Model Checking in BSD Userland and Kernel

BSDCan 2026

Justin Handville

June 20, 2026

Outline

The Threat Landscape is Changing

- Significant investments have been made in Large Language Model (LLM) and agentic based penetration testing.
- LLMs can take existing attack tactics and strategies and apply them / chain them to create novel attacks.
- The number of LLM based vulnerability and error reports — both valid and drive-by vibes — is becoming overwhelming for many projects.

Responding to These Threats

- Triage of these issues is just treading water, unless we can get in front of it.
- We need to raise the bar for both safety and security in open source.
- The BSDs are an excellent place to do this.
- But, what can we do?

My Background

- I've been developing system software, firmware, and languages for 27 years.
- For the past ten years, I've been studying a subset of Programming Language Theory and Formal Methods to find a practical solutions.
- I had the opportunity to spend some time at Google performing triage for their fork of the Linux kernel.
- I got to see first hand how much work went into this, before Linux became their own CNA and after.
- It has only gotten worse...

Formal Methods

- A set of rigorous mathematical techniques for the specification, development, and verification of software and hardware.
- Shifts the goalposts from testing / test automation to formal proof.
- Focuses on defining properties that functions must have for every input, output, and side-effect.
- Approaches range from Calculus of Constructions to Model Checking.

Satisfiability (SAT)

- Boolean satisfiability problem - Can a solution be found for a large Boolean formula?
- The first generalized solver algorithm was defined in 1962 (DPLL).
- The problem was proven to be NP-Complete (Cook and Levin).
- Additional optimizations such as CDCL significantly improved the performance of solvers.
- Boolean formulas are stated in CNF (Conjunctive Normal Form) and reduced via backtracking (DPLL) or conflict-driven learning (CDCL).

Satisfiability Modulo Theories (SMT)

- Extends SAT to solve integer, real number, string, lists, etc.
- At the simplest level, ALU operations can be defined in terms of Boolean circuits and solved like any other SAT problem.
- Additional optimizations can reduce search depth.

Model Checking

- Checks whether a finite-state model of a given system matches a specification.
- Software can be transformed into a finite-state model via bounded model checking: loops are unrolled and recursion is flattened. Unwinding assertions are added to verify the depth of unwinding.
- The specification properties for this finite-state model are negated and passed to a SAT solver.
- If the SAT solver finds a solution, this is a counter-example demonstrating a case where the specification fails.

CBMC

- The C Bounded Model Checker (CBMC; 2004; Kroening and team) extends model checking to C.
- This is the tool demonstrated in this talk.
- C source code is converted via goto-cc to an intermediate form that can be verified by CBMC.
- CBMC includes a list of instrumentation options that can be invoked, either one at a time or all together, to find specific counter-examples.
- Instrumentation looks for violations in memory safety, memory leaks, concurrency errors, integer safety, or user assertions (e.g. function contracts).

Example 1: mod_fido

- fido (filtered "do") is a doas clone built using model checking.
- fido itself is Example 2.
- mod_fido provides authentication persistence on FreeBSD, which the portable version of doas lacks, by adding similar tty cached credential support to the FreeBSD kernel.
- This is done by hooking the ioctl system call, filtering for specific ioctls, and tracking tty state in a tree.
- Don't attack the messenger: I know this is poor practice; I wanted to keep the example simple.

Practical Considerations

- State space explosion: model checking scenarios must be kept small and loop unwinding must be kept to the minimum necessary to verify loop invariants and boundary conditions.
- Scope creep: use CBMC to look for safety and correctness given function contracts and invariants. Use unit testing to verify features.
- Verify function by function, making use of compositional verification.

Compositional Verification

- Write function contracts using Hoare logic (preconditions, invariants, and postconditions).
- If we constrain functions to contracts, we can "shadow" them.
- A "shadow" is bound by the same contracts but is simplified to exhibit non-deterministic behavior, decoupling its domain from its range.
- Soundness check: We run the same model checking scenario against a function and also its "shadow" to verify that both follow the same contracts.
- Once verified, the shadow can be substituted for the original function, simplifying the verification of functions that call this function.

mod_fido code dive

- `https://github.com/nanolith/mod_fido`
- “Beware of bugs in the above code; I have only proved it correct, not tried it. ” — Donald Knuth

Example 2: fido

- fido is a setuid root binary that parses an optional config file to filter policy decisions.
- It spawns a separate fido instance (fork + exec) that opens the config file descriptor, and immediately sandboxes itself via pledge / unveil (OpenBSD) or cap_enter (FreeBSD) to parse the config file and make a policy decision. The result of this decision is returned.
- If the policy decision is approved, fido runs the command under a config defined user and group.

fido code dive

- `https://github.com/nanolith/fido`

Let's Use This!

- I've demonstrated that this can be used today.
- Please reach out if you'd like me to help you integrate it with anything in the userland or kernel.
- I am happy to share what I have learned and assist.

Project Ideas

- Let's threat model this and prioritize by severity of attacks.
- List any areas with a potential for remote exploits. Let's verify that common attack vectors have been mitigated.
- Let's verify pledge / unveil (OpenBSD), capsicum / MAC (FreeBSD), and/or kauth / common kauth extension modules (NetBSD).
- Let's verify that jails and virtual machines are free from common attack vectors that can lead to container escape.
- What about the kernel and userland layer? Can we reduce the attack surface for local privilege escalation?
- Where else?

Questions and Answers

- You have questions. I have answers. Let's see if they match.

Thank you!

- `https://github.com/nanolith`
- justin@danger.farm