The Performance Cost of Shadow Stacks and Stack Canaries

Presenters: Karan Bhanot and Lilian Ngweta

Paper by: Thurston H.Y. Dang (EECS, University of California, Berkeley), Petros Maniatis (Google Inc.), and David Wagner (EECS, University of California, Berkeley)
Outline

- Brief overview
- ROP, shadow stacks and canaries
- Problem statement
- Literature review
- Parallel shadow stack and its limitations
- Discussion of results
- Key points of critique
- Conclusion
Brief overview

- Two approaches for control flow defense against Return-Oriented programming (ROP)
- Paper explores **overheads of shadow stacks**
  - Traditional - 10%
  - Parallel (New) - 3.5%
Return-Oriented Programming (ROP)

- Computer security exploit technique
- Attacker hijacks the control flow
- An attacker performs arbitrary operations using instructions available in machine’s memory that end in a return
Shadow stack

- Secondary stack to protect return addresses
- Stores:
  - Pointer to top of shadow stack
  - Return addresses

Figure 1: Traditional shadow stack and main stack [1]
Shadow stack

- On function call, return address is pushed to main and shadow stack
- **On function return, the two return values are compared using**
  - Checking
  - Overwriting
Shadow stack

**Figure 2**: Possible locations for stack shadow instrumentation [1]
Stack canaries

- Special values stored between return address and local variables
- Checked for intactness before return
- If changed, buffer overflow happened
Shadow Stack Challenges

- Expensive
- Overhead origin is unclear
Research Question

- What are the performance costs of using shadow stack for defense against security exploitations when using CFI?
Problem statement

- Quantify the performance cost of shadow stacks
- Parallel shadow stacks implementation
Related work

- Hardware assisted schemes
- Shadow stack protection
- Split the control and data stacks
- Combination of safe and unsafe stack
Parallel Shadow Stack

- minimal, low-cost implementation of the principle of shadow stacks
- placed at a fixed offset from the main stack

Figure 3: Parallel Stack implementation [1]
Implementation

- Call `setupShadowStack` at the beginning of main.
- The `setupShadowStack` function allocates a memory region at a fixed offset from the stack for the parallel shadow stack.
- For traditional shadow stack, memory is allocated at a random location and copied into `%gs:108`
Implementation (continuation)

- Individual source files are then compiled and assembled into separate object files
- Finally the overheard is obtained
Limitations of Parallel Shadow Stacks

- Not as secure as the traditional shadow stack
- Use more memory than the traditional shadow stack
Results

- Overhead values:
  - Traditional shadow stack: 9.69%
  - No-frills parallel shadow stack: 3.51%
- Overwriting has less overhead than checking
Results

- Additional overheads:
  - Return address checking: 0.8%
  - Zeroing stack values: 0.16%
  - Stack canaries: 2.54%
Discussion

- Overhead related to frequency of return
- Correlation coefficient high - 0.73

Figure 4: Correlation between RET instructions and overhead
Discussion

- **Regression model** for overhead to study the effect of return, load and store instructions:

\[
\text{Overhead} = \alpha \times (\% \text{ RETs}) + \beta \times (\% \text{ RETs} \times \% \text{ loads}) + \\
\delta \times (\% \text{ RETs} \times \% \text{ stores})
\]
Discussion

- **Traditional stack has extra overhead of:**
  - Stack pointer
  - Memory transfer instructions
- **Parallel stack overhead comparable to traditional with extra overheads**
- Checks based stacks make attack much harder
Discussion

- Fortran benchmarks were not calculated
- Certain benchmarks like *gcc* were excluded due to their incompatibility with shadow stack
- Indirect jumps lead to more overheads - 27% from 21%
Critique

- Parallel stacks still not efficient enough for widespread deployment
- Use simpler instrumentation similar to the parallel shadow stack rather than using a loop to check the return address
Critique

- Reducing the number of RETs through inlining
- Relaxations or variations of a shadow stack
- Confidence in calculated overheads
Conclusion

- Parallel shadow stacks were introduced
- Performance overhead computed for shadow stacks
  - Overheads exist in shadow stacks with varying values based on implementation
References


Thank you!