SIFEX: Tool for Static Analysis of Browser Extensions for Security Vulnerabilities

Shikhar Agarwal, Indian Institute of Technology, Delhi, India

Sruthi Bandhakavi & Marianne Winslett, University of Illinois at Urbana-Champaign, USA

BACKGROUND

With the advent of "Web 2.0" and the "services computing" paradigm, web browsers have evolved from a platform to display static data to one facilitating client-side processing and more complicated applications. Browser extensions expand the functionality of browsers and often run with full browser privileges. The extensions could be written by in-experienced programmers, who often don't know how to secure their code or how to properly sanitize the inputs. This coupled with the non-standard semantics of JavaScript may give rise to potential vulnerabilities. These can be exploited by malicious users to steal confidential information like bank details, passwords, cookies, etc. They may also inject code that re-directs a user to a malicious website.

OBJECTIVES

The Research was conducted to achieve the following objectives:

1. To study and compare the three well known tools which are currently used to detect potential vulnerabilities in JavaScript code. These are SIF[1], Gatekeeper[2] and VEX[3].
2. To develop a framework that highlights potential security vulnerabilities in JavaScript based browser extensions in a flow-insensitive, field sensitive and static manner.
3. To use this framework on Firefox extensions to detect any potential vulnerability and compare the results with VEX.
4. To modify BANSHEE, an open source tool that simplifies the task of building constraint-based program analysis, to make it compatible with gcc 4.4.3 .

METHOD

The research work began with a careful analysis of previously known exploitable vulnerablities in browser extensions like Wikipedia Toolbar 0.5.9, Fizzle 0.5.1, Beatnik 1.2, etc. It was observed that most vulnerabilities were due to certain types of *explicit information flows* from sensitive sources to executable sinks. For e.g. consider the case of information flow from the JavaScript object "*window.content.document*" to the "*eval*" method. A webpage may have malicious JavaScript code in one of the elements of window.content.document object. If this object is passed to *eval*, code execution with *root privilege (called chrome privilege)* would take place and this may compromise the system. VEX[3] identified numerous such flow patterns which could be potentially vulnerable.

Motivated by this, work was then conducted towards building a tool that parses the extension's JavaScript code and explicitly lists the sensitive sources in them. We studied SIF, Gatekeeper and VEX, and chose Staged Information Flow, or SIF framework as the back-end and named the tool SIFEX.

SIFEX is developed in ml-lex/ ml-yacc. It parses the extension's code and syntactically finds the sensitive sources, which acts as the *confidential policy* for SIF. SIFEX also takes care of simple aliasing. The implementation also involves making minor changes to the extension's code to make it compatible with Jsure, which is used by SIF framework. Most of the modifications needed are automated through various scripts. We manually analyzed and modified the BANSHEE toolkit to make it compatible with gcc 4.4.3 .

We pass the list of sources into SIF, which gives us the *Residual Policy*, which is a list of variables and fields that is tainted by the flows that were of interest. We then check whether any executable sink is present in this Residual Policy by a grep-search. In the case of eval, we check whether any variable of this Residual Policy is its argument.

Once we have the sites of potentially vulnerable flow in the code, the code is manually checked to classify these as exploitable or not. Our analysis is sound.

RESULT

The present version of the tool checks for the following three flow patterns similar to VEX[3]:

1. RDF to innerHTML
2. Content Doc to innerHTML
3. Content Doc to eval

The analysis was carried out on Ubuntu 32 bit 10.04 LTS- the Lucid Lynx operating system on 1.2GHz 64 bit x86 processor. SIFEX has analyzed 2452 browser extensions and has parsed more than 3.8 million lines of JavaScript code. It has found potential vulnerabilities in 169 extensions. It took a total of 21144 seconds for the tool to generate the output, which means an average of less than 8.7 seconds per extension.

Table 1 gives the results in a tabular form. It can be concluded that SIFEX is far better than a grep-search for the sites of sensitive sources and sinks. Thus SIFEX can greatly reduce human efforts to detect exploitable vulnerabilities in extensions. It is also seen that SIFEX reports more potential vulnerabilities than VEX. Our analysis suggests that SIFEX has more false positives than VEX. This is due to flow-sensitive approach of VEX. But one should also note that VEX has been developed over a considerable period of time, while our analysis is simple and elegant.

SIFEX was able to detect many of the known exploitable vulnerabilities that fit into the three flow patterns that we have checked. We mention some of them as follows:

1) Wikipedia Toolbar version 0.5.9: SIFEX detected flow from Document Doc to innerHTML.
2) Fizzle versions 0.5.1 and 0.5.2: SIFEX detected a flow from RDF to innerHTML.
3) Beatnik version 1.2: SIFEX detected a flow from RDF to innerHTML.

FUTURE WORK

Currently, work is going on to analyze the results of SIFEX to classify the highlighted potential vulnerabilities as exploitable or non-exploitable. We also plan to make our analysis flow-sensitive.

Table 1

FLOW PATTERN	GREP ALERTS	VEX ALERTS	SIFEX ALERTS
Content doc to innerHTML	534	46	} 142
RDF to innerHTML	60	4	
Content doc to eval	430	13	38

CONCLUSION

We developed SIFEX, a tool for static analysis of browser extensions for security vulnerabilities. We ran our tool on thousands of JavaScript-based browser extensions. SIFEX reports various extensions with potential vulnerabilities, and is much powerful than manual inspection for these. Being static in nature, it avoids the overhead put by dynamic tools developed so far. This tool can be used by developers to build more secure extensions.

REFERENCES

[1] Ravi Chugh et al, "Staged Information Flow for JavaScript," *PLDI* '09
[2] Salvatore Guarnieri et al,"GATEKEEPER: Mostly Static Enforcement of Security and Reliability Policies for JavaScript Code," *Usenix Security Symposium*, USENIX, 2009
[3] Sruthi Bandhakavi et al,"VEX: Vetting Browser Extensions for Security Vulnerabilities," *Usenix Security Symposium*, USENIX, 2010