FONDAZIONE BRUNO KESSLER

Software protection and dynamic analysis attacks

Mariano Ceccato [1], *Anirban Majumdar* [2], *Paolo Tonella* [1]

(1) FBK-IRST, Trento, Italy

(2) University of Trento, Italy

Motivation

Dynamic analysis is known to be one of the most powerful tools available to attackers, however its study was somewhat neglected in software security analysis & modeling.

With respect to software engineers using dynamic analysis, **attackers**:

- have a more focused objective;
- can easily accommodate wrong deductions.

FONDAZIONE BRUNO KESSLER

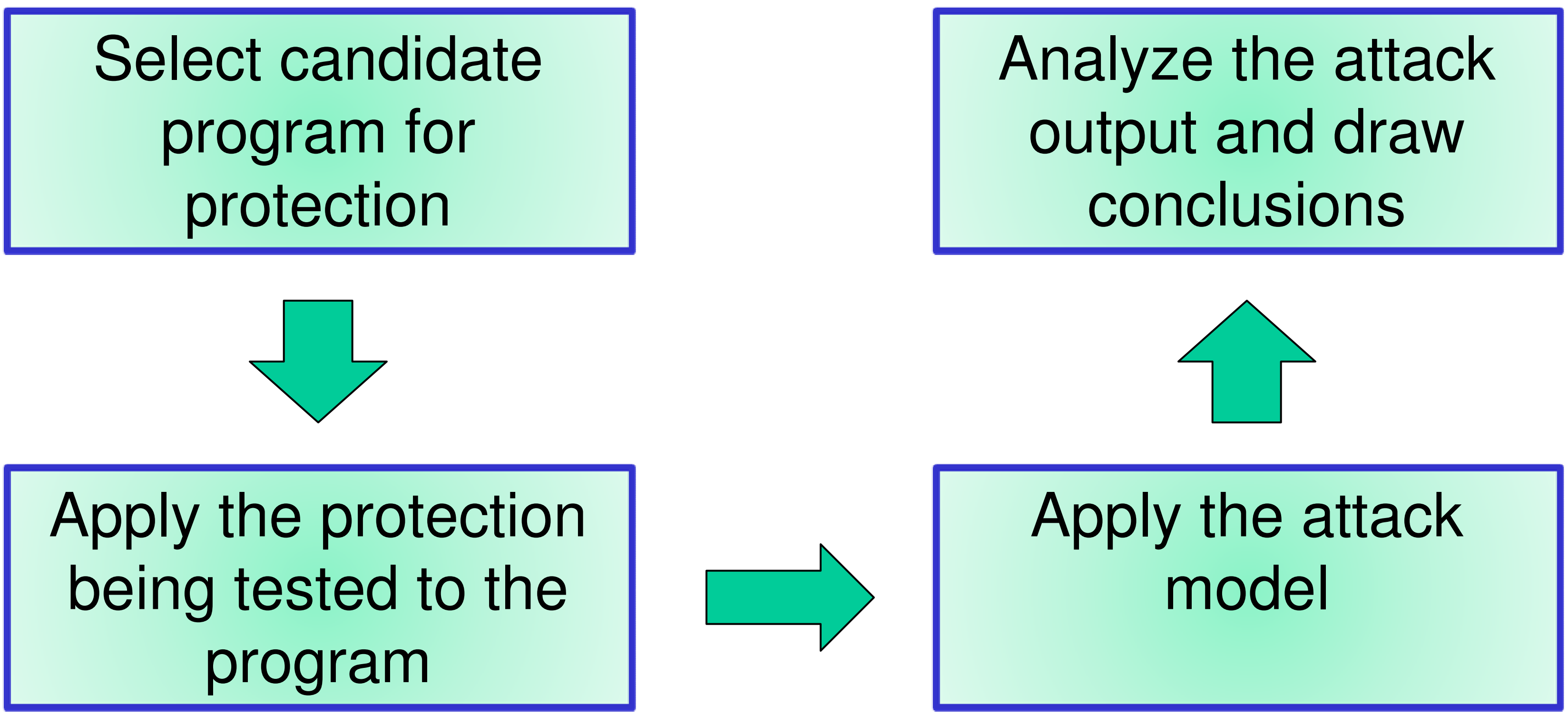
Software protection

- Check-summing, guards.
- Server-side execution (e.g., barrier slicing) and assertion checking.
- Obfuscation (orthogonal replacement).
- Encrypted execution

Reverse engineering attacks:

1. static analysis;
2. dynamic analysis;
3. program comprehension.

Testing software protection techniques



This testing framework can be applied at various abstraction levels.

Attack model

Assumptions:

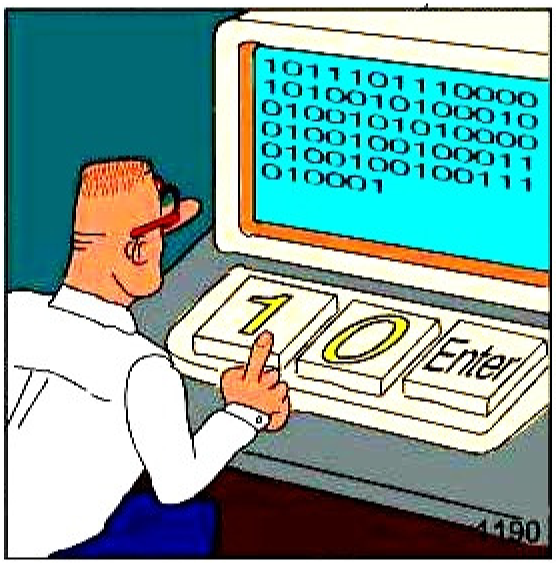
- what the attacker is or is not capable of doing;
- what tools the attacker is supposed to use;
- what information the attacker has access to.

Output:

- secret property of the program, whose integrity is supposed to be protected;
    - accuracy of property localization;
    - capability of tampering (integrity violation).

Dynamic analysis tools

- ❑Debuggers
- ❑Tracers
- ❑Instrumentors
- ❑Sniffers
- ❑Emulators
- ❑Dynamic slicers
- ❑Feature location



… in addition to smart, opportunistic code understanding.

Scenario 1:
breaking check sums

Bob, the attacker, can:

1. Trace all data read from memory.
2. Compare traced data with program's (binary/byte) code.
3. Locate check summing operations.
4. Modify code or execution to forge correct check sum data.
5. Tamper with code integrity without being detected.

Scenario 2:
breaking server-side checks

Bob, the attacker, can:

1. Sniff all messages exchanged with server.
2. Identify security sensitive messages.
3. Locate code producing such messages.
4. Modify code or execution to forge legal messages.
5. Tamper with code integrity without being detected.

Scenario 3:
breaking obfuscation

Bob, the attacker, can:

1. Trace I/O operations that cannot be obfuscated.
2. Associate obfuscated code portions with I/O related computations.
3. Locate code responsible for security sensitive computations.
4. Modify code repeatedly, until desired tampering is achieved.

Scenario 4:
breaking (naive) encrypted execution

Bob, the attacker, can:

1. Trace program's state.
2. Follow data dependencies from obfuscated instructions to memory change.
3. Locate code responsible for the execution of each obfuscated opcode.
4. Inject malicious opcodes into the program's instructions.

Common attack pattern

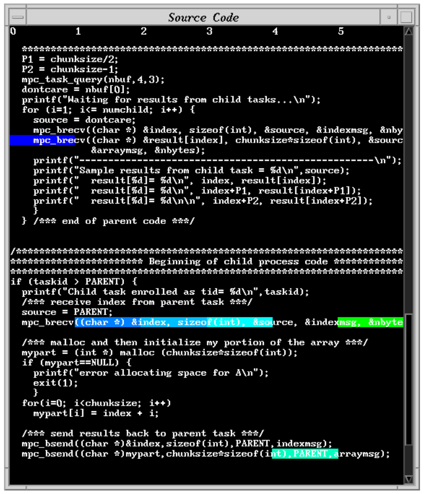
1. Observe: data (program's state), statements being executed, messages exchanged.
2. Identify: security sensitive data.
3. Locate: code containing security sensitive computation.
4. Modify & tamper: violate code or execution integrity without being detected.

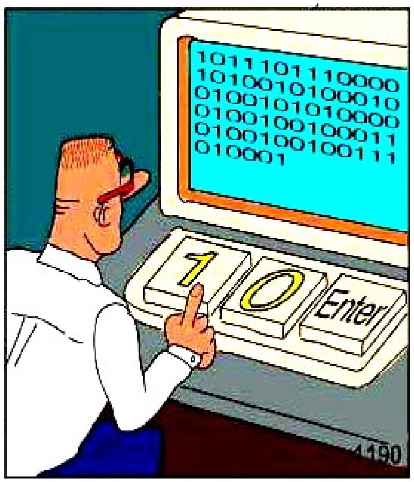
Useful tools & techniques

1. Observe: debuggers, tracers, instrumentors, sniffers.
2. Identify: string matching, pattern matching, clustering, concept analysis.
3. Locate: dynamic slicing, feature location, reconnaissance, code inspection, reverse engineering.
4. Modify & tamper: program transformation tools, emulators.

Opportunistic trial and error

Modify code and/or execution.





Check if tampering is detected.

Building a realistic attack model

- ❖ Change your hat!
- ❖ Apply the common pattern.
- ❖ Be aware of existing tools and techniques.
- ❖ Model opportunistic attempts, based on trial and error.
- ❖ Open your mind: attackers have a lot of fantasy!

Conclusions

In order to build **strong** software integrity protections, we need a deep understanding of dynamic analysis attacks:

- characteristics of available tools and techniques;
- attack model;
- experiments testing the resilience of existing protection techniques;
- empirical studies.