
6.3. JIRA Integration	38
6.4. Performance of Bulk SCR	38
6.5. Bulk SCR Output	39
7. Future Work	39
7.1. SCR Extension Improvements	39
7.2 Bulk SCR Script Improvements	40
7.3. SCR Ecosystem Improvements	41
8. Conclusion	41
9. Glossary	42
10. References	44
Appendices	45
Appendix A: Usability Questionnaire	45
Other feedback:	45
Appendix B: Generic SCR script analyzed result JSON for one policy	46

# 1. Introduction

With 3.8 billion Internet users in 2017, cyber security plays an increasingly important role in today's online world. <sup>[1]</sup> As more and more data, such as personal, financial and medical information, is being stored online, security is rapidly becoming more crucial. This data is accessible through various website's endpoints. When an endpoint is not properly secured, the website and its data are vulnerable to attacks. Attacks violate the integrity of the application and the trust of its users.

Shape Security is a security company located in Mountain View, California which proactively mitigates malicious behaviors on their clients websites. <sup>[2]</sup> To date, the company has prevented over \$1 billion in fraud by deflecting attacks including account takeover, website scraping, and gift card theft on some of the world's most commonly used web and mobile applications. <sup>[3]</sup> Shape's primary defense against automated attacks is Pegasus which is a scriptable reverse proxy that sits between client endpoints and servers.

Shape Security has identified a need to capture and replay Internet traffic that travels through their system to their client's servers. At the time, there was no internal infrastructure to run replays of sessions through their system. Having the ability to replay this data would provide a useful service for several teams at Shape. The replay functionality enables Shape's Quality Assurance (QA) team to analyze how individual web requests are handled by Shape. It allows the Research team to analyze previously captured web traffic and detect malicious behavior. It enables the Development teams to better understand how problematic traffic is being handled or mishandled by experimental or production software. It also helps the Threat Mitigation team evaluate the effectiveness of new filtering rules by inspecting how the new rules affect old archived sessions.

The solution to these issues is solved with Session Capture Replay (SCR) Chrome Extension. For this tool to be effective, the Session Capture Replay extension needed to include an interaction with Pegasus' Shape Shifter Element (SSE) REST API, an easy-to-use front end site to handle individual session captures, and a way to analyze bulk loads of previously captured data. The extension also needed to validate the states of the responses as it flows through Shape's system.

To address these requirements, our team built an SCR extension that enables captured data to be replayed through Shape's system. Pegasus already had API endpoints for capture and replay functionality, however, it was not easily accessible in the prototype version of the SCR. The original SCR prototype lacked the ability to compare the captured session and the replay. Additionally, this tool provides validation to the flags that are set and appended to the data when flowing through Shape's system. The

validators are methods that specify how each field of the capture and replay sessions are compared against each other. Validators provides quantification and clarity to the effectiveness of Pegasus. In addition to this tool, our team built Bulk SCR, a script to handle replaying large amounts of previously stored transactions that flowed through Shape's system. This script applies new filter rules to millions of archived captures, replays those captures, and compares the results with preset validations for further analysis.

Implementation of these tools introduced the following new features for Shape: evaluate the effectiveness of a policy by validating sessions data for both individual and batches of session captures, provide a new graphical interface for visualizing validation results of replayed session data, and lastly, integrate the application with JIRA to expedite the QA processes within Shape.

Different ways future development teams could continue development on these tools include:

1. moving data used in the extension to a global state to reduce the amount of data flowing through the application
2. improving the efficiency of the Bulk SCR tool to be able to handle replay and validation at a higher rate
3. taking historically logged transactions and convert it to capture format.

Section 2 discusses necessary background information for our project. Section 3 expands on the problems and requirements given to us by Shape Security as well as system designs for our project. Section 4 details the implementation of the SCR extension, while Section 5 details the implementation of the Bulk SCR script. Section 6 discusses evaluations and results of this project. Section 7 contains recommendations for future maintainers of these tools. Lastly, Section 8 concludes our project.

## 2. Background

This section discusses Shape Security and its products. Additionally, it talks about the technology used to build the various Session Capture Replay tools.

### 2.1. Shape

The work related to this project requires knowledge of Shape's internal architecture. This section describes how this project integrates into Shape's architecture. Additional terminology can be found in the Glossary section of this report.

#### 2.1.1. Pegasus

Pegasus is the scriptable reverse proxy deployed by Shape that sits between client endpoints and servers inside the SSE. This layer is responsible for enforcing policy configurations and eliminating bot traffic and malicious behavior. A policy is a set of customized configurations that Shape uses to filter traffic to client websites. Reports about all traffic that is intercepted by Pegasus are then batched and sent to Telemetry, described in the next section.

Pegasus processes both pre- and post- requests as well as pre- and post- responses that are sent to their client's servers. The data is logged and summarized in an elaborate system of flags which are used to describe the behavior of the transaction between the user's computer and the client's server for each user interaction on the client's website. When an attack occurs, Shape can identify the malicious behaviour using these flags, and subsequently create new rules to block the interactions and future interactions that match that malicious behavior.

Pegasus contains a REST API that enables specified, labelled traffic to be recorded. There are additional endpoints on the SSE which allow enabling, disabling, and fetching the captured data, as well as running a replay of captured data back through Pegasus under a new policy. Access the REST API is, by default, disabled out in production, so the only way to access it, is by running it under special configurations. To maintain privacy, capture is only enabled on traffic sent with specific request headers which prevents the SSE from capturing data from any user other than the one that enabled capture mode.



### 2.1.2. Telemetry

Data aggregated from Pegasus is eventually sent to Telemetry. Telemetry is the data-store that archives all session data that passes through Pegasus from users using client endpoints. Research teams within Shape use this data to find trends, isolate past attacks, and gain other insights. Shape's Customer Support Team also uses Telemetry data to identify when customers are under attack and if policy changes are having the intended effect.

## 2.2. Technology Stack

This section is an overview of the technologies used in the implementation of the SCR extension and the Bulk SCR script.

### 2.2.1. Chrome Extension

The SCR user interface is implemented as a Chrome extension. Chrome extensions are browser add-ons that can increase the functionality of the Chrome browser. Chrome extensions are developed using basic web stack technologies such as HTML, JavaScript, and CSS. <sup>[4]</sup> They also support frameworks such as React and Angular which enable design of a more legible and reactive user interface. Additionally, the Chrome Extension API provides many useful features such as Chrome storage as well as a suite of built-in methods to retrieve live browser information that can be consumed by a Chrome Extension.

### 2.2.2. React

React is a frontend framework developed in JavaScript utilizing the concept of components and Object-Oriented Design. <sup>[5]</sup> React Components are similar to Classes in Java, but can be even more granular to allow for their reuse throughout a website or app. One of React's key features is its ability to respond dynamically and rebuild pages based on the user's interaction with the page. A core benefit of developing applications with React is that React supports cross-browser development. The SCR extension is implemented using the React framework.

### 2.2.3. JIRA

Atlassian's JIRA is a ticketing based system, with support for bug tracking, issue tracking, and other project management functions. <sup>[6]</sup> Shape Security uses JIRA to assign tasks and issues found by QA

to the engineers that are able to fix them. The SCR extension is integrated with JIRA to reduce time spent by users switching between applications.

## 3. Design

This section discusses the problems being tackled with this project and the requirements that were developed in order to tackle those problems. Additionally, this section discusses the overall system overview and how our tools fit into the existing infrastructure at Shape Security.

### 3.1. Problems and Requirements

In 2016, a prototype of the session capture tool was created within Shape. The tool was a proof of concept that lacked important functionality, such as data legibility, field validation, and integration with other internal infrastructure. The state of the tool made it unusable by employees of Shape. Our new SCR extension tackles a number of problems within Shape's internal infrastructure.

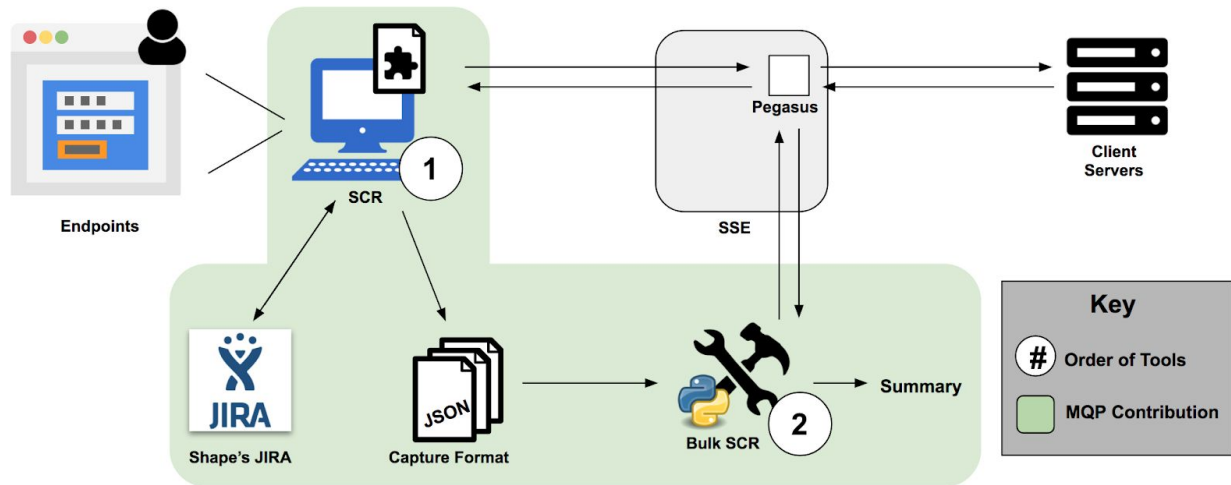
At the forefront of the problem, Shape Security lacked a way to quantify the validity of the replayed capture. One could not easily compare the flags returned from the replayed capture against the original capture. This is an issue when evaluating individual sessions and when evaluating large amounts of historical data. Previously, those wishing to study the results of policy changes needed to deploy the policies live and inspect what decisions Pegasus made on incoming traffic. A requirement for this project was to make an easy-to-use front end to visualize the validations.

Additionally, a major requirement for this extension was the ability to validate the responses of the traffic as it flowed through Pegasus. These validators allow a user to test the expected state of behavior of Pegasus running with different DEX policies. DEX is an internal programming language at Shape used to configure rules on different client endpoints to control the flow of traffic. When Shape detects bot traffic or malicious behavior they can manipulate the rules of a DEX policy to block it from the origin based on factors such as the attacker's browser and system environment. This allows Shape to leverage the information in the attacker's environment to create a DEX policy that blocks the attacker without blocking legitimate users.

Lastly, the SCR extension required JIRA integration to accelerate the debugging process. This enables those working with SCR to have quicker access to Shape's internal ticketing system. JIRA integration includes ticket creation as well as uploading and downloading attachments from existing tickets.

### 3.2. Design of System (System Overview)

Our project is comprised of two tools that integrate into the current infrastructure at Shape. On a high-level, Figure 1 below describes how our project integrates to Shape’s infrastructure.



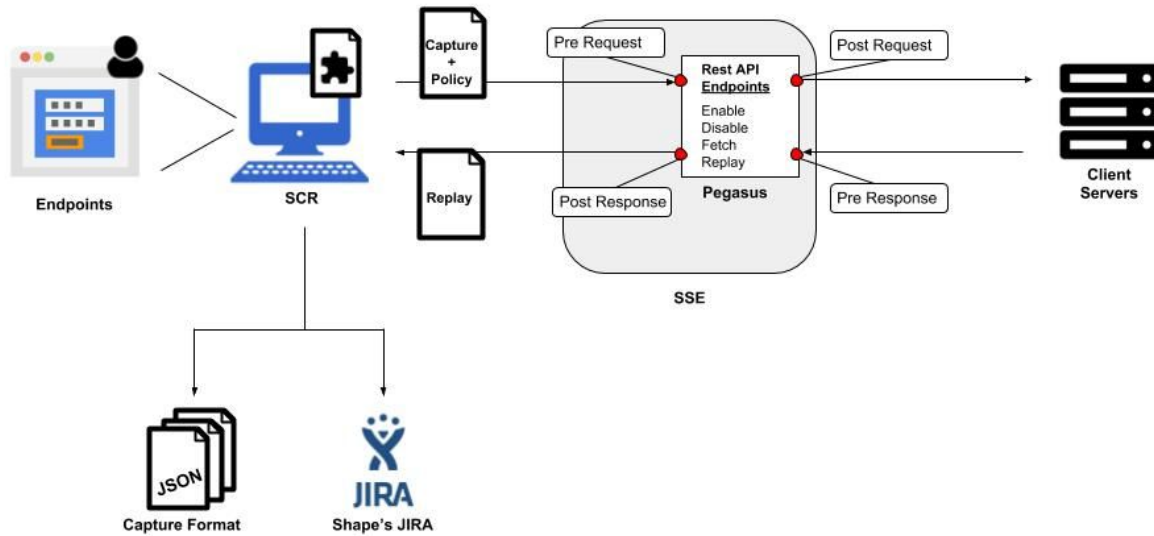
**Figure 1: System Overview**

Tool 1 is the SCR extension, which sits between client endpoints and Pegasus. The extension interacts with Pegasus to have Pegasus capture and replay live session data generated from the SCR extension. The extension then outputs data after validation in a capture format and allows the user to upload or download information to Shape’s internal JIRA. Tool 2 is the Bulk SCR script which takes multiple capture files, executes replay on those files, executes validation, and then outputs results in a hierarchical summary from an individual entry level to a wholistic summary of all the captures.

### 3.3. Design of the Session Capture Replay (SCR) Extension

Figure 2 is an overview of the SCR extension and how it interacts with Pegasus. Pegasus exposes various API endpoints which the extension calls to enable, disable, fetch and replay web request data. When a request with certain flags passes through Pegasus, Pegasus collects information at four different states, pre-request, post-request, pre-response, and post-response. The four states are labelled on Figure 2. When the SCR extension fetches session information, Pegasus returns data from the four states. Two of the four states, post-request and post-response, help users of the extension validate the behavior of Pegasus and verify that Pegasus is setting the correct flags. The SCR extension retrieves the replay from

Pegasus and visualizes the results. Additionally, the extension allows the user to dump the results to a JSON capture format as well as create and upload the capture to JIRA.



**Figure 2:** SCR extension Overview

### 3.4. Design of the Bulk Session Capture Replay script (Bulk SCR script)

The Bulk SCR script was the secondary requirement for this project. Figure 3 is a high-level overview of how the Bulk SCR script ties in with the existing infrastructure at Shape. While the SCR extension is used to replay and analyze one capture in detail, the Bulk SCR script is built to run millions of archived session captures on new policies for high level analysis. The goal is to eventually convert historic transactions stored in Telemetry to the capture format so that previous transactions can be replayed through the Bulk SCR script. The report generated by Bulk SCR gives Shape an idea of how successful a new policy might be.

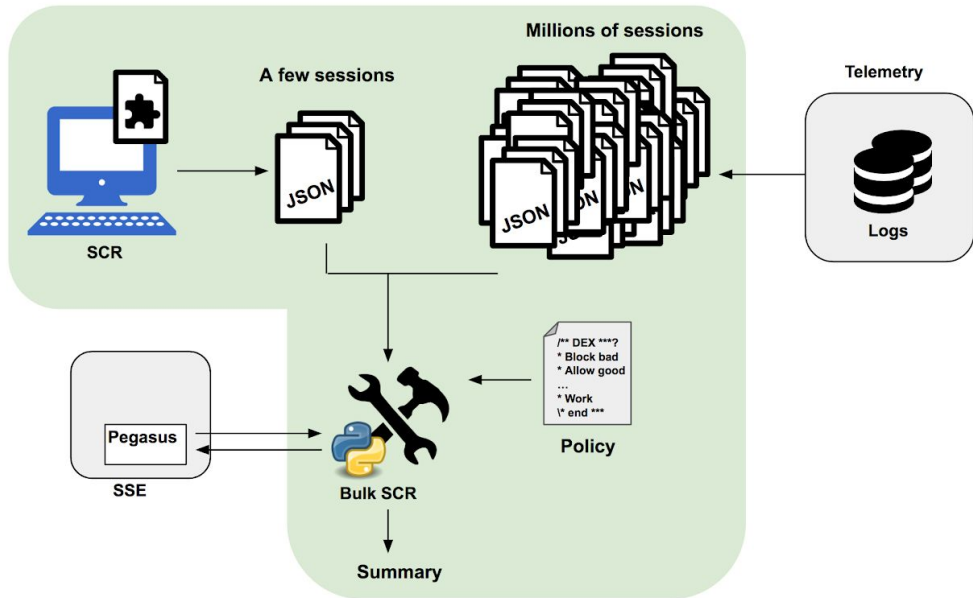


Figure 3: Bulk SCR script Overview

## 4. Implementation of the SCR extension

The SCR extension is a Chrome Extension built with HTML, CSS, and JavaScript using a React framework. The SCR extension is integrated with Pegasus which enables it to capture and replay traffic that passes through Pegasus. The extension helps to visually evaluate the effectiveness of Pegasus by running validations on the web request fields after they pass through Pegasus. The following sections describe the internal workings of the extension and external packages that support the SCR extension.

### 4.1. Axios

Axios is a node module used as a promise-based HTTP client for the browser and node.js.<sup>[7]</sup> A node module is a JavaScript library that encapsulates a single block of related code.<sup>[8]</sup> The module provides a promise-based interface over the regular JavaScript methods for handling XMLHttpRequests (Asynchronous JavaScript and XML (AJAX) calls). A Promise represents the eventual result of an asynchronous operation and is a placeholder into which the successful result value or reason for failure will materialize.<sup>[9]</sup> Axios is used to make network requests to Pegasus and JIRA endpoints via the SCR extension.

### 4.2. JSON Tree Viewer

The SCR extension shows the capture and replay entries in a collapsible JSON format with all the keys and values laid out in a tree format. We found and modified an existing node module `react-json-view`.<sup>[9]</sup>

The following two sections discuss the modifications made to the node module and how the changes were published to Shape's Internal Node Registry.

#### 4.2.1. Modifying Existing Module

The existing `react-json-view` module did not offer functionality to meet the project's requirements. Table 1 below displays the changes we made to the `react-json-view` node module to meet our project requirements and justifications to why we did so:<sup>[11]</sup>

Iteration	Version Number	Changes	Justification for the Change
Base	1.16.0	<i>Not Applicable</i>	This is the base version at which the node module was forked off
1	1.16.1	Added Click Listeners for various data types to listen for expansion and collapse events	To facilitate the side-by-side JSON View in the extension. Collapsing one side, collapses the respective key (if existent) on the other side.
2	1.16.2	Added Icon Generator to generate icons next to rendered values.	To present the applied validator at the respective key in the JSON Viewer.
3	1.16.3	Added various string expansion-collapsion fixes.	<i>Bug fixes</i>
4	1.16.4	Added string overflow prevention guards.	To prevent long values to overflow outside the bounds of the viewer.
5	1.16.5	Added “Row Class Jackers” to take over the background color for the variable rows rendered.	To present validation errors against respective keys on the JSON Viewer.
6	1.16.6	Added String overflow fixes	<i>Bug fixes</i>
7	1.16.7	Added <code>suppressValueField</code> property to the component	To hide values being rendered for the purposes of the Capture Level Defaults Accordion.

**Table 1:** @shape/react-json-view iterations and corresponding changes

#### 4.2.2. Internal Node Package Registry

To make the changes mentioned in the previous section, we needed to fork the existing `react-json-view` module. However, this raised a problem when new users attempted to use the SCR extension and install the required dependencies. Normally, users run an install command, and all of the dependencies are downloaded from the web. Our version of the library differed from the library hosted online, so anyone attempting to install SCR would have been installing a version of the library that was hosted online without the required modifications. For this reason, it was determined that the modified version would be published to Shape’s Internal Node Package Registry. For future developers wishing to update the `react-json-view` module, the steps can be found below:



1. Make the necessary commits to the `react-json-view` repository.
2. Update `package.json` to reflect the new package version that is intended to be published.
3. Tag the latest commit on the `react-json-view` master branch with the newer version number as updated in `package.json` along with a message describing the changes made in that version.
4. Push all your commits *along with the tags* to the remote `react-json-view` repository.
5. Visit Shape's Internal Jenkins dashboard (<http://jenkins.shpsec.com/>), select SSE, select Release Jobs and then select `react-json-view-release`.
6. Click on "Build Now" on the left, choose a tag to be built and published when Jenkins prompts.
7. Once Jenkins completes, the newer version of the repository can be found on Shape's Internal Node Registry. You can update projects relying on `react-json-view` to point to the newer version that was just published.

### 4.3. Validation

Validation flags are methods that enforce how each field of the capture and replay sessions are compared against each other. The following validations are used in the SRC extension and script:

Ignore: Always ok, regardless of value.

Equal If: Ok if both values are equivalent, else true.

Equal: Ok if key is present in both, and values are equivalent.

Has Key: Ok if key is present in the replayed session's value.

Has Value: Ok if key is present in the replayed session's value, and has non-empty value.

Equal To: Ok if arg is equivalent to the replayed session's value.

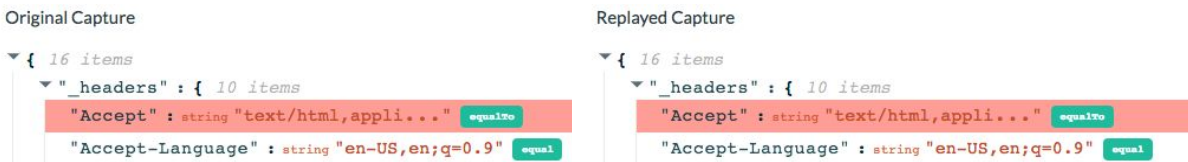
RegEx: Evaluate arg as regEx against the replayed session's value.

The validations are applied and changed by clicking on the validation buttons next to each field of the replay. The validation menu can be seen in Figure 4.



**Figure 4:** Validation Menu for `_bodyTime` field

The results of the validation comparisons provide quantification and clarity to the effectiveness of the new policies. The new policies effectiveness is confirmed by comparing the state of the request/response from the original capture with the state of the request/response after it is replayed. This is driven by the seven validations that are run against each field in the replay JSON when a capture is replayed through Pegasus. A *validation* is designed to verify that the capture and replay meet certain defined criteria for each key present in either one of the capture or replay. A validation is driven by a `Validator` which is comprised of a `ValidatorOption` and an optional argument. `ValidatorOption` represents that operation that must be performed against each key of the capture and replay and returns a `ValidatorResult` indicating if the validation was successful. Figure 5 shows how the validation results can be visually viewed on the SCR extension.



**Figure 5:** Validation where the `Accept` field failed the validation whereas the `Accept-Language` field passed the validation

Within the extension, there are additional visual cues to help pinpoint errors within the validation process. On the entry level, a user can see if the entire entry was successful or not by its icon. For individual keys of interest, a count of the number of errors is displayed next to the key which contains any failed validations.



**Figure 6:** Validation Summary displayed in the SCR Chrome Extension

Figure 6 shows the visual summary provided by the extension for finding validation errors. Each request entry is shown on the top level accordion titled "Request #" followed by the name of the resource requested. Each entry in the top level accordion shows the various keys of interest from the replayed capture, namely: post-request and post-response. If any of the request entry accordions have errors, the error is propagated to the top level accordion with a red cross to the right. If all of the request entry accordions are free of errors, the higher level accordion displays a green check to the right.

#### 4.3.1. Hierarchical Validation States

To visually display Validators, Validation Results, Error Count, the SCR extension introduces the concept of a "validator package". Each validator package corresponds to a single capture entry and the corresponding replay entry (if the capture was replayed). Each validator package is a JavaScript object with the following keys:

1. `validator` : A JavaScript object defined as:
  - **Keys:** A key of interest (“\_postRequest” or “\_postResponse”)
  - **Values:** A JavaScript object with keys as JSON-stringified namespaces and values as the corresponding validators. The object contains keys that are a union of the namespaces of the capture and the replay.
2. `updateFunction`: A function when called with a key of interest and namespace returns another function that accepts the new validator to be updated at the corresponding position.
  - The function must be of the form:

```

updateFunction = (keyOfInterest, namespace) => {
  return (newValidator) => {
    // Do something with newValidator
  }
}

```

**Code Snippet 1: Validator Package** updateFunction

3. `restoreToDefaultsFunction`: A function when called with a key of interest and namespace returns another function that accepts no parameters which restores the validator at the corresponding spot to the default validator.

- The function must be of the form:

```

restoreToDefaultsFunction = (keyOfInterest, namespace) => {
  return () => {
    // Restore validator to default
  }
}

```

**Code Snippet 2: Validator Package** restoreToDefaultsFunction

- The default validator may be inherited from:
  - i. the Capture Level Default validator at the key of interest and namespace
  - ii. the default Capture Level Default Validator as inherited from the uploaded capture
  - iii. the plugin level extension default as provided in the Settings (defaults to `equalIf`)

4. `validationResults`: A JavaScript object defined as:

- **Keys:** A key of interest (“\_postRequest” or “\_postResponse”)
- **Values:** A JavaScript object with keys as JSON-stringified namespaces and values as the corresponding `ValidationResult`. The object contains keys that are a union of the namespaces of the capture and the replay.

5. `errorCount`: A JavaScript object defined as:

- **Keys:** A key of interest (“\_postRequest” or “\_postResponse”)
- **Values:** An integer representing the number of errors based on the `ValidationResults`.

6. `validationClasses`: A JavaScript object defined as:

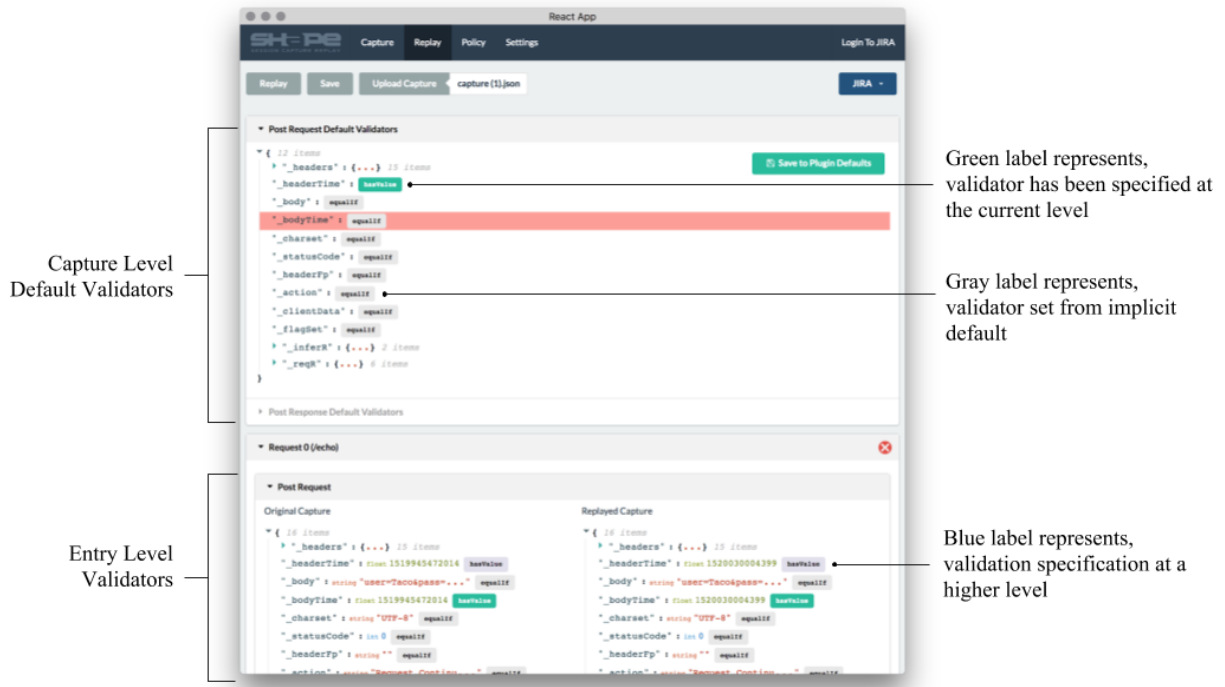
- **Keys:** A key of interest (“\_postRequest” or “\_postResponse”)

- **Values:** A JavaScript object with keys as JSON-stringified namespaces and values as the corresponding classes to be applied to their variable rows. The object contains keys that are a union of the namespaces of the capture and the replay.

Validator packages are generated every time:

1. a new capture is fetched from the Capture tab (described in Section 4.5); or,
2. a new capture is uploaded to the Replay tab (described in Section 4.6); or,
3. a replay was executed on an existing capture from the Replay tab (described in Section 4.6)

The validators can be set at three different levels, namely: implicit default, capture level default, and entry level. An implicit default validator can be set from the Settings Tab. Every field in an entry that does not have a validator specified to it will fall back on the implicit default. `equalTo` is the default implicit default validator for the SCR extension. Validators can also be set globally on fields, using the Capture Level Default Validators accordions on the Replay Tab. Validators set at the Capture Level will apply to the specified field for every instance throughout the capture, unless one has been specified at the Entry Level. Entry Level validators have the highest preference. The validation hierarchy can be visually seen in Figure 7.



**Figure 7: Validation Hierarchy Overview**

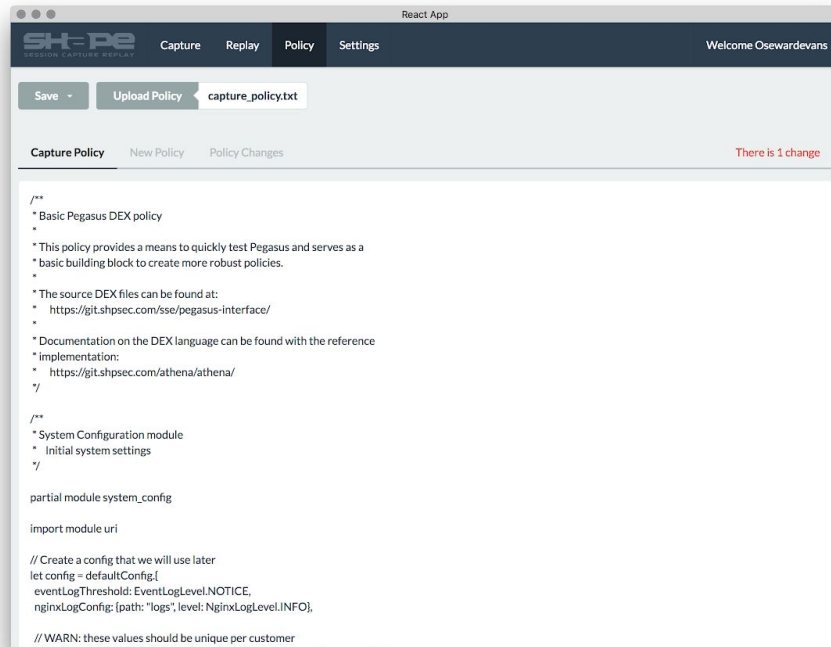
#### 4.3.2. Local Storage Interface

With respect to the Validators, Chrome's Local Storage is used to store the plugin's:

1. implicit default validator; and,
2. capture level default validators for various keys of interest (“\_postRequest” or “\_postResponse”)

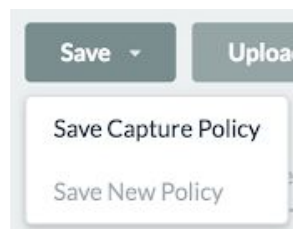
The Local Storage Interface, its implementation, uses and purposes are further expanded on in Section 4.8.4.

## 4.4. Policies Tab



**Figure 8:** SCR extension - Policy Tab

Policies are the filter rules applied to Captures and are the instructions that tell Pegasus whether or not to block certain traffic. All interactions with policies are conducted on the Policy tab as shown in Figure 8. Here a user is able to save/upload policies as well as compare the line changes between the new and old policy. When a user uploads a new policy, the policy is attached to the current capture in the extension. Any subsequent replays of that capture via the Replay Tab will be run with the newly uploaded policy. The Save button as shown in Figure 9 allows the user to save either the Capture Policy or New Policy as a text file.



**Figure 9:** SCR extension - Policy Save Button

Another feature of the Policy tab is the ability to explicitly view changes in a policy. When manipulating a policy to invoke different flags results, the extension highlights the lines that have changed in the policy. Figure 10 below shows how the Policy Diff Viewer highlights differences between two policies.

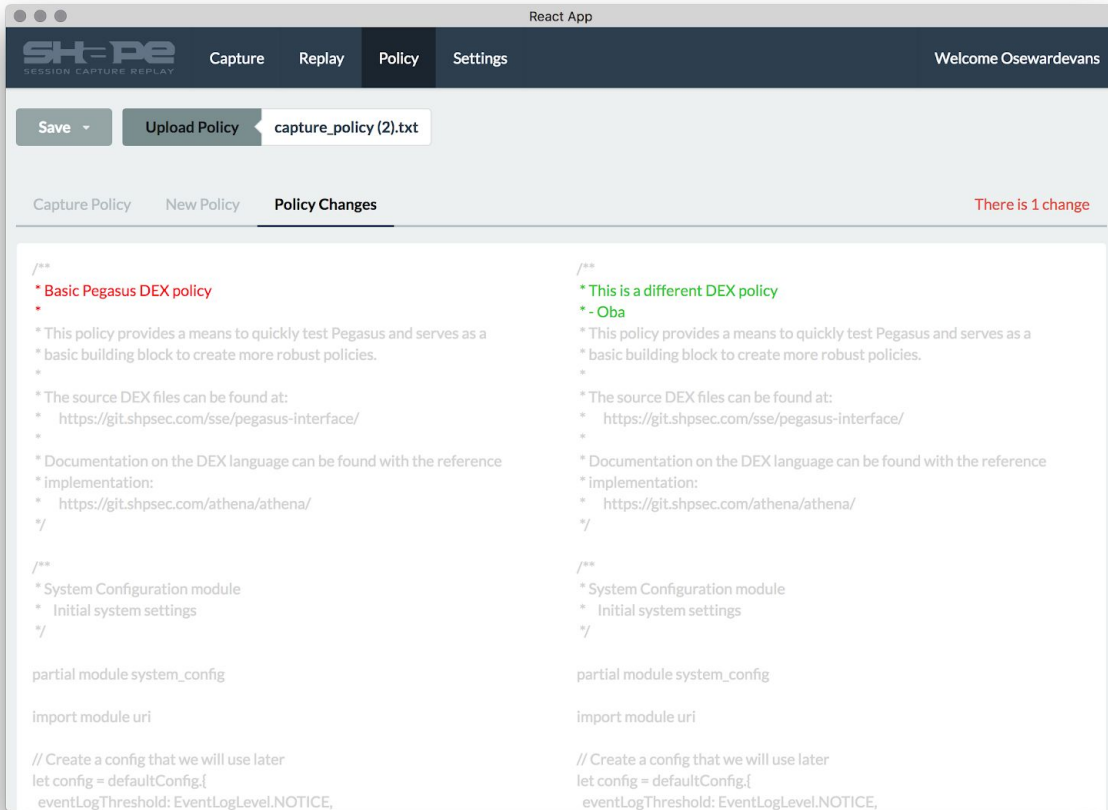
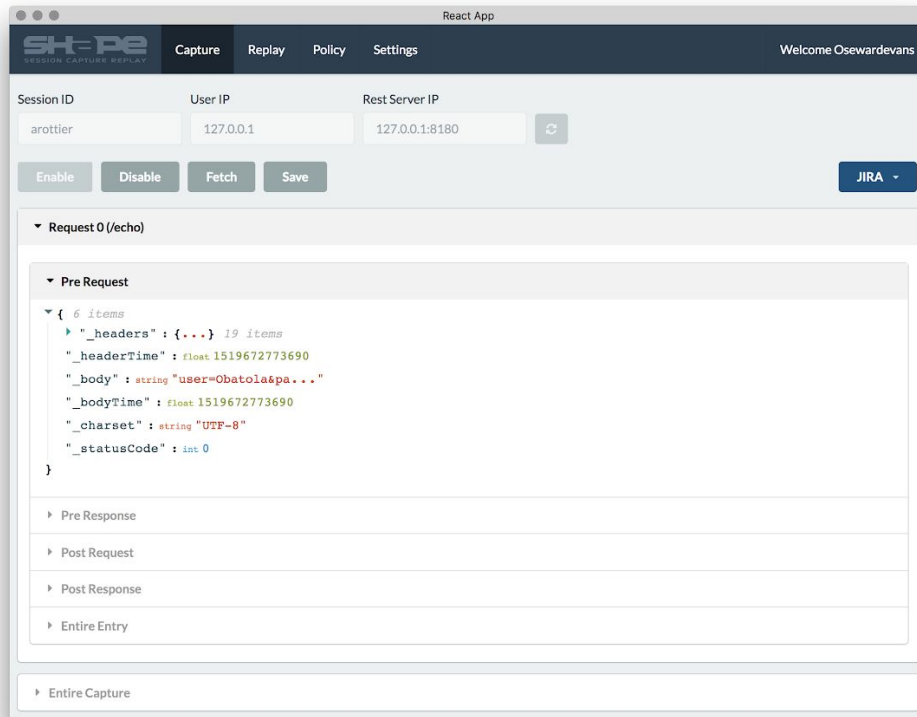


Figure 10: SCR extension - Policy Diff Viewer



## 4.5. Capture Tab



**Figure 11:** SCR extension - Capture Tab

From the Capture tab, users are able to configure the Session ID, User IP, and Rest Server IP. The Session ID is attached as a header to all outbound requests from Chrome. The User IP is the current user's IP address. The combination of the Session ID (sent as a header) and the user IP address requesting for the resource signal whether Pegasus should collect information about the transaction (request and response). The Rest Server IP is the IP address where the Pegasus instance is running and where the extension will communicate with to enable, disable, fetch or replay a session capture. Figure 12 shows the above described part of the Capture tab.



**Figure 12:** SCR extension - Capture Inputs

Below these inputs are the buttons that interact with Pegasus's REST API. Clicking the Enable button triggers a two-fold process:

1. Perform a GET request to Pegasus running at the Rest Server IP along with the Session ID and User IP indicating that session capture must be enabled
2. Attach a Chrome web-request hook for all outbound requests from Chrome that adds an additional header with the name `x-shpsec-session-id` and the value being the Session ID

The Fetch button is disabled by default and is enabled after session capture is enabled via the extension. Clicking the button performs a GET request to Pegasus running at the Rest Server IP along with the Session ID and User IP to retrieve all the transactions that were captured by Pegasus. The UI is updated with the transactions returned from Pegasus.

The Disable button is disabled by default and is enabled after session capture is enabled via the extension. Clicking the button triggers a two-fold process:

1. Perform a GET request to Pegasus running at the Rest Server IP along with the Session ID and User IP indicating that session capture must be enabled
2. Remove the Chrome web-request hook for all outbound requests attached when the Session Capture was enabled by clicking the Enable button

The Save button is disabled by default and is enabled after a capture was successfully fetched via the extension from Pegasus. The button saves the capture data into a JSON file **without** any of the validators unlike the Replay tab, which is discussed in the following section.



**Figure 13:** SCR extension - Capture Buttons

When data is fetched from Pegasus, it is displayed on the page. Requests are displayed sequentially in the order that they are made with the name of the resource requested. Data from these requests can be viewed at each state it was captured through Pegasus. Figure 14 shows a sample of data that was returned from the fetch REST call.

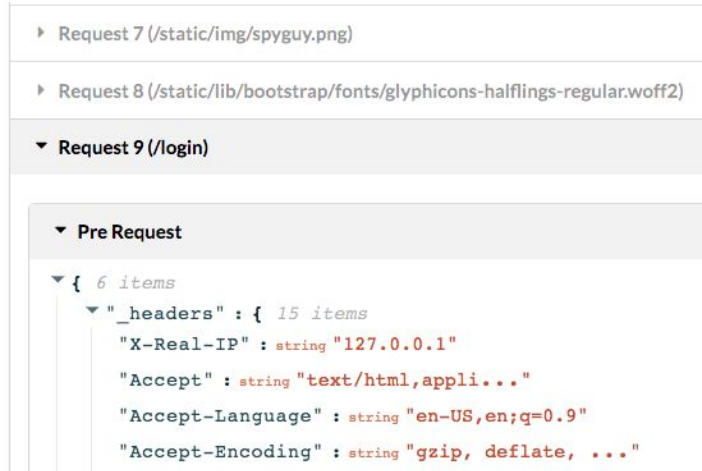


Figure 14: SCR extension - Capture Resources

## 4.6. Replay Tab

From the Replay tab, users can run existing captures through the replay endpoint of Pegasus. Clicking the Replay button performs a POST request to Pegasus running at the Rest Server IP with the Session ID and User IP as query parameters to the endpoint and the capture as the request body.

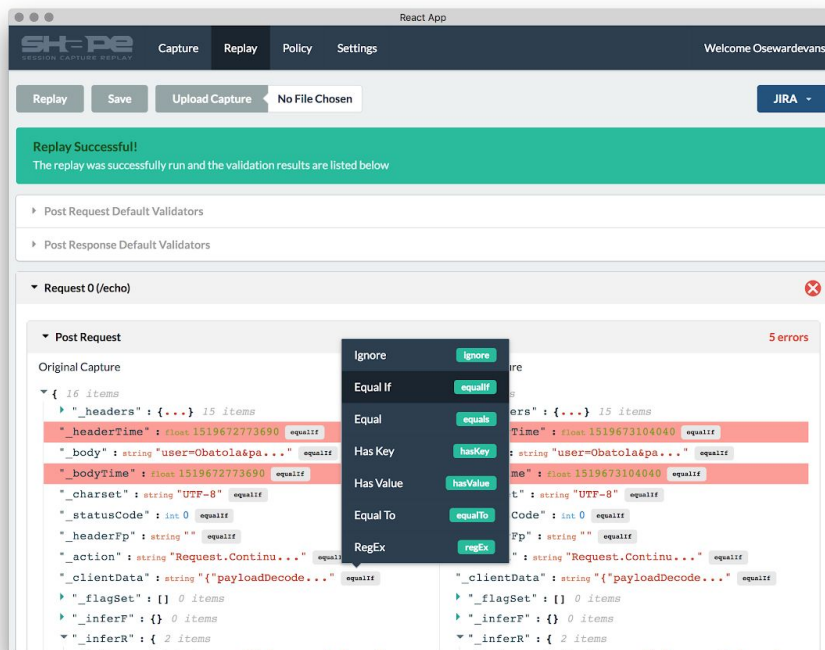


Figure 15: SCR extension - Replay Tab

## 4.7. JIRA Integration

The SCR extension is integrated to work with Shape’s internal ticket handling system, JIRA. Access to the extension’s JIRA interface can be found on the Capture and Replay tabs. Internally, Shape uses JIRA to handle new features, issues and bugs with it’s products. When an issue is discovered, QA files a ticket with the information regarding the issue. SCR’s direct integration with JIRA is particularly useful as an engineer can download and upload captures from JIRA without ever leaving the extension.

### 4.7.1. Login

A menu item to the right titled “Login To JIRA” is visible to the top right of the extension if the user is not logged into JIRA. The login tab has a button that opens a new window landing on the JIRA homepage. If the user is not logged in, JIRA will redirect to Shape’s Okta website to facilitate the login process. When the user logs in, the JIRA homepage is presented. Upon closing the window, the extension polls if the user is logged in. If the login was successful, the top right menu button now shows the logged in user’s JIRA username. Figure 16 demonstrates the workflow of the extension’s JIRA Login functionality.



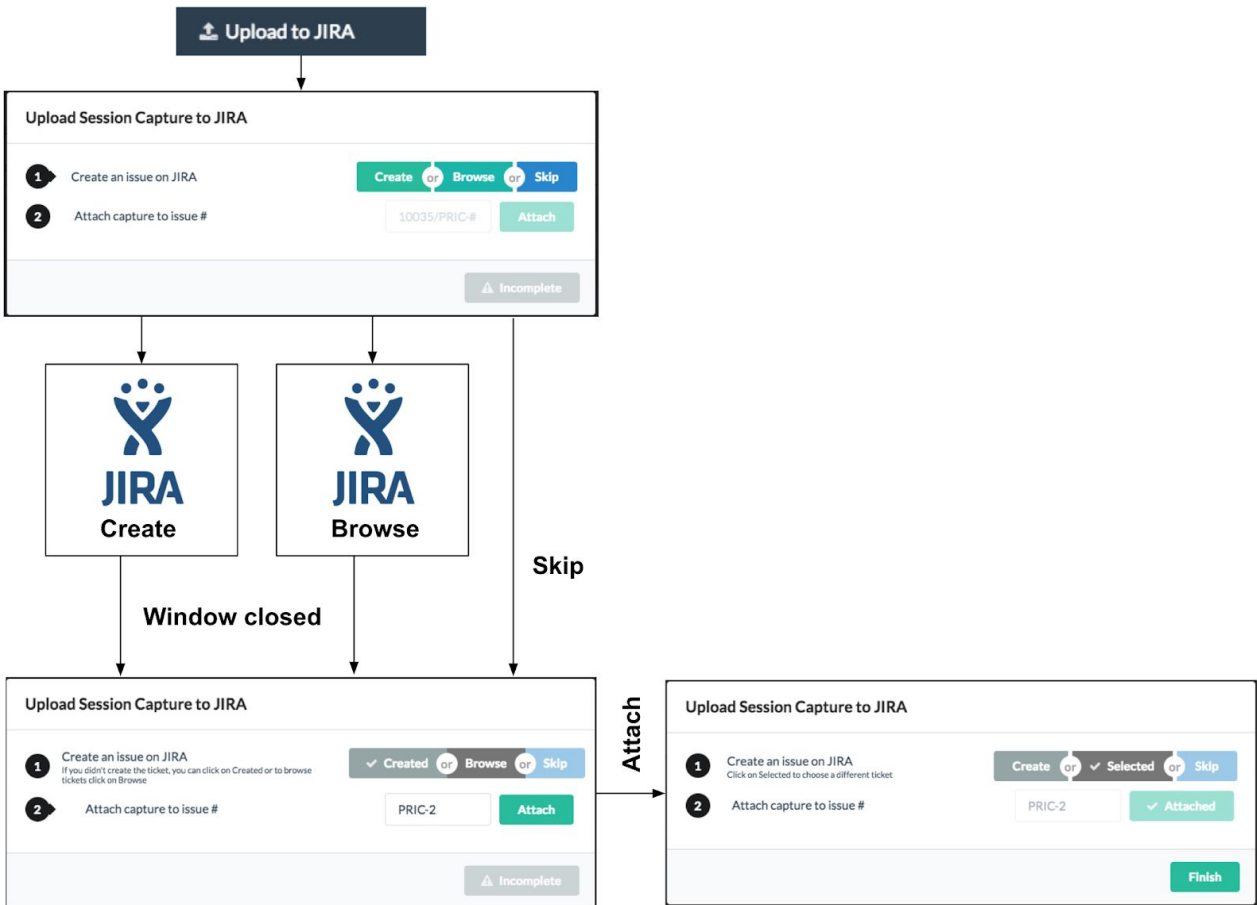
**Figure 16:** SCR extension - JIRA Login Process

### 4.7.2. Upload

With the extension’s JIRA Upload functionality, a user can:

1. Create a new issue and upload a capture to the new issue; or,
2. Browse an existing issue and upload a capture to that issue; or,
3. Directly upload a capture to an existing issue with knowledge of the issue number

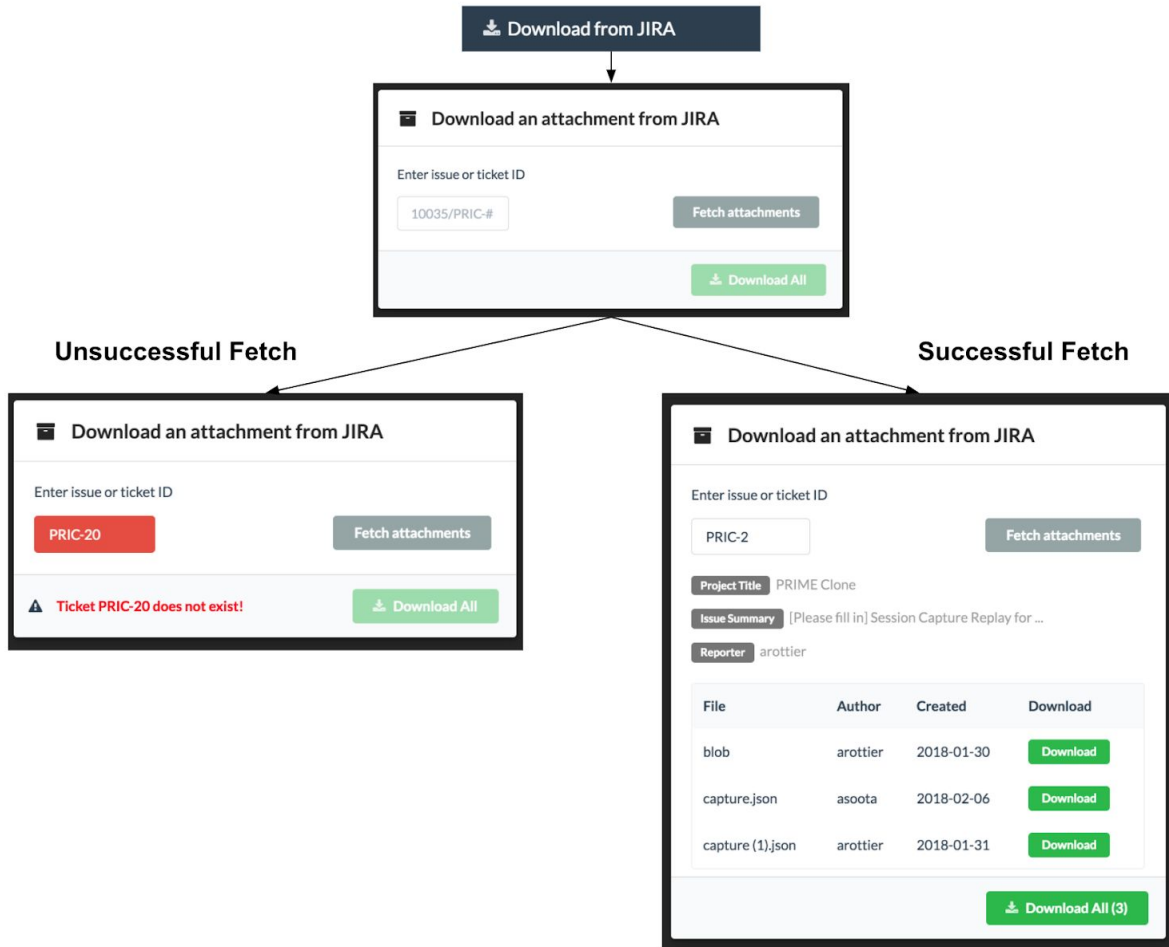
By clicking the “Attach” button the user can upload the capture to the stated issue number. The capture is attached as a JSON file to JIRA. Figure 17 demonstrates the workflows with the extension’s JIRA Upload functionality.



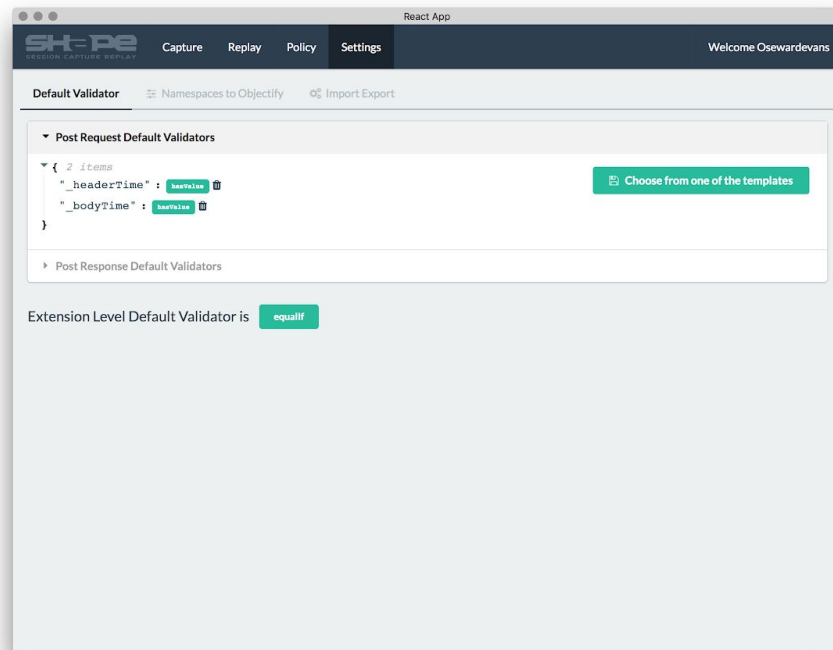
**Figure 17:** SCR extension - JIRA Upload Capture to an issue process

#### 4.7.3. Download

With the extension’s JIRA Download Functionality, a user can directly download a capture attached to a given issue number from JIRA. Figure 18 demonstrates the workflow of SCR extension’s JIRA Download functionality.



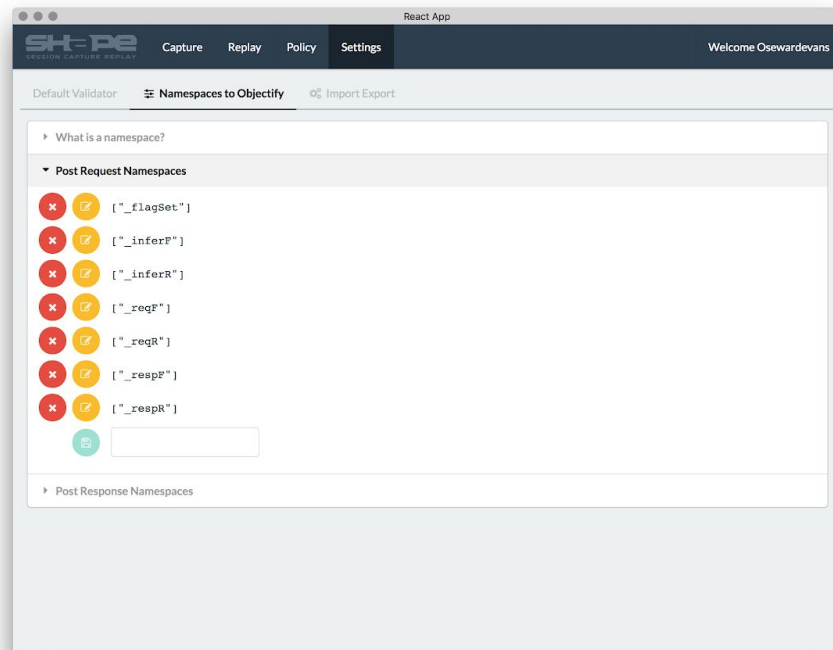
## 4.8.1. Default Validators



**Figure 19:** SCR extension Settings Tab - Default Validator Sub-tab

The Settings tab allows for configuration of the Default Validators for the keys of interest `_postRequest` and `_postResponse` as shown in Figure 19. These validators are picked up by the Replay tab to populate validators. The Default Validator Settings Sub-tab allows for configuration of these Default Validators. The Implicit Plugin Default Validator can also be configured from this tab. The Default Validators sub-tab also allows for choosing default validators from templates hardcoded in the extension. Once the user chooses a template, the user gets a choice between merging the template with existing validators and replacing the template with *all* the existing validators.

## 4.8.2. Namespaces to Objectify

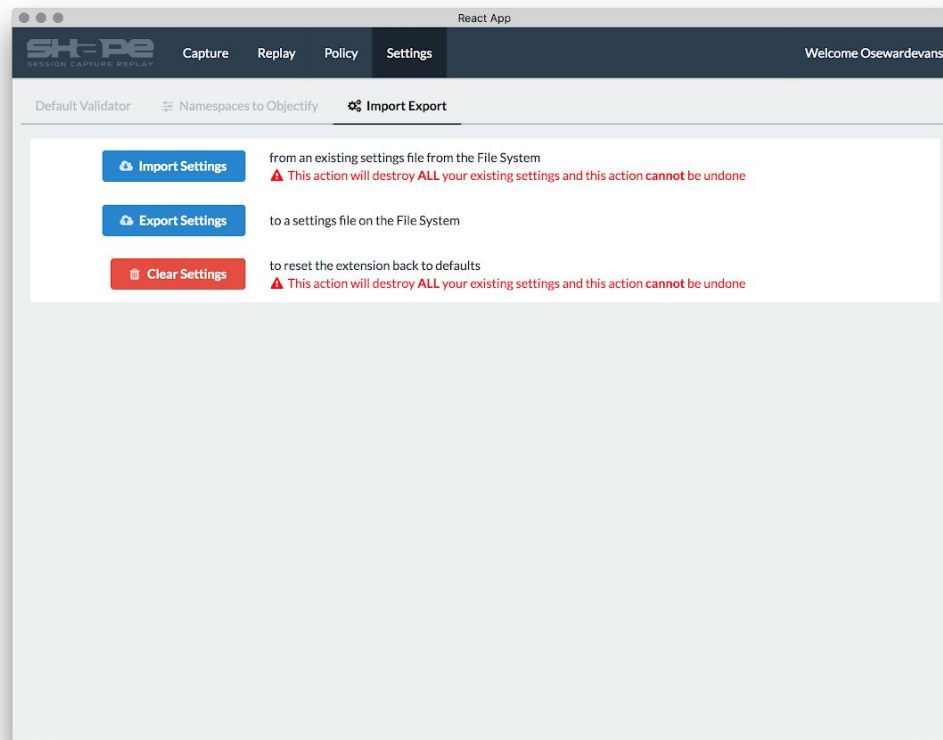


**Figure 20:** SCR extension Settings Tab - Namespaces to Objectify Sub-tab

There are various fields in the response from Pegasus that come as stringified-JSONs. Comparing such JSON strings could be difficult visually. The Namespaces To Objectify Settings tab allows the user to configure the namespaces which they are want a more detailed analysis of as shown in Figure 20. When a Capture or Replay is received by the Replay Tab, the extension decomposes the strings back to JSON format for easier comparisons. The Namespaces to Object Settings sub-tab allows the user to add, modify and delete response fields that they would like SCR extension to validate on. Nested fields within response flags can be validated on when the namespace is listed in Namespaces to Object Settings sub-tab.



### 4.8.3. Import Export Settings



**Figure 21:** SCR extension Settings Tab - Import Export Sub-tab

The Import Export Settings Sub-tab allows the user to import, export or clear out their extension settings as shown in Figure 21. These settings maintain properties of the extension regarding default validators, the namespaces to validate on, and other settings within the extension. The “Import Settings” button allows the user to upload extension settings to the extension. Clicking “Import Settings” button triggers a two-fold process:

1. Open a File Selector Dialog allowing the user to choose a settings file from the File System (FS)
2. If the file is a valid JSON file, the extension’s Local Storage is *cleared* and replaced with contents from the file selected

The “Export Settings” button allows the user to download all extension settings to a file on the File System. Clicking the “Export Settings” starts a download of a JSON file with contents extracted from the extension’s Local Storage.

The “Clear Settings” button allows the user to restore the extension settings back to defaults. Clicking the button will clear the extension’s Local Storage.

#### 4.8.4. Chrome Local Storage Interface

All of the extension’s settings and configurations are stored in Chrome Local Storage. To facilitate synchronous Local Storage get and set calls, we designed a class `LocalStorageInterface` as a Singleton to be a single source of truth datasource. All interactions with Chrome’s Local Storage throughout the extension take place through this class.

### 4.9. Testing

This section contains the various test frameworks and libraries used to implement unit tests and report code coverage for the SCR extension.

#### 4.9.1. Mocha

Mocha is a feature-rich JavaScript test framework that runs on Node.js making asynchronous testing simple. <sup>[12]</sup> Mocha was used to unit test most almost all the JavaScript files in the source backing the SCR extension. Each file tested with Mocha was given its own test suite named after the filename or the React component being tested.

#### 4.9.2. Istanbul

Istanbul is an extensive test coverage library built to work with Mocha. <sup>[13]</sup> Istanbul has the ability to track statement, branch, functions, and line coverage of tests and provides both console and HTML output. Figure 22 shows example console output for test coverage results from Istanbul for the SCR extension.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	85.54	72.43	78.03	86.23	
src	100	100	100	100	
Utilities.js	100	100	100	100	
src/components	93.52	58.33	100	93.33	
InputContainer.js	100	100	100	100	
Main.js	98.14	55.56	100	98.14	... ,98,100,102
SessionFileInput.js	100	66.67	100	100	13,77

**Figure 22:** Istanbul Test Coverage for the SCR Chrome Extension

#### 4.9.3. Sinon

Sinon is a library that helps with JavaScript testing. Sinon provides test spies, stubs, and mocks for JavaScript that work with any testing framework. <sup>[14]</sup> Many functions within the SCR codebase rely on the response from other functions and classes. Sinon provides the capability to mock the behavior and return value of the dependency functions to simplify testing. Sinon is also particularly useful to mock external behavior, like results from a network call, database call or a File I/O. Sinon helps isolate and test the core functionality of a method rather than redundantly testing dependency functions.

#### 4.9.4. Chai

Chai is a Behavior-Driven Development (BDD)/Test-Driven Development (TDD) assertion library for node that can be used with any JavaScript Testing Framework. Chai provides several interfaces to facilitate testing like the `should`, `expect` and `assert`. <sup>[15]</sup> For unit-testing the codebase of the SCR extension, we chose to use the `assert` interface.

#### 4.9.5. Enzyme

Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate and traverse the rendered React DOM in a unit-test environment. <sup>[16]</sup> Enzyme contains its own set of assertions that work with parsing DOM elements for specific values.

## 5. Implementation of Bulk SCR Script

The Bulk SCR script is a Python script that allows users to replay one or more captures sequentially in bulk with new policies and inspect the validation results. The Bulk SCR script is integrated with Pegasus which allows the script to replay existing captures through Pegasus enabling Shape to evaluate the effectiveness of new policies. This section describes the internal workings of the Bulk SCR script.

### 5.1. Invocation

The script must be invoked through the command line. To invoke the Bulk SCR script, the user must run `python Main.py` in the root directory where the script resides. By default the script will:

- Replay each capture placed in a folder named `captures` in the same directory as the script with each policy placed in a folder named `policies` in the same directory as the script
- Output high level results to the console

The execution of the script can be customized to the user's needs by passing additional arguments:

- The argument, `--captures`, dictates the directory from where the captures to be replayed are picked up. The default is a directory called `captures`.
- The argument, `--policies`, dictates the directory from where the policies to run against the captures must be picked up. The default is `policies`.
- The argument, `--log`, dictates the script to output the analyzed results in the console. `--log` is the default argument, and will be enforced, unless `--quiet` is invoked.
- The argument, `--quiet`, quiets default logging, from `--log`, that is outputted in the console.
- The argument, `--raw`, dictates the script to output the raw replayed captures as JSON files into the `output/raw` directory relative to the location of the script. The raw replayed captures can then be used by other Shape teams for more individual analysis.
- The argument, `--json`, dictates the script to output a JSON representing the results.
- The argument, `--html`, dictates the script to output the results JSON same as the `--json` argument, but in a HTML file where with analysed results for each policy.
- The argument, `--metrics`, dictates the script to compute and output the total time it took the script to execute.

## 5.2. Replay Integration

In order to replay captures in the Bulk SCR script, we use the Python library `requests` to send the capture with the new policy to the Pegasus server and receive the replayed capture. Replay requests are handled sequentially, one at a time. The responses from Pegasus are then handled by the validation phase, described in the Section 5.3.

## 5.3. Validation

Validation of the capture and replay entries are handled similarly to that of the SCR extension. The validators are used as described in capture file that is being replayed. If validators are missing for a certain field/fields in the capture, the script falls back to the default validator `equalIf`. Refer to Section 4.3 for a detailed explanation on how the validation is executed and how the Validation Results are handled.

## 5.4. Validation Results and Script Summary

Running Bulk SCR script with the arguments, `--raw`, `--log`, `--html`, or `--json`, determines the format in which validation results are outputted. The results are computed at all levels, including individual fields validations, summaries for each individual capture, and a total summary for the entire load of captures. Using the `--json` argument, a JSON representing the collective results is created with a sample structure shown in Appendix B. Using the `--raw` argument, outputs the results JSON as a file, while `--log` creates a text summary of the results JSON. A snippet of what is logged in the console is shown in Figure 23. To keep the size of the summary short, the script only reports information about the captures, entries and fields that failed validation.

```
Results For Policy: name_of_policy.txt
  1 of the 1 captures failed validation.

Results For capture: capture_name.json
  1 of the 1 entries failed validation.

Failed Results For Entry 0:
  Post Response
    1 field failed validation

    ['parent_field_name`, `field_name`] failed with validator 'operation'
```

**Figure 23:** Example of console output by the Bulk SCR script

## 6. Evaluation and Results

Since SCR is a new feature for Shape, evaluation of the project is based on features created. This section encapsulates the evaluation of our project including performance metrics as well as the new baselines this application introduced.

### 6.1. SCR Validation

With the prototype SCR extension, users were only able to view response data in plaintext. The new SCR extension renders response data in hierarchical tree format while pairing each field of the data to the corresponding validator. Validation of individual fields on a graphical interface is a new feature for Shape Security. Figure 24 below shows an abstraction of the improvement made to the prototype.

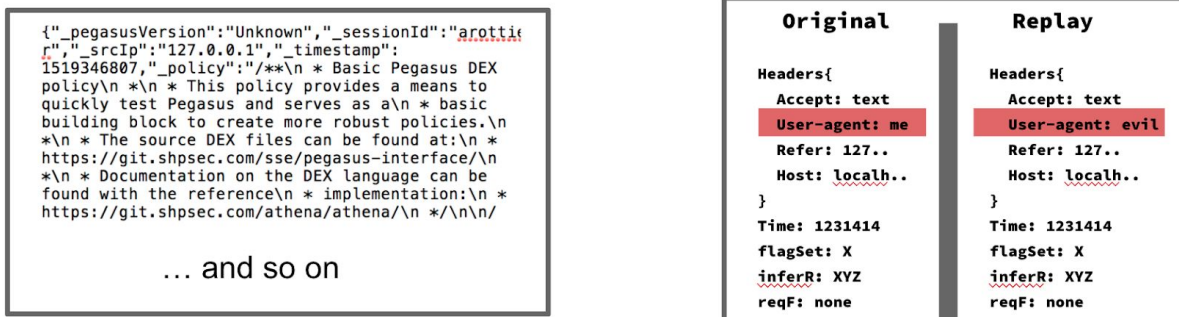
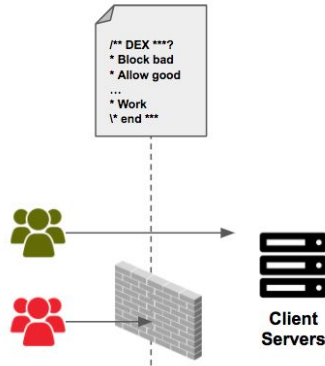


Figure 24: SCR extension UI and Validation Improvements; left (original), right (new)

### 6.2. Policy Effectiveness

One of the advantages of Session Capture Replay technology is that it quantifies the effectiveness of a new policy. On an individual capture replay, a user can validate specific behavior of how a policy deployed to Pegasus interacts with web traffic. On a large scale, millions of previous transactions can be replayed, validated and summarized. These summaries hold quantifiable statistics as to the effectiveness of a new policy. This helps Shape make more informed decisions when it comes to policy deployment for their clients. Figure 25 shows what an effective policy should do on a high level.



**Figure 25:** Abstraction representing Policy Effectiveness

### 6.3. JIRA Integration

Since the SCR extension is essentially a debugging programming, users often need to have JIRA open when using this tool. To alleviate time spent navigating between two separate application interfaces, SCR extension directly implements core JIRA features for logging in, creating tickets, and uploading and downloading attachments. This features improves the usability of SCR extension as it allows users to stay on one application during the replay process.

### 6.4. Performance of Bulk SCR

We ran metrics test for the Bulk SCR script, and received results for how long each capture will take running each against each argument. The tests were run on a Mid 2015 MacBook Pro with 2.2GHz quad-core Intel Core i7 processor, 16GB of 1600MHz DDR3L onboard memory (RAM), 256GB PCIe-based flash storage, 802.11ac Wi-Fi wireless networking card running macOS High Sierra (v10.13.1). The tests were conducted by utilizing five duplicate captures, each containing 200 identical entries, in which each entry contained three false comparisons in the post response, and one false comparison in the post request. Time metrics for this process were computed by taking the difference in time from execution to when the process was completed. The metrics were timed in seconds and calculated into ‘hours per million entries’ (hpme), and ‘entries per second’ (eps). The results are as follows:

Argument Used	Seconds To Run	eps	hpme
python Main.py --raw -m -q Best: Worst:	8 9	125 111	2.22 2.50
python Main.py --log -m Best: Worst:	17 19	58.8 52.6	4.72 5.28
python Main.py --json -m Best: Worst:	21 23	47 43.4	5.83 6.40

**Table 2:** Performance Metrics for Bulk SCR

## 6.5. Bulk SCR Output

The Bulk SCR script is able to out replay and validate results in three formats, namely Console Output, JSON, and HTML. These results help Shape determine how effective a policy is. Outputs contain individual unsuccessful validations in the capture files, summaries for each capture, and summaries for the entire bulk replay. The features augment the SCR extension by performing the same validation steps but gives an evaluation for mass amounts of data.

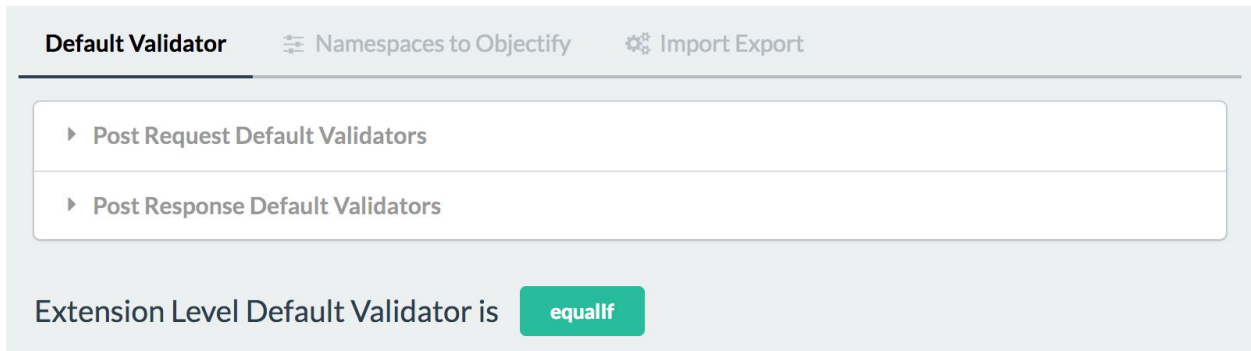


## 7. Future Work

Future SCR extension improvements range from UI to back-end improvements, while the Bulk SCR script could use a major performance boost. Sections 7.1 through 7.3 go in detail about how each tool can be improved.

### 7.1. SCR Extension Improvements

The first improvement would be to implement an implicit default selector to the replay page. This addition would be mimicking the Extension Level Default Validator option that is currently on the Settings Tab for the Replay Tab as seen in Figure 26. Unlike the Extension Level Default Validator on the Settings Tab, the option on the Replay Tab would immediately change the default validation operation for the fields in the capture that has not been specified at the specific capture or post request/response validation phase.



**Figure 26:** Settings Tab: Default Validators and Implicit Default Validator Settings

Another improvement to the SCR extension would be to refactor the code, using a library like Redux to improve state management and facilitate a global storage for application state. Redux works with React, and allows the projects to make state, store, and UI changes using actions that are sent throughout the whole program. <sup>[17]</sup> These actions allow every component to respond on their own, based off of the broadcasted messages. Implementing this would clean up a lot of the information and state propagation that is happening throughout the codebase. In consequence, changes to the code would be easier and more intuitive to implement.

Lastly, our team sent a questionnaire, found in Appendix A, to teams within the Shape that will be using SCR extension. The purpose of this questionnaire was to gather feedback about our application

and document suggestions on how to make future improvements. Responses from the questionnaire were logged into Shape's Github Enterprise repository for our project. Raw feedback from the questionnaires is also located in a Google Drive directory. A link to this can be found in the readme portion of the repository. Future work would include analyzing the feedback received and incorporating the feedback into the extension.

## 7.2 Bulk SCR Script Improvements

For the Bulk SCR script, we encourage future maintainers of the application to improve its performance. Currently, we are reaching a base metric of replaying 125 capture entries per second and running analysis on 47 captures per second. First, we recommend configuring Pegasus to run in quiet mode, which means all console logging would be suppressed. This method would reduce I/O time and improve the turnover rate for Pegasus' replay endpoint.

Second, we recommend finding ways to improve the efficiency of the validation portion of the script. This portion of the script currently accounts for about half of the time taken to complete a bulk replay and validation of capture data. Ideas to improve this include replacing existing, hand written code blocks with Python libraries which proven efficiency metrics. Another way to improve performance during the validation stage is to implement threading to split the workload of validation across multiple cores on the system. The more optimal the Bulk SCR script becomes, the quicker Shape is able to make more informed decisions about effects of new policies.

Another important addition would be to implement code coverage for the Python tests. Currently there are 93 tests running against the Python code, but no way to detect the percentage of the codebase that has not yet been covered.

## 7.3. SCR Ecosystem Improvements

The last major task to improve the Session Capture Replay ecosystem is a tool that can convert Telemetry data into a capture format that can be consumed by the SCR extension and Bulk SCR script for replaying. Since Telemetry stores all of the transactions that go through Pegasus, the ability to replay and validate this data would give further insight on the effectiveness of a new policy against live web traffic data. New policies could be run against historical data which would allow Shape to come up with metrics as to how successful a new policy is going to be. This would help the Threat Mitigation team in Shape make better informed decisions during pre-policy deployment.

## 8. Conclusion

Shape Security's primary defense against automated attacks is Pegasus- a scriptable reverse proxy that sits between the end user and the origin servers. Shape Security identified a need to replay captured Internet traffic through their client servers and validate the response. Our project implemented that functionality on both an individual capture level with the help of the SCR Chrome extension and bulk level with the help of the Bulk SCR script.

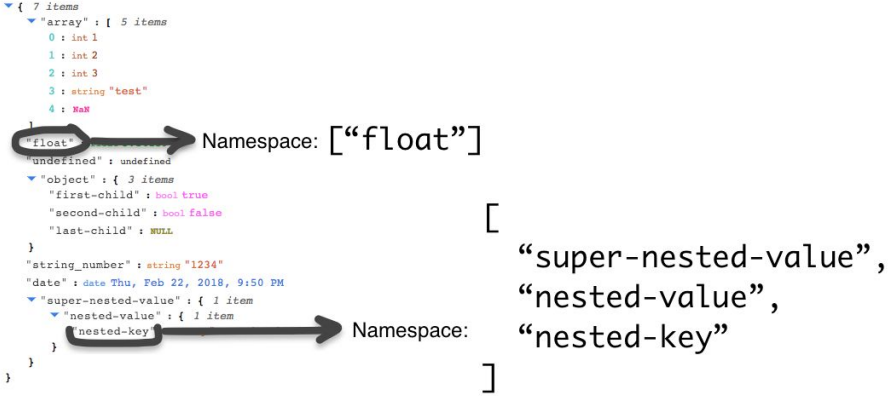
The SCR Chrome extension is designed to allow for a detailed analysis of one capture. The extension is able to do so by replaying a single capture through Pegasus and visualizing validation results. The extension was also designed to integrate well with JIRA.

The Bulk SCR script was a secondary requirement for this project and is designed to run run millions of archived session captures on new policies for high level analysis. The report generated by Bulk SCR gives an idea of how successful a new policy could be.

Shape Security proposed solution has not only been implemented, but is ready to be used internally by Shape employees for debugging, research, marketing, and configuring policies. The SCR extension was written modularly in React so it is extensible for future maintainers of the application. Documentation of future steps and enhancements have been logged in Shape's Github Enterprise for the project so future maintainers can immediately pick up where our team left off. Our code is documented and our test coverage is 86% for SCR extension.

The SCR Chrome extension and Bulk SCR script derives value for Shape Security and various teams within Shape. First, the tools enable QA engineers to more easily test Pegasus and validate policies configurations. Second, the Bulk SCR script helps improve the decision making when deploying new policies. Additionally, Shape's research teams gain a new way to evaluate history data.

## 9. Glossary

Term	Definition
<p>Namespace</p>	<p>A namespace is an list of keys which you traverse down to find a value.</p> <p>For example, a namespace of the form: ["_headers", "User-Agent"] indicates that the value being searched for can be found by</p> <ol style="list-style-type: none"> <li>going to the root of the object; then,</li> <li>going into the value corresponding to the <code>_header</code> key; and finally,</li> <li>choosing the value corresponding to the <code>User-Agent</code> key</li> </ol> <p>Figure 27 provides a visual example for the same:</p>  <p>Figure 27: Visual example explaining a namespace</p>
<p>Mapping</p>	<p>Another term used interchangeably for a JavaScript object, or more loosely a key-value pairing.</p>
<p>Namespace Mapping</p>	<p>Represents a mapping that is flatten-out to just one level with specialized keys that are JSON-stringified namespaces of where the values it in the original mapping. Table 3 provides as an excellent example:</p>

	<pre> {   "_h": {     "U-A": [1, 2, 3],     "n": {       "ival": "abc"     }   },   "key": "value" } --&gt; {   ["_h","U-A"]: [1, 2, 3],   ["_h","n","ival"]: "abc",   ["key"]": "value" } </pre> <p style="text-align: center;"><b>Table 3:</b> Original mapping to namespace mapping example</p>
Key of Interest	Refers to the state of the data in Pegasus. This can be one of four states, pre-request, post-request, pre-response, and post-response.
Policy	Represents a set of customized configurations used to filter traffic to Shape’s client’s websites specified using the DEX configuration language.
True-Depth Mapping	<p>Represents a generic mapping that is not flattened out like the namespace mapping. Table 4 provides an excellent example for going from a namespace mapping to the respective true-depth mapping:</p> <pre> {   ["_h","U-A"]: [1, 2, 3],   ["_h","n","ival"]: "abc",   ["key"]": "value" } --&gt; {   "_h": {     "U-A": [1, 2, 3],     "n": {       "ival": "abc"     }   },   "key": "value" } </pre> <p style="text-align: center;"><b>Table 4:</b> Namespace Mapping to the respective True-Depth Mapping example</p>
Objectify	Refers to the action of converting a stringified-JSON back to its original JSON object. The action is safe meaning if the string is not a stringified-JSON, the string is left untouched and program execution is resumed without any errors.
DEX	<b>Decision Engine Xscript</b> is a proprietary configuration language used to specify the behavior of the Pegasus scriptable reverse proxy.

## 10. References

- [1] How Much Data is Created on the Internet Each Day?, Jeff Schultz, October 10, 2017, March 3, 2018, <https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/>
- [2] Protection Against Cyberattacks and Automated Fraud, Shape Security, March 3, 2018, <https://shapesecurity.com/>
- [3] Industry Defense Against Cyberattacks and Automated Fraud, Shape Security, March 3, 2018, <https://shapesecurity.com/industries/>
- [4] What are extensions?, Google Chrome, March 3, 2018, <https://developer.chrome.com/extensions>
- [5] Yes, React is taking over front-end development. The question is why., Samer Buna, March 3, 2018, <https://medium.freecodecamp.org/yes-react-is-taking-over-front-end-development-the-question-is-why-40837af8ab76>
- [6] Jira (software), Wikipedia, March 3, 2018, [https://en.wikipedia.org/wiki/Jira\\_\(software\)](https://en.wikipedia.org/wiki/Jira_(software))
- [7] axios/axios: Promise based HTTP client for the browser and node.js, GitHub, March 3, 2018, <https://github.com/axios/axios>
- [8] Understanding module.exports and exports in Node.js, Cho S. Kim, July 17, 2017, March 3, 2018, <https://www.sitepoint.com/understanding-module-exports-exports-node-js/>
- [9] react-json-view, github.com/mac-s-g, March 3, 2018, <https://www.npmjs.com/package/react-json-view>
- [10] Understanding JavaScript Promises, spring.io, March 3, 2018, <https://spring.io/understanding/javascript-promises>
- [11] react-json-view, Axe Soota, March 3, 2018, <https://git.shpsec.com/sse/react-json-view>
- [12] Mocha - the fun, simple, flexible JavaScript test framework, github.com/mochajs, March 3, 2018, <https://mochajs.org/>
- [13] Istanbul, a JavaScript test coverage tool, github.com/istanbuljs, March 3, 2018, <https://istanbul.js.org/>
- [14] Sinon.JS - Standalone test spies, stubs and mocks for JavaScript. Works with any unit testing framework., github.com/sinonjs, March 3, 2018, <http://sinonjs.org/>
- [15] Chai, github.com/chaijs, March 3, 2018, <http://chaijs.com/>
- [16] airbnb/enzyme: JavaScript Testing utilities for React, AirBnB, March 3, 2018, <https://github.com/airbnb/enzyme>
- [17] Read Me - Redux, github.com/reactjs, March 3, 2018, <https://redux.js.org/>

## Appendices

### Appendix A: Usability Questionnaire

To test the usability of our applications we asked users to do the following tasks without our team's assistance. Please complete the tasks below:

Were you able to...

Start the application and all of its dependencies (Pegasus, Siftly). (Yes/No) Explain:

Enable capture mode, generate traffic, fetch the capture, disable capture mode then save. (Yes/No) Explain:

Replay a capture. (Yes/No) Explain:

Replay a capture with a new policy uploaded from the policy tab. (Yes/No) Explain:

Edit the validators from within a capture and from the settings page. (Yes/No) Explain:

See and comprehend validation results after replay. (Yes/No) Explain:

Login to JIRA. (Yes/No) Explain:

Create a ticket on JIRA and upload a capture. (Yes/No) Explain:

Download an attachment from JIRA. (Yes/No) Explain:

Other feedback:

## Appendix B: Generic SCR script analyzed result JSON for one policy

```
{
  "failed_captures_count": 1,
  "successful_capture_results": [],
  "failed_capture_results": [
    {
      "capture_name": "captures/capture_1_entry.json",
      "failed_entries_count": 1,
      "successful_entry_results": [],
      "failed_entry_results": [
        {
          "post_request_failed_fields_count": 0,
          "post_response_result": {
            "failed_results": [
              {
                "message": "error_message",
                "validator": {
                  "arg": "argument",
                  "op": "operation"
                },
                "namespace": [
                  "parent_field_name",
                  "field_name"
                ],
                "result": false
              }
            ],
            "successful_results": [
              {
                "message": "message_of_success",
                "validator": {
                  "op": "operation",
                  "arg": "argument"
                },
                "namespace": [
                  "field_name"
                ],
                "result": true
              }
            ],
            "failed_fields_count": 1,
            "key_of_interest": "_postResponse"
          },
          "post_response_failed_fields_count": 1,
          "entry_index": 0,
          "post_request_result": {
            "failed_results": [],
            "successful_results": [],
            "failed_fields_count": 0,
            "key_of_interest": "_postRequest"
          }
        }
      ]
    }
  ],
  "policy_name": "policy.txt"
}
```