Phase2: State Inspection
Control Flow Integrity

Alexander Küchler, Martin Morel

Injected Code

StackAnalysis.c

- void registerFunction(char*)
- void deregisterFunction(char*)
- void response()
- void verifyStack()

Transformation Pass

- FunctionPass
- Beginning of function: call registerFunction(char*)
- End of function: call deregisterFunction(char*)
- Beginning of sensitive function: call verifyStack()
- Iterate over statements, on every function call, add new edge to call graph (own implementation)

Stack trace verification

- Problem: call graph can be arbitrary complex

- Naive idea: Hashing and remove cycles
- Problem: Easy to attack, exploding states, complex graph analysis

- $\Rightarrow$ No hashing but check every state
- Build complete stack and check against known good call graph edge by edge
- Performance optimizations: Store as few edges as possible, efficient algorithms,…

Response model

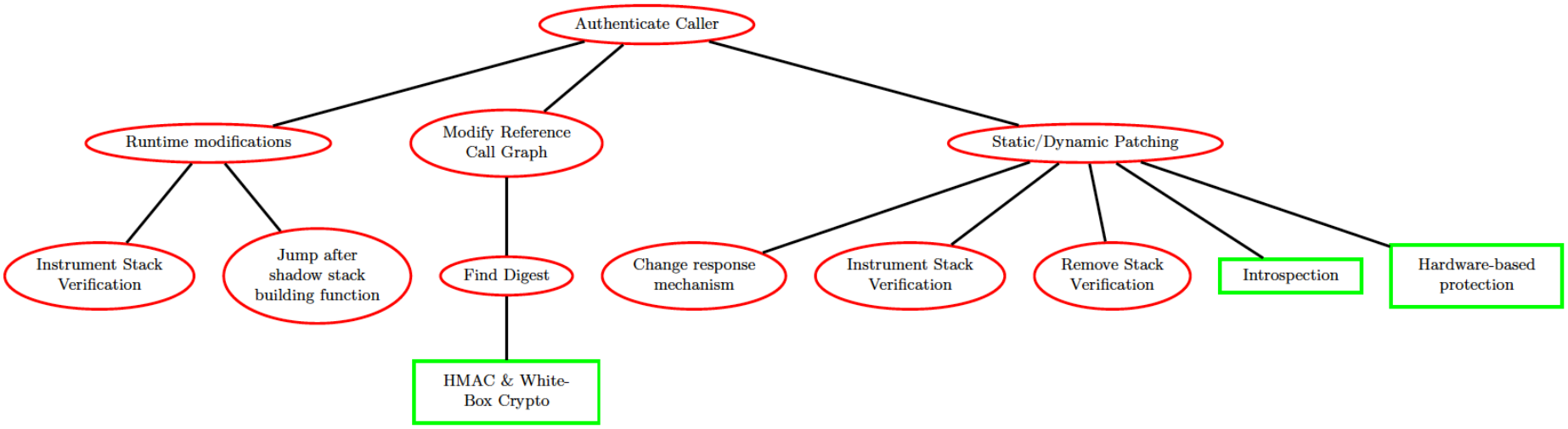
- Print invalid access to command line
- Aborting

- Highly customizable! Only need to edit one method in StackAnalysis.c
- Extensions: Different response depending on sensitive function

Performance evaluation

- Test program: 6 methods, almost no memory consumption, only function calls and 1 printf-statement
- Fast instrumentation, only small increase with increasing number of functions
- Memory overhead: 10 * original memory
- Runtime overhead: Increase of 48% for stack management, depends on number of paths to sensitive functions and number of sensitive functions

Security evaluation



Automated attacks

- Modifying the graph.txt file and patch new checksum
- Overwrite call of verifyStack() with NOPs

DEMO

- Tool demonstration
- Attack: Modifying the graph.txt file
- Attack: Overwrite call to verifyStack()