Algorithmic Trace Effect Analysis
Context Based Security in Programming Languages

David Van Horn
dvanhorn@cs.uvm.edu
http://www.cs.uvm.edu/~dvanhorn/

Christian Skalka
skalka@cs.uvm.edu
http://www.cs.uvm.edu/~skalka/

Abstract

Trace effect analysis is a programming language analysis for ensuring correct and secure behavior of software with respect to so-called temporal properties of programs. Temporal properties express the well-ordering of program events such as when files are opened or when resources are acquired. By enforcing program event order specifications, we can ensure that files are opened before being read, or that privilege activation occurs before privileged resources are acquired. In general, the ability to enforce temporal properties in programs allows a wide variety of safety and security guarantees to be made about software.

Fundamental abstraction: Event traces

Many program correctness properties are expressible as properties of program event traces.

- Security handshake protocols, eg. SSL
- File open before read
- Allocate before use
- Access control: privilege activation before privileged action

Well-formedness of traces expressible and enforceable as program monitors or checks in program logic.

We take a language-based approach, integrated the necessary abstractions into a programming language so that a programmer can articulate temporal properties.

Logical typing rules

\[
\begin{array}{l}
\text{EVENT} \\
\Gamma, H \vdash e :: \{ \} \\
\text{WEAKEN} \\
\Gamma, H \vdash e :: T \\
\Gamma, H \vdash e :: \tau \\
H \subseteq H' \\
\end{array}
\]

Example: Detecting security violations in Secure Socket Layer (SSL)

For a program sending and receiving data over an SSL socket, we can use a web browser that supports HTTPS, the relevant events are opening and closing of sockets, and sending and receiving of data packets.

An example event trace produced by a program run could be:

\[
\begin{align*}
\text{ssl_open("snark.cs.jhu.edu", socket_1); ssl_handshake(socket_1); ssl_handshake_success(socket_1); ssl_packet(socket_1, socket_2); ssl_handshake(socket_1); ssl_close(socket_1, socket_2)}
\end{align*}
\]

Trace effects can be post-processed to analyze variations to the core language.

- **Simplification**: Traces may be simplified in a semantic-preserving way in order to improve model checking efficiency.
- **Stack-based analysis**: In a stack trace model, event ordering during function execution are forgotten when the function returns. Function activations annotated with events; function return erases event.
- **Exceptions**: "Pre-effect" constructs allow us to add exceptions to the language with a trivial extension to the algorithm.

Results

- Proof establishing correctness of logical system
- Proof establishing correctness of transformation algorithms
- Proof establishing soundness of algorithm
- Prototype implementation of analysis
- Prototype implementation of transformation algorithms
- Extension of analysis to Java

Publications