Final Report on SecCloud Results

SecCloud team

February 7, 2018

1 Initial Proposal

The SecCloud project will provide a comprehensive language-based approach to the definition, analysis, and implementation of secure applications developed using Javascript and similar languages. Our high level objectives is to enhance the security of devices (PCs, smartphones, ect.) on which Javascript applications can be downloaded, hence on client-side security in the context of the Cloud.

2 Results

2.1 Formal semantics of JavaScript for certified analyses

To establish the correctness of tools manipulating JavaScript programs, we first need a mechanized semantics of JavaScript, i.e., a description of the evaluation rules of the language expressed in a formal language such as that of the Coq proof assistant. While prior work has identified several interesting subsets of JavaScript, none have been related to the full-blown semantics of JavaScript through mechanized proofs. We have built of formal specification of JavaScript in Coq[1] in the form of an inductive definition (to prove properties), and executable definition (to run tests), and a correctness proof between these definitions. In addition, we are developing a tool to assist the development of the specification, called JSExplain.[2] In a nutshell, this tool is a debugger for the specification: for any program, one is able to trace not only the state of the program but also the state of an interpreter following closely the specification. JSExplain's development will continue thanks to funding for an engineer from CominLabs.

[1]`http://jscert.org/`

[2]`https://github.com/jscert/jsexplain`

2.2 Certified multi-semantics for non-interference analyses

Programming languages such as JavaScript provide many ways to accidentally leak private information. As they are too complex to be directly analysed, we have developed a systematic technique to simplify the proof of non-interference analyses. Given the semantics of a programming language, we mechanically build an annotated semantics that tracks information flows. This new semantics is tailored to capture non-interference and more easily prove analyses[3]. This work is the topic of Gurvan Cabon's PhD thesis.

2.3 Quantitative hybrid information flow monitor

Fingerprinting is a web tracking technique consisting in running a script which collects data identifying the user. We have developed a methodology to analyse how much identifiable information a tracker may learn about a user through an execution of a script. We have shown that this problem can be stated as an information-flow problem that answers the question: what is the probability that a server can identify a user after analysing the output of the script? If the probability is low, the user is unlikely to be tracked. If the probability is high, this is a tracking script trying to identify the user. We have also developed a quantitative information flow monitor computing how much the tracker learns when running a script in the user's browser.[4] The monitor uses a combination of dynamic and static analysis and over-approximates on-the-fly the amount of information that is learnt by running the script. If the amount of information is below a threshold, it is safe to send the output to the server. Otherwise, counter-measures need to be taken, such as shutting down the connection or providing forged but credible output. For this purpose, we have developed theoretical foundations for browser randomisation.[5] More recently, we have proposed a new version of a quantitative information flow monitor that is more precise in computing the knowledge of the tracker.[6] This new version expresses the monitoring of attacker knowledge in a general framework of semantics-based program analysis, and shows how a knowledge monitor can be combined with existing

[3]Gurvan Cabon, Alan Schmitt. Annotated multisemantics to prove Non-Interference analyses

[4]Frédéric Besson, Nataliia Bielova, Thomas Jensen. Hybrid Information Flow Monitoring Against Web Tracking

[5]Frédéric Besson, Nataliia Bielova, Thomas Jensen. Browser Randomisation against Fingerprinting: A Quantitative Information Flow Approach

[6]Frédéric Besson, Nataliia Bielova, Thomas Jensen. Hybrid Monitoring of Attacker Knowledge

monitoring techniques for information flow control, such as the 'no-sensitive-upgrade' principle.

2.4 Client-side security of web browsers

During the PhD thesis of Deepak Subramanian, we focused on the client side security of the web browsers, and limited ourselves to the context of Javascript. We analyzed potential risks of WebRTC, a HTML5 communication technology, in joint collaboration with KU Leuven and published the results in the 31st Annual ACM Symposium on Applied Computing 2016.[7] We also provided a mechanism where user's sensitive information is protected from disclosure (confidentiality) as well as unauthorized modifications (integrity) despite the vulnerability being exploited. For that purpose, we affirm that the vulnerabilities based on malicious script are characterized by illegal information flows. Hence, we proposed an approach based on Information Flow Control (IFC). Indeed, IFC-based approaches are more encompassing in their scope to solve problems and also provide more streamlined solutions to handling the information security in its entirety. Our approach is based on a practical IFC model, called Address Split Design (ASD), that consists in splitting any variable that contains sensitive data and maintaining the symbol table to protect accesses to the secret part of these variables. We have implemented our model on the chromium V8 engine, a full-fledged JavaScript engine. Following the implementation, performance and conformance testing have been done on our implementation. The measured performance drop is significantly smaller than other comparative approaches. We further showed that implementation of our approach does not affect the general working of existing websites by performing such a test over the top websites of the internet. Further, we have also been able to verify that our model can be used to protect variables in several scenarios that would have otherwise caused disclosure of secret information. This approach was presented in the 9th International Conference on Security of Information and Networks 2016[8] and in the 9th International Symposium on Foundations and Practice of Security 2016.[9]

[7]Willem De Groef, Deepak Subramanian, Martin Johns, Frank Piessens, Lieven Desmet. Ensuring Endpoint Authenticity in WebRTC Peer-to-peer Communication

[8]Deepak Subramanian, Guillaume Hiet, Christophe Bidan. Preventive information flow control through a mechanism of split addresses

[9]Deepak Subramanian, Guillaume Hiet, Christophe Bidan. A Self-correcting Information Flow Control Model for the Web-Browser

2.5 Dynamic Security Analyses for JavaScript

We have studied how to modularly extend JavaScript interpreters with dynamic security analyses in particular information flow analyses. This has led us to study ways to improve on the standard JavaScript module pattern. This pattern is commonly used to encapsulate definitions by using closures. However, closures prevent module definitions from being extended at runtime. We have proposed a simple pattern that not only opens the module, but allows one to extend the module definitions in layers.[10] The pattern leverages the with construct and the prototype delegation mechanism of JavaScript to mimick a form of dynamic binding, while minimizing the changes made to the module code. Florent Marchand de Kerchove's PhD thesis[11] details the proposal further and shows its application to the modular extension of Narcissus, a full-blown JavaScript interpreter, with several dynamic analyses, including the information flow of Austin and Flanagan based on multiple facets. A comparison with a previous ad hoc implementation of the analysis illustrates the benefits of the proposal.

[10] Florent Marchand de Kerchove, Jacques Noyé, Mario Südholt. Extensible Modules for JavaScript

[11] Florent Marchand De Kerchove. Extending interpreters by diverting, or how to extend interpreters without modifying their source code