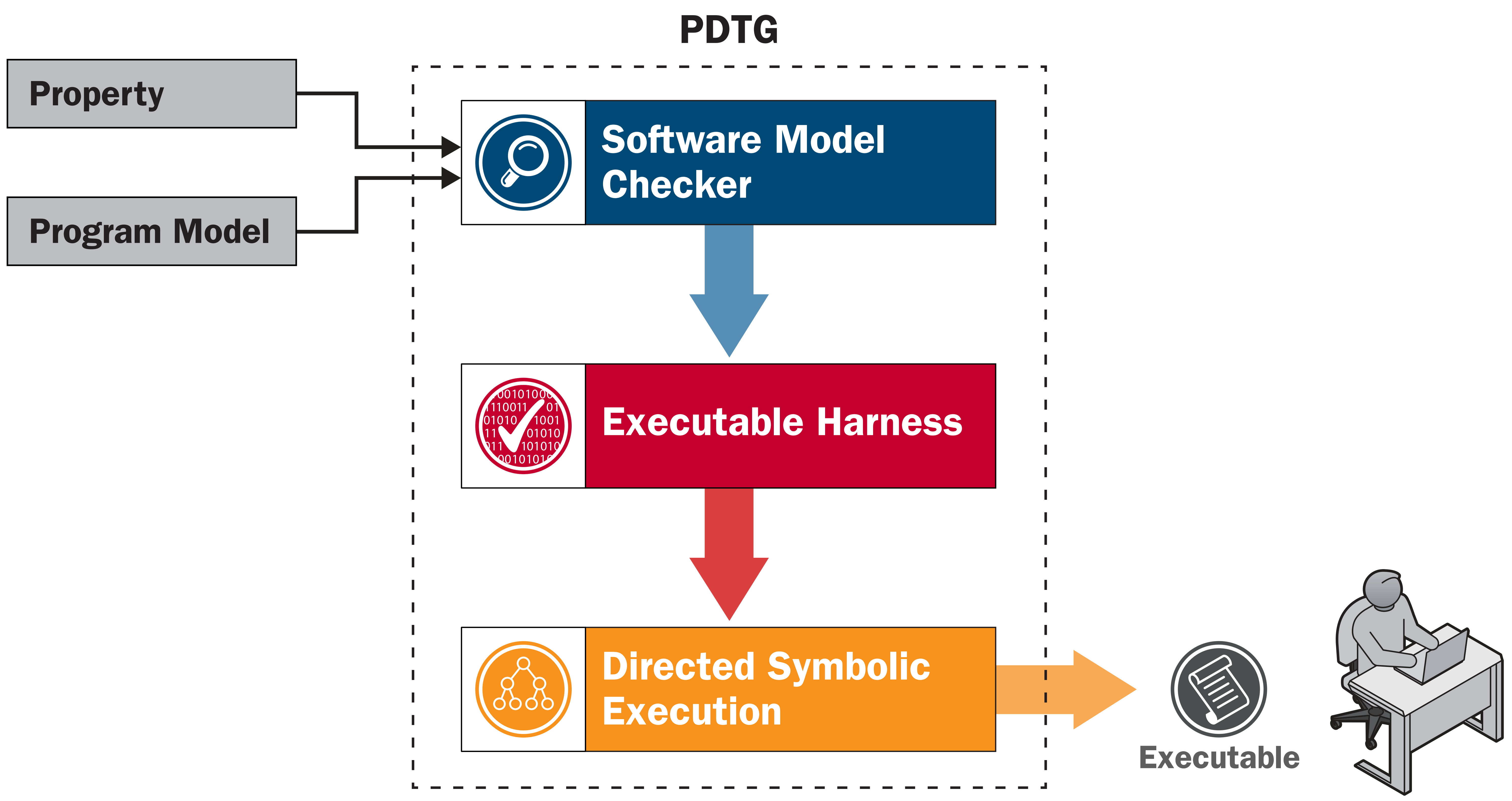
Property Directed Test-case Generation

Manually finding inputs to trigger a behavior of interest in a program is complex and time consuming. In this project, we repurpose existing formal methods techniques to help automate this problem. We use counter examples produced by SEI's Seahorn model checker to create executable harnesses that demonstrate how the behavior of interest can be reached.

Model checker (Seahorn) produces counter example (trace) showing how to reach property or behavior of interest

Executable harness implements external methods needed to execute path in trace

KLEE is a symbolic executor that fuses together trace with values from executable harness to produce valid executable

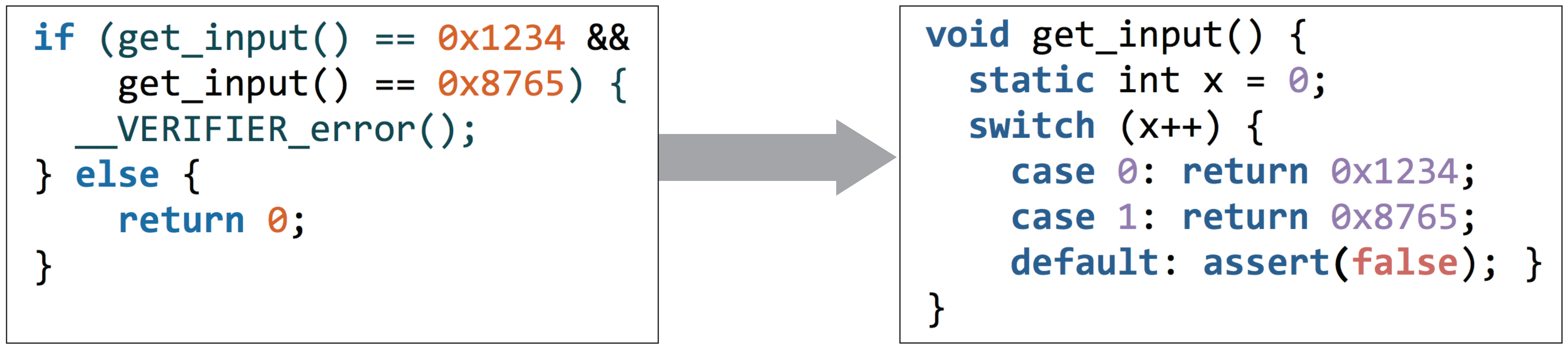


Verifying Linux Device Drivers

A common problem when model checking software is understanding the results that the model checker yields. For example, a small discrepancy in modeling can result in a complicated counter-example that is difficult to understand. We applied PDTG to model checking instances of Linux Device Drivers where the model check failed, and automatically produced an executable harness that showed the problematic execution. The final harness can be executed in a debugger and reviewed step by step, which makes correcting the problem much easier.

Reverse-engineering Malware

We also used PDTG to assist in reverse-engineering malware. We start with a sequence of API calls that may indicate malicious or interesting behavior. For example, enumerating processes on Windows requires calls to `CreateToolhelp32-Snapshot`, Process32First, and Process32-Next in sequence. PDTG can construct a harness that forces the program to execute these calls, and thus display the malicious behavior for an analyst. We tested this technique on the Gh0st RAT variant.



```
if (get_input() == 0x1234 &&
    get_input() == 0x8765) {
  __VERIFIER_error();
} else {
    return 0;
}
```

```
void get_input() {
  static int x = 0;
  switch (x++) {
    case 0: return 0x1234;
    case 1: return 0x8765;
    default: assert(false); }
}
```