

# **Analyzing Information Flow in JavaScript-based Browser Extensions**

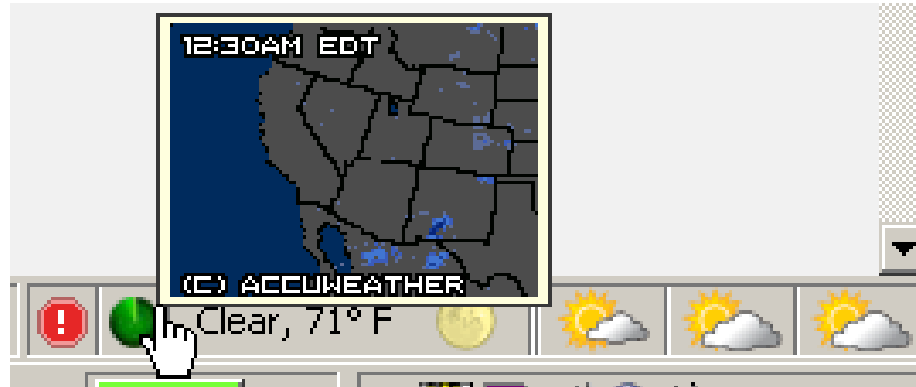
**Mohan Dhawan and Vinod Ganapathy**

Department of Computer Science

Rutgers University

# JavaScript-based Extensions (JSEs)

- Modern browsers support extensions
  - JavaScript-based Extensions

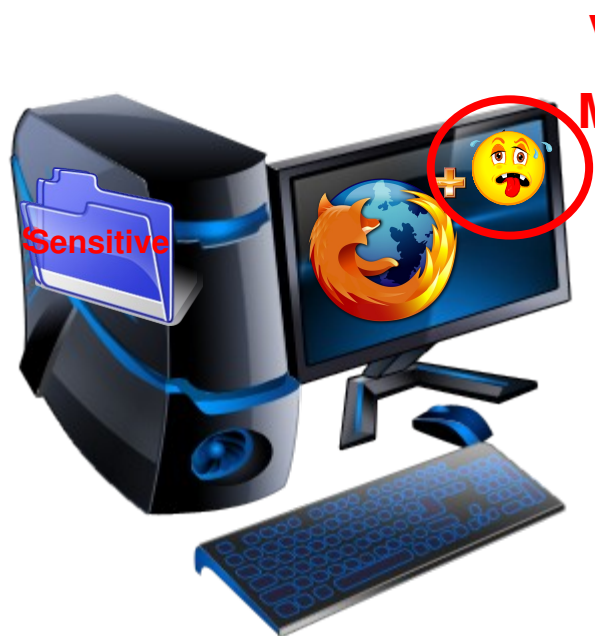


- Hugely popular
  - 1.5 bn JSEs downloaded, 150 - 190 mn used daily

[Mozilla Add-ons Statistics Dashboard]

# JSEs – A Security Risk

- Unrestricted access to system resources



- Hard to detect malicious code
- Inadequate sandboxing of JSE actions
- Lack of good development and debugging tools for JSE

# Outline

- Introduction
- **Motivating Example**
- **Solution**
- **Evaluation**
- **Conclusion**

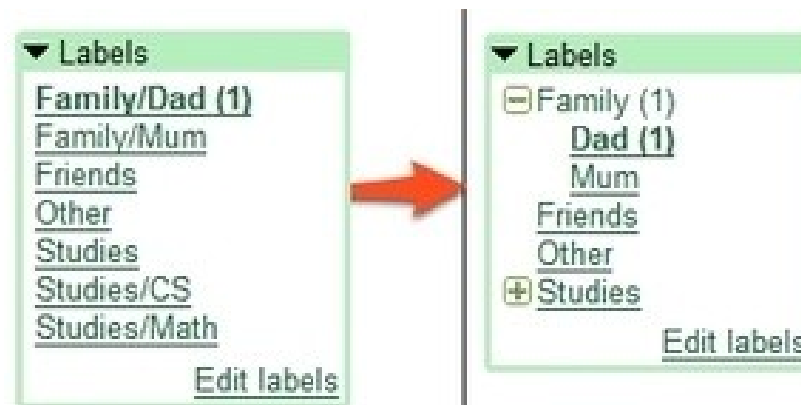


# GreaseMonkey

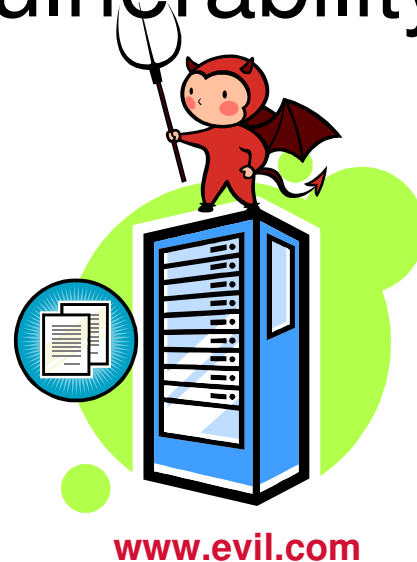
- Highly popular Firefox extension
  - nearly 3 million active daily users

[Mozilla Add-ons Statistics Dashboard]

- Exports a set of APIs for users to customize and program the way web pages look and function



# GreaseMonkey / Firefox Vulnerability



GreaseMonkey



Sensitive

Firefox with  
GreaseMonkey

Alice

**Exploited JSEs can  
lead to disclosure of  
confidential data**

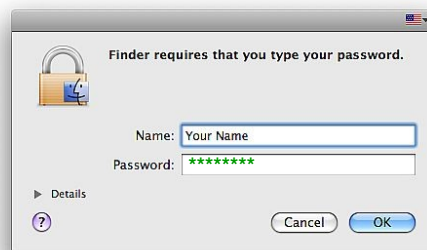
# Firefox Sniffer (FFsniff) – A Malicious JSE

**Sniffs all form fields**

**Emails them to the attacker**



**Firefox with  
FFsniff**



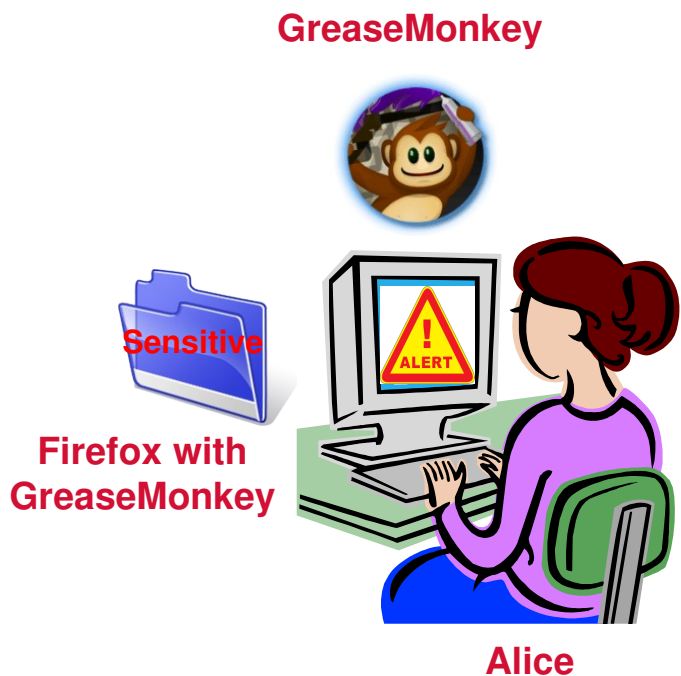
**Submit to the website**



# Outline

- Introduction
- Motivating Example
- **Solution**
- **Evaluation**
- **Conclusion**

# Solving the GreaseMonkey Problem



1. Mark data as sensitive
2. Take action when sensitive data is sent out



# Our Solution

- Security Architecture for Browser Extensions (Sabre)

**Enhance browser with JavaScript information flow analysis**

- Attach **security labels** with each JavaScript object
- Track the propagation of these labels
- Take action when a sensitive object is externalized

# Security Labels



Information flows from **sources** to **sinks**.

- Network
- File System



Sabre



- File System
- User Interface

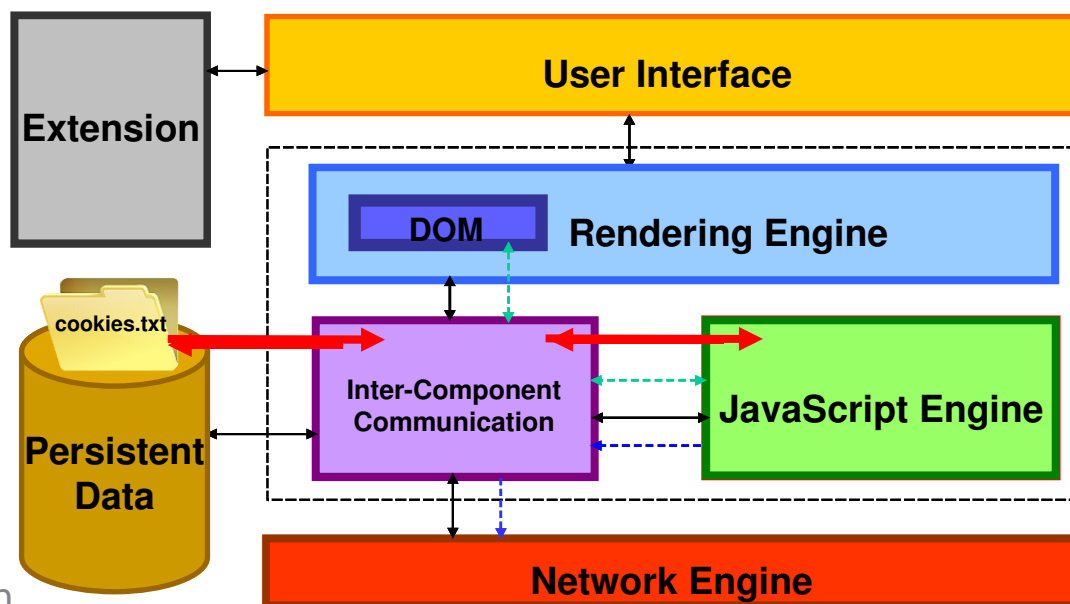
# Challenges in Real JSEs

1. Cross - Domain Flows
2. Implicit Flows
3. Benign Flows
4. Provenance

# Challenge 1 : Cross – Domain Flows

- JavaScript in a JSE can interact with other browser sub-systems

```
var cookieMgr = Components.classes["@mozilla.org/cookieManager;1"].  
    getService(Components.interfaces.nsICookieManager);
```



# Challenge 1 : Cross – Domain Flows (Object Access)

```
var cookieMgr = Components.classes["@mozilla.org/cookieManager;1"].  
    getService(Components.interfaces.nsICookieManager);
```

**Problem : Label propagation for objects and properties not managed by JavaScript**

**Solution : Assign sensitivity label of component to JavaScript objects**

- JavaScript can interact and store data in the DOM
  - Modify the DOM to store security labels also

# Challenge 2 : Implicit Flows

```
1: x = false;
2: y = false;
3: if (document.cookie == "abc") {
4:     x = true;
5: } else {
6:     y = true;
7: }
8: if (x == false) {
9:     // Line 6 was executed, and x is not tainted
10: }
11: if (y == false) {
12:     // Line 4 was executed, and y is not tainted
13: }
```

**Problem : How to handle direct control dependency?**

**Solution : Labeled Scope**

$$L(\text{lhs}) = L(\text{rhs}) \cup L(\text{scope})$$

## Challenge 2 : Implicit Flows (contd.)

```
1: x = false;
2: y = false;
3: if (document.cookie == "abc") {
4:   x = true;
5: } else {
6:   y = true;
7: }
8: if (x == false) {
9:   // Line 6 was executed, and x is not tainted
10: }
11: if (y == false) {
12:   // Line 4 was executed, and y is not tainted
13: }
```

**Problem : How to deal with all implicit flows?**

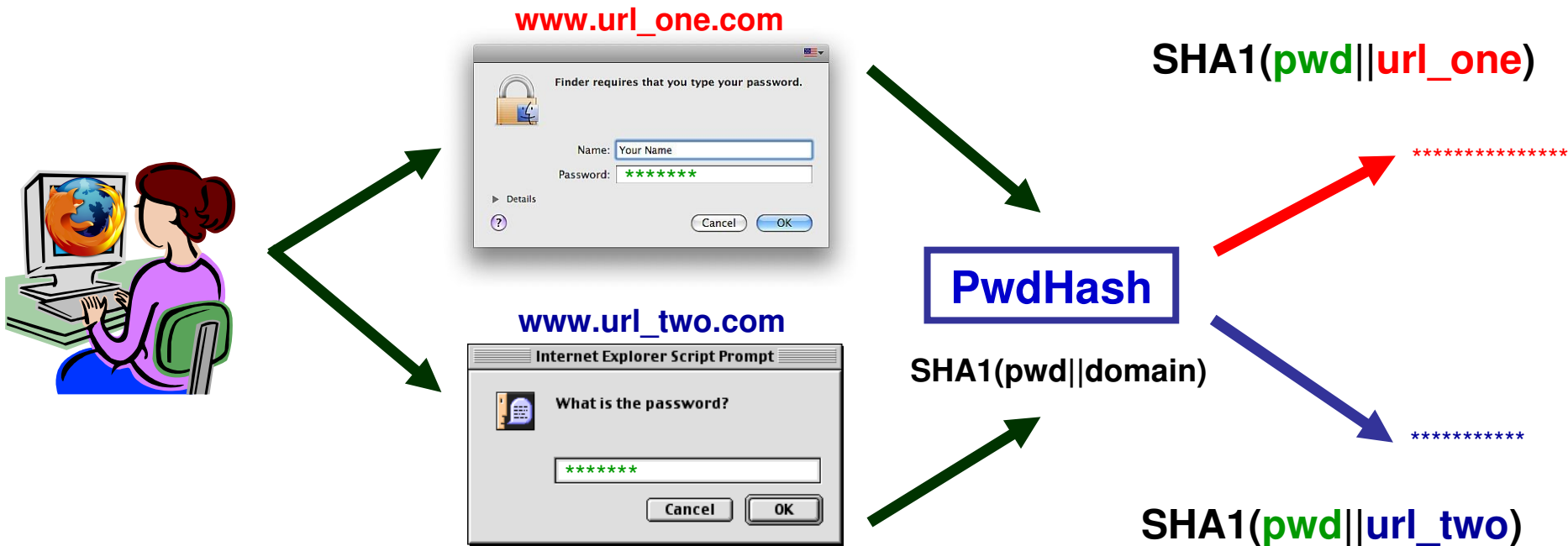
**Solution : Static analysis**

- Currently enhancing Sabre with support for static analysis

# Challenge 3 : Benign Flows

- Benign JSEs may contain flow violations
  - PwdHash

[Usenix Security '05]



## Challenge 3 : Benign Flows

- Disallowing them could render JSE dysfunctional

### **Problem : How to identify such flows?**

- Difficult to isolate malicious / benign behavior at runtime

### **Solution : Security analyst supplies a security policy to white-list trusted JSEs or declassify specific objects**

- De-classification of password field in PwdHash  
`<declassify, stanford-pwdhash.js, finish, 330, field.value>`

# Challenge 4 : Provenance

- Origin of the script
  - Needs to be determined only once at the time of dispatching the script for execution
- JSEs contain overlays
  - Describe patches for the UI and contain JavaScript code
  - Event - driven and not explicitly dispatched for execution

**Problem : Track provenance for “all” JavaScript including code in JSE overlay files**

**Solution : Per bytecode provenance tracking, or separately verify the overlay files**

# Outline

- Introduction
- Motivating Example
- Solution
- **Evaluation**
- **Conclusion**

# Evaluation - Goals

- Effectiveness
  - Classify behavior of benign JSEs
  - Determine information flow violations in malicious JSEs
- Performance
  - Impact on JavaScript performance
  - Compare overhead due to per-bytecode provenance check for overlay code

# Evaluation - Methodology

- Evaluated Sabre using a suite of 24 JSEs
  - Comprising over 120K lines of JavaScript code
- Enhance the browser with the JSE being tested and examine any flow violations
- Test Setup
  - Integrated Sabre with Firefox 2.0.0.9.
  - 2.33Ghz Intel Core2 Duo, 3GB RAM, Ubuntu 7.10

# Results - Categorizing Benign JSEs

Behavior key: (1) HTML forms; (2) HTTP channels; (3) File system; (4) Loading URLs; (5) JavaScript events.

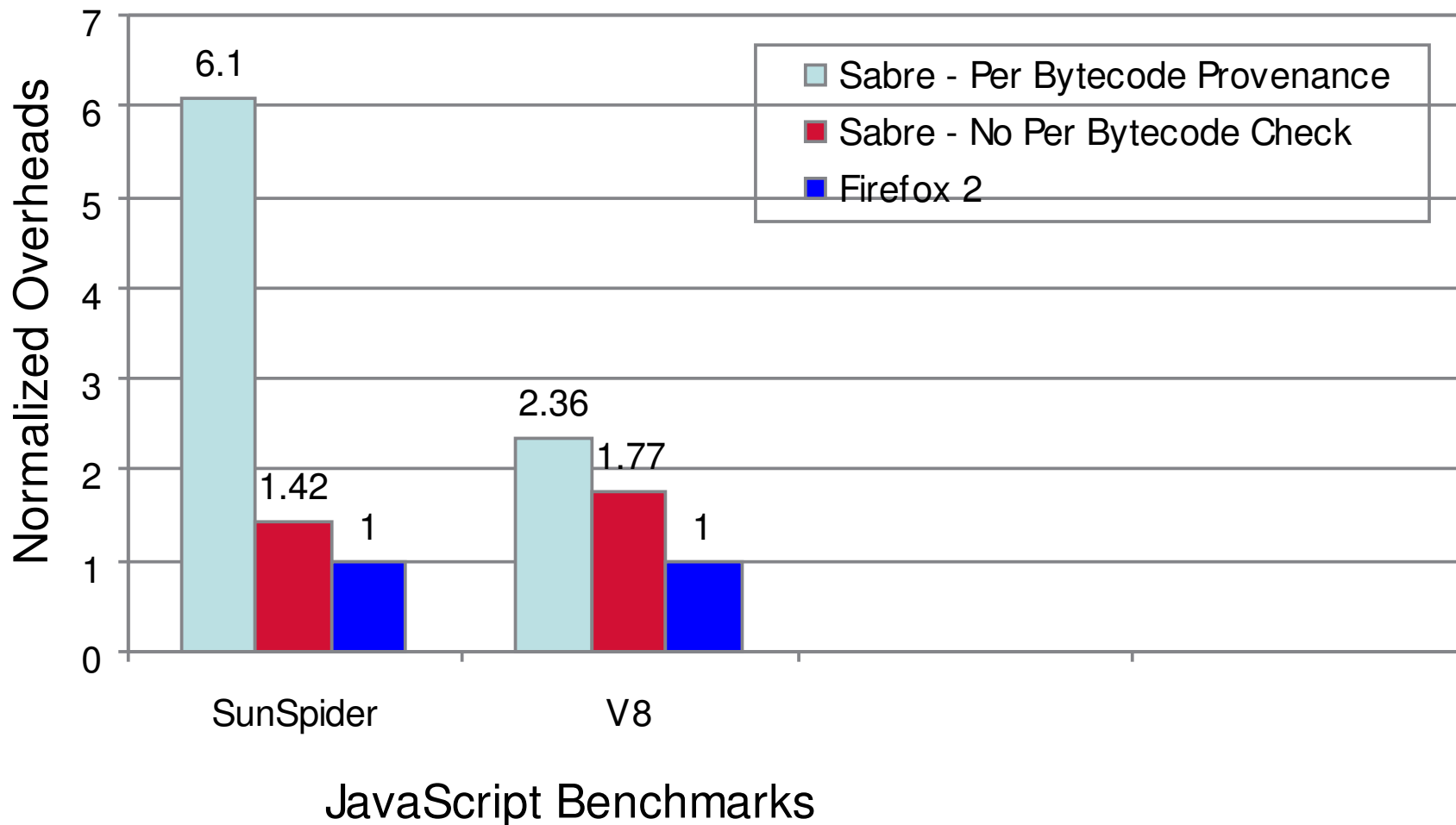
JSE	1	2	3	4	5
1. Adblock Plus		✓	✓		
2. All-in-One-Sidebar			✓		
3. CoolPreviews		✓	✓		
4. Download Statusbar			✓		
5. Fast Video Download				✓	
6. Forecastfox		✓	✓	✓	
7. Foxmarks Synchronizer		✓	✓		
8. Ghostery			✓		
9. GooglePreview		✓	✓		
10. Greasemonkey (0.8.1)		✓	✓		
11. NoScript		✓	✓		
12. PDF Download		✓	✓	✓	
13. Pwdhash	✓				
14. SpeedDial			✓	✓	
15. StumbleUpon		✓	✓	✓	
16. Stylish		✓	✓	✓	✓
17. Tab Mix Plus			✓	✓	
18. User Agent Switcher			✓		
19. Video DownloadHelper		✓	✓		
20. Web-of-Trust		✓	✓	✓	

**White-listing / De-classification of trusted JSEs is essential.**

# Results - Accuracy

- **Vulnerable & Malicious JSEs**
  - GreaseMonkey v0.3.3
  - Firebug v1.01
  - FFsniFF
  - BrowserSPY
- **Result**
  - Precisely identified all flow violations
  - No false positives during normal web browsing

# Results - Performance Overheads



# Outline

- Introduction
- Motivating Example
- Solution
- Evaluation
- **Conclusion**

# Conclusion

- Exploited JSEs can cause loss of sensitive information
- Policy-based access control is coarse grained and overly restrictive
- Sabre uses information flow tracking across browser sub-systems to prevent security violations in untrusted JSE code