# Information Leakage Detection in JavaScript Browser Extensions

Jin Huang
Wright State University

Lu Liu
University of Colorado, Boulder

Jamie Weiss
Eastern Kentucky University

## Abstract

Browser extensions are being used widely and commonly by many. However, they are not always secure and have a potential to contain vulnerabilities through leakage of sensitive user information. Extension examples include profile information autofills or shopping assistants which accesses a user's sensitive information such as account passwords or banking information. This access then leads to the risk of developers storing the information somewhere or sending it to miscellaneous network servers.

The system developed from this project checks browser extensions written in JavaScript for any possible vulnerabilities related to information leakage. It is designed with a mindset of creating an elegant way of analyzing extension files by storing the read file information in optimal data structures and targeting the behaviors that many vulnerabilities show. Sensitive information includes but is not limited to: forms, cookies, passwords, URLs, account information, various history, orders, and bookmarks.

## Introduction

KEEP CITATIONS Citation of Einstein paper (?, ?). Citation of Freud book (Freud, 1900). Quick summary here

### Background

2010 third of chrome users with at least one browser extention

extensions with capability of taking user information and either saving locally or sending off of user's computer

estimation of 85% of organizations being affected by malicious browser extensions

### Objectives

creating system to detect information leakages in JS extensions

determin whether an extension's source code is deem vulnerable or not before being uploaded onto browser stores.

## Overall Framework

Rough overview of framework and system design

### Parsing Abstract Syntax Trees

usage of AST data structure, recursion

finding and using ESPRIMA

ESPRIMA's usage of the ESTree specification, storing output trees as .JSON files for analysis

### Interpreter

structure of interpreter with recursive switch statement and different cases

specifics on certain cases(?)

### Locating Sources Variables

collection of different trends/APIs/commands that obtain different information that can be deemed sensitive

encryption/decryption (?)

storing the sensitive information tags into a table

### Locating Sink Functions

collection of different APIs that were used as sink areas

elaboration on how the sink functions were identified

### Vulnerability Analysis

elaboration on environment created to keep track of sensitive variables

methods on how to connect source variables with the sink methods found, and finalizing the identified vulnerabilities

## Evaluation

some kind of evaluation overview

### Tests Results

creating test codes and example vulnerabilities

running interpreter system on created examples

**conclusion**

system is able to identify a code's vulnerable sources and sinks

future expansion on further cases, sources, and sinks for more thorough usability

adding in real world extension source codes with the testing data

needing to continue expansion due to data driven model

## References

Freud, S. (1900). *The interpretation of dreams*. Avon. (1965 ed.)