

Independence From Obfuscation: A Semantic Framework for Diversity

Riccardo Pucella and Fred B. Schneider

Northeastern U., Cornell U.

CSFW, 2006

Randomization

- Address Space Randomization
 - select memory segment base addresses at load-time
 - select stack ordering and padding at compile-time
- Instruction Set Randomization
 - randomize instruction encodings at load-time
- Other randomizations
 - reorder instructions (e.g., by basic block)
 - rename system API calls
 - randomize dynamic memory allocator

Goals of Randomization

- Slow down attacker
 - attacker must probe to learn randomization keys
 - large enough key = attack infeasible
- Diversify replicas
 - no one kind of attack compromises all replicas
 - remaining replicas continue to provide service
 - behavior difference among replicas alerts admins
 - diversification can be automated
- Attacks of interest
 - code-injection
 - memory/control-flow safety exploits

Randomization vs. Type-systems

- Type-systems for security
 - write code in well-typed source language
 - eliminates memory/control-flow safety vulnerabilities
 - injected code unreachable
 - problem: porting legacy code can be painful or impossible
 - problem: C seems poised to continue its reign of terror...
- Main Questions
 - Is randomization provably less powerful?
 - Can we measure the relative strength of randomizing vs. type-systems?

Motivating Comparisons

- Buffer overflow
 - ASLR: overflow exploit yields useless behavior with high probability
 - Dynamic typing: overflow yields termination
- Leaking a password
 - Cryptography: leaked cyphertext not exploitable with high probability
 - Information flow: no bits of info released

Main Idea

- Randomization = probabilistically-checked, strong (i.e., dynamic) type system
 - failed dynamic type-check terminates with probability p , continues with probability $1-p$
 - as p approaches 1, randomization approximates type-safety
- Notation
 - programs (P), keys (K)
 - morphs: $\tau(P, K)$
 - implementation semantics: $[[P]]_i(inps)$
 - morph semantics: $[[P]]_i^{\tau, K} = [[\tau(P, K)]]_i$

Theory

- “Resistible” attacks
 - attacks are inputs (*inps*)
 - attack *inps* is **resistible** if not all members of set $\{ [[P]]_i^{\tau, Ki}(inps) \mid 1 \leq i \leq n \}$ are observably equivalent
 - irresistible attacks are **interface attacks**
- Concrete Example
 - Toy-C: C-like, imperative, non-type-safe language
 - operational semantics
 - ASLR randomizer τ_{addr}

Type System for Toy-C

- Goal: Devise a (dynamic) type-system for Toy-C that is equivalent to τ_{addr}
- First attempt: (T^{strg})
 - guard all memory dereferences
 - guard instructions simulate all morphs up to next output to see if they agree
 - disagreement = termination with probability p
 - conservatively approximates τ_{addr}
- Second attempt: (T^{info})
 - no guards at memory-reads
 - invalid memory-read yields value with low-integrity label
 - outputting low-integrity data = termination with probability p
 - exact, assuming key collection is finite

Open Problems

- If key collection is infinite, all type-systems are conservative approximations of randomizer (Theorem 5.2)
 - In practice, key collection is finite (but large)
- How to compute probability p ?
 - What is the probability that attack inps will succeed against morph τ with key set K_1, K_2, \dots, K_n ?

Selected References

- F. B. Schneider and R. Pucella. **Independence from Obfuscation: A Semantic Framework for Diversity.** In *Proc. 19th IEEE CSFW*, 2006.
- S. Goldwasser and Y. T. Kalai. **On the Impossibility of Obfuscation with Auxiliary Inputs.** In *Proc. 46th IEEE FOCS*, 2005.
- B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. **On the (Im)possibility of Obfuscating Programs.** In *Proc. 21st CRYPTO*, Vol. 2139 of LNCS, pp. 1-18, 2001.