Instruction Set Randomization (ISR)

- Observation: Code injection attacks succeed if the injected code is compatible with the execution environment.
  → Randomizing the underlying system's instructions set.

Outline

- Instruction Set Randomization (ISR)
- Incremental Key Guessing
- Return Attack
- Jump Attack
- Extended Attack
- Attack Requirements
- Experiment
- Countermeasures
Questions

- ISR prevent any type of code-injection attacks?
- Stack and heap-based attack?
- Return-to-libc?
- Return Oriented Programming?

Incremental Key Guessing

- How to determine the randomization key?
  - Brute force: key space very large
  - Incremental key guessing:
    - Challenge: distinguish correct and incorrect guesses
    - Observation: in x86 architecture, some useful instruction: have 1 or 2 bytes: Return (1 byte) and Jump (2 bytes)

Return Attack

- near return (0xc3)
- 1 byte, 256 guesses.
- Problem: compromise stack, server will usually crash soon after correct guess

Jump Attack

- 2 byte instruction -> $2^{16}$ guesses
- Wrong guess - application crash, the socket is closed immediately
- Correct guess – create an infinite loop, the socket remains open.
Incremental Key Breaking (Jump attack)

- 256 attempts for the jump opcode at base-1

Figure 3. Incremental jump attack.

Extended attack

- Need 8 masks
- Use a combination of short (0xeb) and near jump (0xe9)
- Does not produce the infinite loop on target.

Figure 5. Extended attack.

Eliminate False Positives (Jump Attack)

- Harmless instructions: 0x90, 0xfc, 0x42
- Eliminate the guess if apparently correct behavior does not occur

Conditional Jumps
- 16 conditional jumps (0x70–0x7f)
- Opposite conditions differ by last bit
  - JZ (0x74), JNZ (0x75)
- At most 32 attempts to find first infinite loop
  - 0x00, 0x10, ..., 0xf0, 0x01, 0x11, ..., 0xf1
  - At most 24 guesses for each additional key after the first 2 key bytes guessed.

Figure 4. Eliminating false positives.
Eliminate False Positives (Extended Attack)

Interrupt instruction (0xc020)
- If the guess is wrong and decrypts to a harmless instruction → crash

Figure 5. Extended attack.

Attack Requirements

- No re-randomization
- Remotely observable behaviors
- Vulnerability that allows injection code at known address
- Simple encryption scheme (XOR)

Would these things make the attack impractical? Will ASLR affect these attacks?

Experiment

Target:
- Simple echo RISE- protected server with a buffer overflow vulnerability
- Forks a process for each request
- Modified RISE to initialize the encryption keys before fork (same key for all child processes)
- Turned off Fedora ASLR

Results

-100 byte key in 6 mins (Jump attack), 2mins (return attack).

Table 1. Jump attack results (averages over all trials).
Countermeasures

Breaking requirements:
- Re-randomize with different keys periodically
  - Monitor the process crashes
- Stronger encryption
  - Use a stronger cipher like AES
- Turn on ASLR

N-Variant systems

N-Variant Systems
A Secretless Framework for Security through Diversity

http://www.nvariant.org
USENIX06
N-Variant Systems

Key ideas:
- Any attack that succeeds against Variant A must cause Variant B to crash.
- Does not rely on keeping any secrets

Strong points
- Unlike ISR, and ASLR, no need to keep secrets → secret-breaking, guessing attacks (FEEB paper attack), insider attacks
- While WθX, CFI limit to specific attack classes, N-variant defense against any attacker capability
- Memory corruption attacks, code injection attacks, accessing absolute addresses.

Thank You for Your Attention!