Loren Taylor  
MOPS: an Infrastructure for Examining Security Properties of Software  

This paper describes software tool called MOPS that can be used for finding security bugs in other software based on pre-defined models. It is also used for verifying that certain known bugs do not exist in the software being analyzed. The designers of MOPS also intend and claim it to be scalable to much larger and more complex programs.

The approach MOPS uses is based on automata theory. MOPS uses Finite State Machines (FSMs) to model the security properties that will be checked in the software to be analyzed. It models the program execution with a Push Down Automata (PDA), and then uses model checking techniques to test if any state that violates a security property is reachable in the PDA.

The primary bugs that the paper focuses on identifying are those that violate ordering constraints. These are known as temporal safety properties and they simply describe the order in which security-relevant operations should occur. The paper then goes on to give some examples of simple ordering constraints and why they have their respective orderings. One of which is that the \texttt{chroot} system call should always be immediately followed by the \texttt{chdir(“/“)} call.

MOPS performs these analyses at compile time, however it does not check every possible execution path. The authors claim that the techniques they use from model checking and program analysis can be substituted instead of the infeasible brute force approach. This property, if it is true, allows MOPS to detect all bugs that are defined as security property FSMs. This is an excellent quality with a minor cost of producing some potential false positives.

MOPS relies on the assumption that security properties can be described by regular languages and therefore as FSMs. It is conceivable that if there are properties that can only be described by something less restrictive, then you would not be able to model it and MOPS would not be able to detect it.

I liked this paper because it flowed well, used good, relevant examples, and acknowledged its shortcomings throughout. Some of these shortcomings include not detecting control flow through function pointers and signals. MOPS also does not do dataflow analysis by default since this operation is expensive, but it can do explicit analysis through the use of \textit{pattern variables}.

The authors briefly mention that they have a novel technique that reduces the size of a program without compromising the model checking. I would have liked to have seen more discussion on this and at least a proof outline or sketch.

I don’t think they made their case that MOPS is scalable because they only cite MOPS performance in one specific instance in section 5.4. The authors conclude that MOPS will scale well because it performed well on sendmail 8.12.0. I would have liked to have seen a larger Performance section with results from more programs on different platforms.