PhD research proposal

Enrico Steffinlongo

May 28, 2014

Background

In these years there has been an increment in the usage of applications that have to interact both with user sensitive data and with the untrusted outer world. This increment is led by browser applications (HTML5 apps, browser extensions,...) and by smartphone apps, where there is little control on the quality and the safety of software. To mitigate these risks, typical proposals are built on a privilege separated architecture that divides the system in different components, giving to each component only limited permissions; to preserve interaction between isolated components, the latter can communicate using a tight Message Passing Interface. Moreover, to avoid privilege escalation attacks where a component that is compromised by an attacker can obtain permissions that it did not originally have, permissions are usually given statically, before the execution of the application. According to the permission system, there is a strict punctual mapping between components and permissions in order to reduce the possible exploits of the attacker.

The message passing interface between components is often regulated by a centralized monitor and it can deliver different kinds of messages: from simple strings to complex object, with pointers and methods. The message passing interfaces constitutes a relevant attack surface against privilege-separated applications, because it may lead to privilege escalation attacks, when a compromised, or malicious component that does not have a permission can send messages asking other components to trigger security-sensitive operations.

An important example of real systems with the feature above is Android. As discussed in [5, 8, 9, 7, 10] this architecture suffers various attacks like privilege escalation, since applications written by programmers that are not security experts can expose a large attack surface, being over-privileged or lacking of proper checks on the incoming messages.

Another case of privilege separated architecture which uses a simple message passing interface is the Google Chrome Extension framework. In this architecture, that extends the functionality of the browser, as shown in [3, 4], the choice is to separate every app in two sets of components: Backgrounds and Content Scripts. The former is the part of the application that interacts with sensible data, such as password and cookies, and with the browser core; as such, it is isolated from the page on which the extension is running, in order to prevent a malicious script in the page from being executed with high privileges. Content Scripts, instead, have no permission except the one used to send messages to the Background, and they have access to the page on which they are injected. To avoid direct delegation of privileges from the background page to

some content script, valid messages include only strings: objects, functions and pointers cannot be exchanged. This choice restricts the attack surface, since an attacker cannot send functions to be executed in the privileged background page. Unfortunately it also reduces the expressiveness of the language since any object is marshaled using a JSON serializer to a string that contains only values or pair field-value. Such serialization also fail in presence of recursive object (e.g., DOM elements) and breaks the typical prototype-based inheritance of Javascript. This reduce severely the potentiality of the framework because to deal to this limitation often developers has to do complex workaround just to perform simple tasks.

We argue that current privilege-separated architectures suffer limitations that affect both functional and non-functional requirements together with more complexity on the component-permission mapping. For example, even if sometimes permission can be required and revocated dynamically (e.g., Chrome Extensions [1]), often the programmer gives to each component the maximum of the permission required throughout its execution, even if a certain privilege is only rarely used.

Another crucial aspect in the design of an architecture for development of security critical software is the choice of the languages adopted. Scripting languages tend to be more easy to deal with and more expressive for development simple application they suffer of their weak typing discipline because it is hard to statically check them and is very hard to grant security properties leaving to the architecture the role to mitigate such missing. Strongly typed languages (like Java, C#, Haskell) are less liberal and through typing gives warranties on their behavior leaving the architecture to be more relaxed, but they are more complex and verbose, so not much suitable to small applications; so developers tend to avoid them.

Proposal

In this scenario interesting researches could examine deeper the state of the art and propose frameworks to increase security for users and robustness of applications, and tools for programmers.

In particular is interesting to explore deeply the trade-off between safety, easiness and expressivity of the chosen language. As written before the language is a crucial choice for the development and for getting security properties. In this field various solution can be adopted to enhance safety without limiting too much developers like various kind of typing such soft-typing, inference or automatic annotation to make the application more robust at run-time.

Another interesting improvement can be the enhancement of expressivity and safety of the architectures proposing more sophisticated message passing interfaces, allowing for example for example exchange of pointers and functions, better dynamic acquisition of privileges, checked delegation, strong sandboxing of some component [2], centralization or distribution of the monitor of the messages and a permission policy more fine-grained.

Such analysis will be done using both static and dynamic techniques taking in particular attention formal correctness using typical approach like typing [11], Flow logic [15, 12, 14], knowledge coming from the abstract interpretation field [6, 13] and model checking.

The research will start from actual solutions, their analysis and refinement through the various approaches listed before making tools prototypes able to show the effectiveness of the solution.

References

[1] Chrome extension overview
https://developer.chrome.com/extensions/overview, May 2014.

[2] Devdatta Akhawe, Prateek Saxena, and Dawn Song. Privilege separation in html5 applications. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 23–23, Berkeley, CA, USA, 2012. USENIX Association.

[3] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting browsers from extension vulnerabilities. Technical Report UCB/EECS-2009-185, EECS Department, University of California, Berkeley, Dec 2009.

[4] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. An evaluation of the google chrome extension security architecture. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 7–7, Berkeley, CA, USA, 2012. USENIX Association.

[5] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 239–252, New York, NY, USA, 2011. ACM.

[6] Giulia Costantini, Pietro Ferrara, and Agostino Cortesi. Static analysis of string values. In *Proceedings of the 13th International Conference on Formal Methods and Software Engineering*, ICFEM'11, pages 505–521, Berlin, Heidelberg, 2011. Springer-Verlag.

[7] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. Privilege escalation attacks on android. In *Proceedings of the 13th International Conference on Information Security*, ISC'10, pages 346–360, Berlin, Heidelberg, 2011. Springer-Verlag.

[8] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[9] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[10] Adrienne Porter Felt, Steven Hanna, Erika Chin, Helen J. Wang, and Er Moshchuk. Permission re-delegation: Attacks and defenses. In *In 20th Usenix Security Symposium*, 2011.

[11] Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. Typing local control and state using flow analysis. In *Proceedings of the 20th European Conference on Programming Languages and Systems: Part of the Joint European Conferences on Theory and Practice of Software*, ESOP'11/ETAPS'11, pages 256–275, Berlin, Heidelberg, 2011. Springer-Verlag.

[12] Rene Rydhof Hansen. Flow logic for carmel. Technical report, 2002.

[13] David Van Horn and Matthew Might. An analytic framework for javascript. *CoRR*, abs/1109.4467, 2011.

[14] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[15] HanneRiis Nielson and Flemming Nielson. Flow logic: A multi-paradigmatic approach to static analysis. In TorbenÆ. Mogensen, DavidA. Schmidt, and I.Hal Sudborough, editors, *The Essence of Computation*, volume 2566 of *Lecture Notes in Computer Science*, pages 223–244. Springer Berlin Heidelberg, 2002.