Poster: Using Object Capabilities and Effects to Build an Authority-Safe Module System*

Darya Melicher
Carnegie Mellon University

Yangqingwei Shi
Carnegie Mellon University

Valerie Zhao
Wellesley College

Alex Potanin
Victoria University of Wellington

Jonathan Aldrich
Carnegie Mellon University

CCS CONCEPTS

• **Security and privacy**; • **Software and its engineering → Language features**; **Modules / packages**; *Object oriented languages*;

KEYWORDS

Language-based security, capabilities, authority, modules, effects

The principle of least authority states that each component of a software system must have only the authority necessary for its execution and nothing else. This principle is a cornerstone of the security of software applications, but it is difficult to enforce in practice. Current programming languages, as well as non-linguistic approaches, do not provide adequate control over the authority of untrusted modules [1, 5]. To fill this gap, we designed and implemented a capability-based module system that facilitates controlling the security capabilities of software modules [2]. Furthermore, we are currently working on augmenting our module system with an effect system to make our design authority-safe. Our approach simplifies the process of ensuring that a software system maintains the principle of least authority, and also allows for attenuation of module authority [3]. Our design is implemented as part of the Wyvern programming language [4].

Modules in Wyvern are first-class, allowing us to model the common case of dynamically loaded modules. We consider modules representing or using system resources, such as the foreign function interface, file system, and network, to be security-critical modules and designate them as resource modules. References to resource modules are capability-protected, i.e., to access a resource module, the accessing module must have the appropriate capabilities.

Wyvern's module system allows a software developer to verify modules' capabilities at compile time by looking only at modules' interfaces and not at their code, thus significantly simplifying the task of controlling the capabilities a module holds. In addition, from a theoretical viewpoint, our capability analysis uses a novel, non-transitive notion of capabilities.[1] For example, suppose a third-party extension module has access to a logging module, which in turn has access to the file system. In a capability analysis with a transitive notion of capabilities, the extension module is said to hold a capability for accessing the file system, which is too conservative and leads to an overapproximation of modules' capabilities. In contrast, in Wyvern, the extension module is said to hold a capability to access the file system only if that capability is explicitly passed to it, and otherwise, the extension module has only attenuated authority [3] over the file system via the logging module.

Taking our work a step further, we would like to provide software developers with insight into what exactly a module can do with an imported resource, i.e., the module's authority. For example, for a security expert analyzing the scenario above, it is important to know exactly what effects the logging module has on the file system and be able to verify that the logging module follows the principle of least authority. Towards this step, we are currently working on adding to Wyvern an effect system that can account for the effects a module has on each resource.

In Wyvern, effects are capability-based, meaning that, in their definitions, they refer to capabilities. For instance, the logging module in the scenario above may have the `fileIO.appendToFile` effect, where `fileIO` is the capability the logging module holds to access the file system and `appendToFile` is a developer-chosen name for the effect of appending to a file in the file system. Another distinct feature of our effect system design is effect abstraction, which:

- gives software developers flexibility to choose a level of effect abstraction based on the importance of various resources in the application (e.g., using the higher-level, logging effects vs. the lower-level, file-system effects);
- allows software developers to enforce what resource may be used to implement certain functionality (e.g., the logging module must be implemented using the file system resource and not any other resource).

On the theory side, we are working on using our effect system to formalize the notion of authority attenuation and prove Wyvern to be authority-safe.

*This poster abstract is partially based on the work presented at ECOOP 2017 [2].

[1] In the ECOOP 2017 paper [2], we called this "authority," but the term "capability" is more consistent with prior literature.

HoTSoS '18, April 10–11, 2018, Raleigh, NC, USA

ACM ISBN 978-1-4503-6455-3/18/04.
https://doi.org/10.1145/3190619.3191691

REFERENCES

[1] M. Maass. 2016. *A Theory and Tools for Applying Sandboxes Effectively.* Ph.D. Dissertation. Carnegie Mellon University.
[2] D. Melicher, Y. Shi, A. Potanin, and J. Aldrich. 2017. A Capability-Based Module System for Authority Control. In *European Conference on Object-Oriented Programming*.
[3] M. S. Miller. 2006. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control.* Ph.D. Dissertation. Johns Hopkins University.
[4] L. Nistor, D. Kurilova, S. Balzer, B. Chung, A. Potanin, and J. Aldrich. 2013. Wyvern: A Simple, Typed, and Pure Object-Oriented Language. In *Workshop on Mechanisms for Specialization, Generalization and Inheritance.*
[5] Z. C. Schreuders, T. Mcgill, and C. Payne. 2013. The State of the Art of Application Restrictions and Sandboxes: A Survey of Application-oriented Access Controls and Their Shortfalls. *Computers and Security* 32 (2013), 219–241.