Slide 1

Hybrid Security Analysis of Web JavaScript Code via Dynamic Partial Evaluation

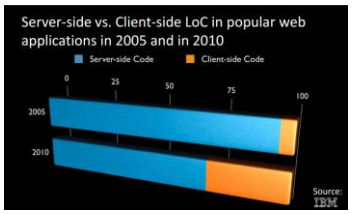
Omer Tripp, Pietro Ferrara, Marco Pistoia
IBM Watson Research Center
ISSTA 2014

Presented by
Peeratham (Karn) Techapalokul
11/17/2015

ISSTA : Software Testing and Analysis
TAJ: Effective Taint Analysis of Web Applications", PLDI 2009

Slide 2

Logic is Migrating from Server to Client



https://www.owasp.org/images/f/f4/ASDC12-Unraveling_some_of_the_Mysteries_around_DOMbased_XSS.pdf

This bar charts gives us the idea that business logic is migrating from server to client side.
Proportion of Line of codes in the client-side is growing when comparing web application in 2010 and 2005
This raises a growing security concerns over the client side.

Slide 3

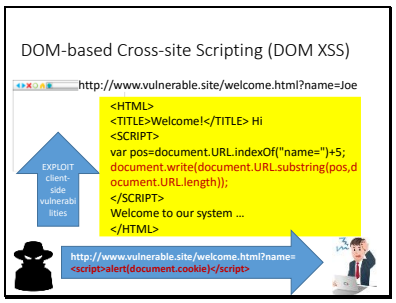
Client-side Web Application

- HTML (represented by XML DOM node tree)
- JavaScript embedded in HTML page
  - modify style and content
  - dynamically manipulate DOM elements
  - modify DOM in unintended ways by hackers?

The client side code consists of JavaScript embedded in HTML page
Javascript is used to dynamically mainipute HTML elements or we call it DOM (document object model)
We have learned that dynamic feature of JavaScript make it very challenging for static and dynamic analysis tool.

So making security analysis tool for client-side web application is challenging and that's the problem this paper tries to address

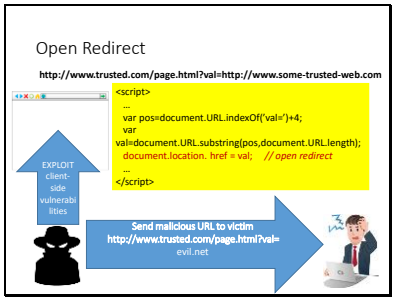
Slide 4



Specifically, the paper address two primary categories of client-side vulnerabilities.
The first one is DOM-based cross-site scripting.
The vulnerable client-side code is in yellow box.
The goal of the code is to write some text from the URL onto the screen of the user by inserting it into the HTML using JavaScript. However malicious javascript can be injected into pages.

Malicious javascript can take control over the web page.
The attacker can write javascript code to steal important information displayed on the page or secret information from the page's cookie and send them the server owned by hackers.

Slide 5

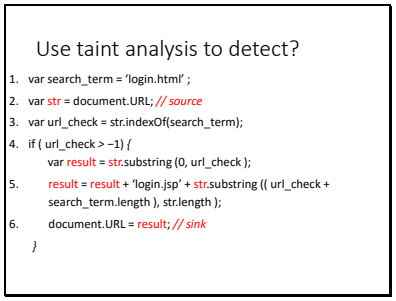


Another example of vulnerability we are considering is Open Redirect.
The code in the yellow box is unsafe redirection as it does not check for user-input URL.
This vulnerability is used in phishing attacks to get users to visit malicious sites without realizing it.

The user may assume that the link is safe since the URL starts with trusted url.
However, the user will then be redirected to the attacker's web site (evil.net)
which the attacker may have made to appear very similar trusted.com.

Where the malicious site trick the user to enter credential information.

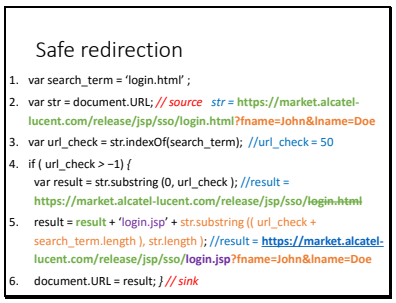
Slide 6



Let's take a look a more complex example and how we can use taint analysis to detect the vulnerability
This code performs redirection
the source is the URL of the web which may contain user input as part of the url
the sink is the statement performing redirection

Traditional taint analysis such as TaintDroid and TAJ would flag this code as vulnerable because there is a flow from source to sink. There is a possibility that user-input can influence redirection target.

Slide 7

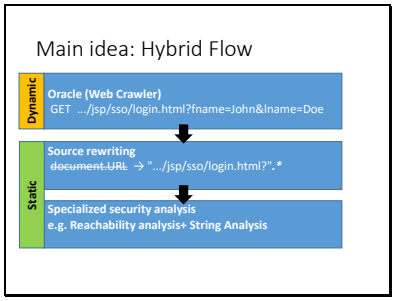


However when examine closely this is safe redirection.

The analysis approach we just did is what JSA the proposed tool in this paper can perform and the tool would also indicate no vulnerabilities found in this case too.

So what make this possible?

Slide 8



Basically we need to know the string value from DOM expression so DOM information is available when we do static analysis
So dynamic component of JSA is Web Crawler collecting relevant DOM information, and act as dynamic oracle

Also we need to take into account unknown user-input value.
and a way to analyze string as it is manipulated by the client-side code.
And that is the key component of JSA tool.

source rewriting module replaces DOM expressions with partially concretized values
it represents user-controlled portions abstractly as .* regular expression

JSA perform string analysis to track abstract string value as it is being manipulated
JSA report vulnerability if unsafe abstract value flows into sink.

Slide 9

Contribution

- Novel hybrid security analysis of client-side Web JavaScript code
  - Apply partial evaluation to JavaScript based on dynamic HTML environment to enable string analysis on abstract string values
- Reduction of 94% in false alarms while no single true positive is lost (compare with static taint analysis)

Slide 10

Overview

- Introduction
- Background
- JSA
  - JSA algorithm
  - String Analysis
- Implementation & Evaluation

Slide 11

JSA Algorithm

- Inputs:
  - call graph over JavaScript functions in the HTML page collected by Web crawler
  - dynamic oracle allows query of DOM expression values
- Outputs:
  - a set of security vulnerabilities detected over call graph

Slide 12

JSA Algorithm

- scan call graph for sources and sinks
  - sources : e.g. document.URL
  - sinks: HTML rendering methods e.g. document.getElementById(*id*).innerHTML = *"…"*;
- for each source and sink pair (st, st')
  - check reachability of data from source to sink
    - NO; no further analysis is required
    - if reachable: **perform security analysis**

Slide 13

Perform Security Analysis

- partially evaluate RHS DOM expression of sink by querying dynamic oracle
  - document.URL => http://www....
  - abstract string value (concrete+abstract segments)
- perform **string analysis**(forward analysis) until reach fixpoint
- query set of all abstract values that may flow to the sink
- report unsafe abstract values
- repeat reachability and security analysis for all source and sink pairs

conservatively treat user-input by abstract away user input segment from concrete input segment

Slide 14

String Analysis

- a refinement of taint analysis
- formally expressed as abstract interpretation
- **string abstraction** consists of a concrete **prefix** and a possibly unknown **suffix**
- **prefix =** string representation + boolean flag indicating if prefix has a suffix
- tracks abstract string values through local vars, procedure calls
- tracks integral values e.g. *indexOf* often used by *substring operation*

Slide 15

String Analysis (cont'd)

- partition (**P**) is a set of prefixes (**Prx**) with same lower-case representation
- **Idx** keep track of string index i.e. *indexOf*
- set of partitions (**PPrx**)
- define set of abstract semantics for string operations in Javascript

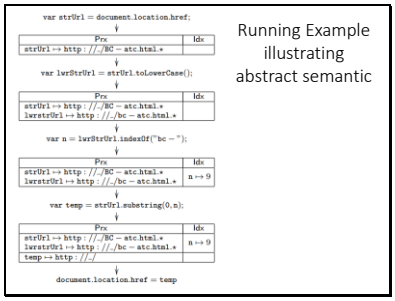
$$\mathbb{S}_{PPrx}[\![\mathtt{x} := \texttt{"str"}, (\mathbf{P}, i)]\!] = \{(\mathbf{P}', i)\} \cup \{(\{(\mathtt{x}, (\mathtt{str}, \mathtt{false}))\}, \emptyset)\}$$
$$\mathbb{S}_{PPrx}[\![\mathtt{x} := \texttt{"str"}.*, (\mathbf{P}, i)]\!] = \{(\mathbf{P}', i)\} \cup \{(\{(\mathtt{x}, (\mathtt{str}, \mathtt{true}))\}, \emptyset)\}$$

where $\mathbf{P}' = fgtStrVar(\mathtt{x}, \mathbf{P})$ in both definitions.

Example: abstract semantics of x := "str"

URL is case insensitive but string searching is case sensitive
group different prefixes having same lower-case representation in a partition so prefixes can share data-flow fact

Slide 16



Running Example illustrating abstract semantic

Slide 17

Overview

- Introduction
- Background
- JSA
  - JSA algorithm
  - String Analysis
- Implementation & Evaluation

Slide 18

Implementation

- Implemented JSA on top of WALA
- JSA currently integrated into IBM security AppScan Standard Edition
- AppScan has built-in AppScan crawler which acts as a dynamic oracle for JSA's partial evaluation

JSA currently integrated into IBM security AppScan Standard Edition
it's a commercial black-box security assessment product for testing both server and client side of web applications

Slide 19

Experimental Evaluation

- Compare JSA with two baselines:
  1. AppScan combined with a taint-analysis engine for static client-side security assessment
  2. AppScan without JSA (using dynamic client-side testing capabilities)
- 675 real-world websites (all Fortune 500 companies and top 100 websites IT and security vendors)
- findings were classified as being true/false positive by security experts

avoid almost all false alarms in static approach in 1
get true findings more than dynamic approach 2

Slide 20

JSA vs baseline Taint analysis

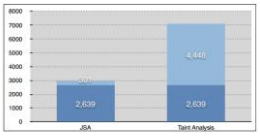


Figure 8: Breakdown of reported vulnerabilities by JSA and baseline taint analysis into true positives (bottom/dark) and false positives (top/pale)

gain in accuracy is significant
every true report by static Taint analysis appears in JSA report

JSA use average 3 seconds to analyze webpage
Taint analysis complete in less than 2 sec

Slide 21

JSA vs baseline Black-box testing

- randomly select 60 out of 675 websites

Configuration	Vulnerable websites	False positives
JSA enabled	33	4
JSA disabled	8	0

the granularity is at website level and not specific vulnerabilities
if the website has at least one false positive vulnerability, then it's counted as one

JSA outperform dynamic client-side testing in terms of coverage
JSA found 29 true positives vs 8 for pure dynamic client–side testing
and 3 sec vs 30-60 secs per webpage

Slide 22

Thank you!

Questions?