A Taxonomy Built on Layers of Abstraction for Time and State Vulnerabilities

Horia V. Corcalciuc, School of Computer Science, University of Birmingham, Birmingham, UK

ABSTRACT

Software classifications have been created before with the purpose of keeping track of attack patterns as well as providing a history for the various vulnerable software packages. This article focuses on one single class of such attacks, conventionally known as "Time and State" attacks. The authors offer a more fine-grained analysis of the anatomy of such attacks. They reason about vulnerabilities by using "swimlane" diagrams which are loosely derived from UML diagrams, annotated with semantics of concurrent programming, such as the notions of traces and stability. The authors offer a taxonomy based on abstraction layers, implying thereby some form of tree hierarchy where vulnerabilities inherit properties from the upper abstract layers and share code-level flaws on the lower layers. That allows them to classify attacks by what they share in common, which is a different approach than other related classification attempts.

Keywords: *Abstraction, Attacks, Classification, Diagrams, Security, Signals, Software, Taxonomy, Time-of-Check-To-Time-of-Use (TOCTTOU), Vulnerabilities*

1. INTRODUCTION

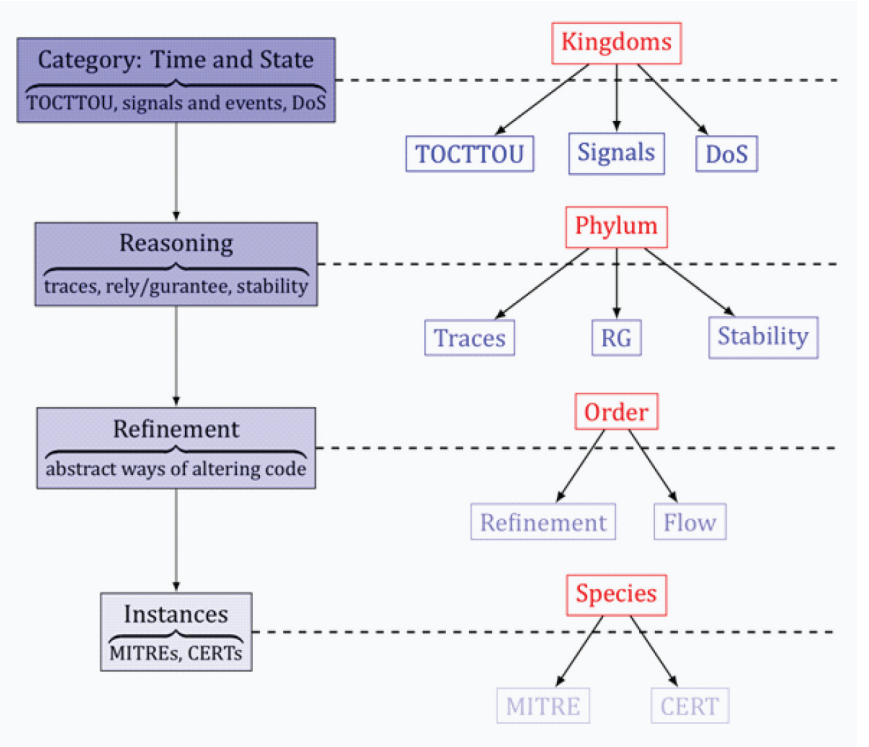
There have been several attempts of bringing some order to security classifications, variously called "*Top Ten Vulnerabilities*", "*Seven Deadly Sins*" or "*Pernicious Kingdoms*" (Tsipenyuk et al., 2005) The latter classification, by McGraw and collaborators, is the most scientific one. It borrows the idea of a taxonomy from biology. McGraw predicts that more sophisticated attacks will become increasingly dangerous, such as the class of "Time and State" attacks.

This paper, as an extension of a workshop paper (Corcalciuc, 2012), reasons about "*Time and State*" attacks and offers a method of building taxonomy trees based on layers of abstract concepts. We notice that attacks frequently exploit a theoretical concept rather than local defects in software packages. We leverage concepts from programming theory in order to make "*Time and State*" attacks more precise.

The terminology of "*Kingdoms*", "*Phylum*", "*Order*" and "*Species*" are only crudely related to our taxonomy and we adopt only the structure of the biological taxonomy. We use that terminology in order to provide a distinc-

DOI: 10.4018/jsse.2013040103

Figure 1. Our taxonomy is annotated using McGraw's terminology. Each level describes a level of abstraction and every vulnerability can be classified by following the tree structure of a given attack. The upper layers are populated with abstract concepts such as TOCTTOU, signals and even more broadly DoS and reach down to lower layers where attacks distinguish themselves by local defects in a software package.



tion between abstract security concepts and code- level safety issues on the lower layers of the tree.

Compared to biology, in terms of security, vulnerabilities with common traits on the upper layers will be grouped together. We limit the article to the Time-of-Check-To-Time-of-Use (TOCTTOU) and "*Signals and Events*" as illustrated in Figure 1. Additionally, we attempt to classify Denial-of-Service (DoS) and explain how DoS can be both a Kingdom or appear by consequence tied to other Kingdoms.

The "*Kingdom*" (Singer, 1950) rank is reserved for very high level classifications with a broad variety of descendants. The upper layers are reserved for abstract concepts which trickle down to the lower levels of the tree. In biology, the "*Phylum*" rank is a grouping of organisms based on a general abstraction of structure (Valentine, 2004). Thus, the "*Phylum*" is populated by the abstraction layer holding formal concepts such as traces, states and predicates.

The connection between our taxonomy and the rank ``Order'' is that biological ranks group elements together based on small but important differences. For example, Zoology makes a distinction between moths and butterflies - which is not a trivial distinction. In our taxonomy, ``Order'' represents general elements of flow control and refactoring (Fowler, 1999) the result of which may slightly change the code but sufficient enough in order to distinguish between a program and the re-factored equivalent.