KATHOLIEKE UNIVERSITEIT LEUVEN

Towards a Formal Model for Software Tamper Resistance

Brecht Wyseur
October 2009, Riva del Garda, Italy

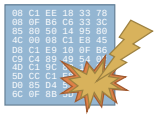
Joint work with **Cataldo Basile, Stefano Di Carlo, Thomas Herlea, and Jasvir Nagra**

Outline

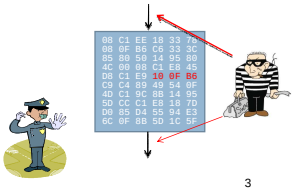
- Introduction: software tamper resistance
- Predicates, traces, …
- Tamper proofing: the definition
- Relation to existing notions
  - Obfuscation
  - Non-malleable obfuscation
- Tamper verification techniques
- Theory put into reality

Software Tamper Resistance

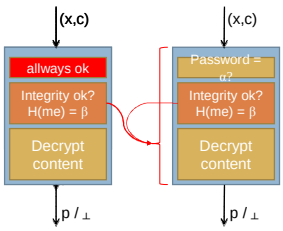
- Tamper **Protection** (proactive)
  - Obfuscation



- Tamper **Detection** (reactive)
  - Act when tampering occurred



Software Tamper Resistance



- (self-)checking techniques
  - Verify the integrity of a program by computing a checksum (or hash) of the code
- Cloning attack
  - Execute a tampered copy
  - Refer any integrity verification to the untampered copy (distinguish between read code and execute code)

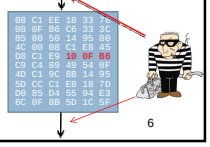
Towards a notion of tampering

- What kind of techniques are good tamper protection techniques? (both tamper proofing as tamper verification)
  - Need for a model
    - Surprise: we did not find a suitable model that explained what "tampering" *really* is.
- Candidate techniques: test whether any bit of the implementation has been modified
  - Context: "White-box" – an adversary can always modify an implementation.
    - Intuition: TRS wants to prevent *useful* modification

Towards a notion of Tampering

- Possible protection techniques: obfuscation, Self-checksumming, …
  - Are these suitable techniques?
    - Is it necessary that *no* information leaks?
    - I don't care about … code; implementation details; non-sensitive operations; …
- What is executed is what matters
  - Execution Traces

Execution trace:
`08 C1 EE 18 33 78 08 0F B6 C6 33 3C 85 80 …`



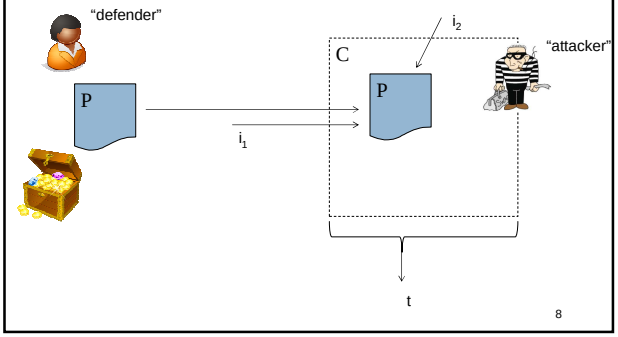
Execution Traces

- Traces $\mathcal{T} = \{\text{elementary computation}, \text{timestamp}\}^*$
  - A sequence of elementary computations (e.g., instructions that hit the CPU)
  - It shows: when an adversary skips a part of a program; executes out of order; runs other programs in parallel (debugger, …)
  - A bit idealistic…
- Execution engine E

$$E: \underset{P,}{\text{binary}} \times \underset{C,}{\text{context}} \times \underset{I}{\text{inputs}} \longrightarrow \underset{t}{\mathcal{T}}$$

7

Overview



8

Defender's Objectives

- Predicate **D**

$$D: \underset{t}{\mathcal{T}} \longrightarrow \begin{cases} 1 \text{ if goals are met} \\ 0 \text{ else} \end{cases} \{0,1\}$$

  - Example: conjunction of sub-goals

$$D = (D_1 \wedge D_2 \wedge \cdots \wedge D_n) \vee \overline{D_1 \vee D_2 \vee \cdots \vee D_n} = \bigwedge_i D_i \vee \overline{\bigvee_i D_i}$$

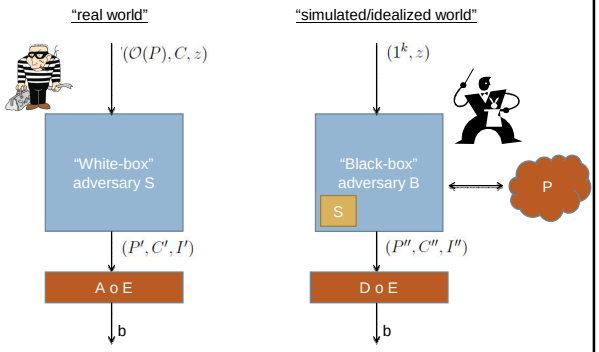
  "Either all sub-goals are satisfied, or none of them"
- Attackers objective

$$\begin{aligned} A &= \overline{D} \\ &= \overline{\textstyle\bigwedge_i D_i \vee \overline{\bigvee_i D_i}} \\ &= \textstyle\bigvee_i D_i \wedge \bigvee_i \overline{D_i} \end{aligned}$$

Adversary and Defender share sub-goals!

9

Tamper Protection Technique



10

Tamper Protection Technique

DEFINITION 2 (Tamper protection technique). *A probabilistic Turing Machine $\mathcal{O}$ is a D-**tamper protection technique** for the class of programs $\mathcal{P}$ in the context C, if $\forall P \in \mathcal{P}$ all of the following properties are satisfied:*

- correctness – *$\mathcal{O}$ turns the program P into a functionally equivalent program $\mathcal{O}(P)$; there exists a polynomial p, such that if P halts in t steps, $\mathcal{O}(P)$ will halt in $p(t)$ steps, and $|\mathcal{O}(P)| \leq p(P)$.*
- soundness – *The probability that a polynomial-time adversary S given the code of $\mathcal{O}(P)$ is able to* tamper *with the program such that the defender's goals D are falsified in a way that a polynomial-time algorithm B given black-box access to the program P cannot do (running in the pre-fixed context C) is negligible. Formally,*

$$\left| \begin{array}{l} Pr[(P', C', I') \leftarrow S(\mathcal{O}(P), C, z) : A(E(P', C', I')) = 1] \\ -Pr[(P'', C'', I'') \leftarrow B^P(1^k, z) : D(E(P'', C'', I'') = 0] \end{array} \right| \leq neg(|P|, |I|), \quad (1)$$

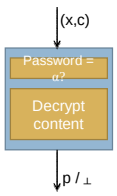
the probabilities taken over the coin tosses of S and B; z denotes dependable auxiliary input.

11

Example

- Digital Rights Management example: restricted decryption of content.
  - D1 – password correct
  - D2 – decrypt content
  - D-predicate: $D = (D_1 \wedge D_2) \vee \overline{D_1 \vee D_2}$

$$\mathcal{O}(P)(x, c) = \begin{cases} \text{Dec}_k(c) \text{ if } x = \alpha \\ \perp \text{ else,} \end{cases}$$



  - "Black-box" success probability: $2^{-|\alpha|}$
  - "White-box" success probability: 1 (A = D_2)

12

Relation between Obf. and TR

- Obfuscation VBBP [Barak et al., 2001]

$$\left|\Pr\left[B(1^s, \mathcal{Q}(P)) = \pi(P)\right] - \Pr\left[C_B^P(1^s) = \pi(P)\right]\right| \leq neg(|P|)$$

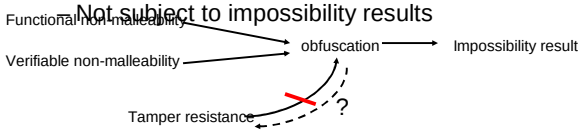
  - Impossible to achieve … (for all programs)
- Non-malleable Obfuscation [Canetti and Varia, 2009]
  - Functional non-malleabilitv:

$$|\Pr[P \leftarrow A(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C, D) = 1]$$
$$-\Pr[Q \leftarrow S^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } E(C, D) = 1]| < \frac{1}{\rho(n)}$$

  - Functional non-malleability ➔ Obfuscation
  - Verifiable non-malleability ➔ Obfuscation

Relation between Obf. and TR

- Our definition does **NOT** imply obfuscation
  - TR not stronger than Obfuscation
    - → see example
  - Not subject to impossibility results



Open question: is obfuscation stronger than TR?

TR >! Obfuscation -- Example

- Augmented DRM example
  - When password is correct (D_1), decrypt content c (D_2), and present the secret s (D_3)
  - P is implemented as O(P) as follows:

$$\beta = \mathcal{H}(\alpha)$$
$$\gamma = \alpha + k$$

$$\mathcal{O}(P)(x, c) = (y_1, y_2) = \left(\mathrm{Dec}_{\gamma \oplus x}(c); \begin{cases} s \text{ if } \mathcal{H}(x) = \beta \\ 0 \text{ else} \end{cases}\right)$$

  - "Obfuscation-adversary": can extract the secret s (wins under the predicate-based obfuscation definition)
  - "Tampering-adversary": can only decrypt when able to find α, 'the' pre-image of β

Outline

- Introduction: software tamper resistance
- Predicates, traces, …
- Tamper proofing: the definition
- Relation to existing notions
  - Obfuscation
  - Non-malleable obfuscation
- Tamper verification techniques
- Theory put into reality

Tamper Detection (Reactive Prot.)

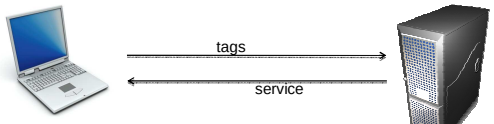
- At run-time: incomplete traces to be verified

DEFINITION 4 (Satisfactory Traces). *Let π be a predicate on traces. A finite trace prefix is called* ***π-satisfactory*** *if there exists a trace suffix such that $\pi(\text{prefix}||\text{suffix}) = 1$.*

DEFINITION 5 (Unsatisfactory Traces). *Let π be a predicate on traces. A finite trace prefix is called* ***π-unsatisfactory*** *if there exists a trace suffix such that $\pi(\text{prefix}||\text{suffix}) = 0$.*

- Remote Evaluation:
  - Where the defender can monitor the attacker,
  - The attacker depends on the continued cooperation of the defender in order for the program to achieve the **shared sub-goals**

Tags (attestations)



$$\begin{array}{ccc} \mathcal{T} & \xrightarrow{D} & \{0,1\} \\ \downarrow G & & \\ \text{TAGS} & \xrightarrow{V} & \{0,1\} \end{array}$$

- Tag verification: V predicate
- Soundness of tag-verification:

$$\Pr[V(G(t)) \neq D(t)] \leq neg(|\text{TAGS}|)$$

Tamper Verification Technique

DEFINITION 6 (Tamper verification technique). *A probabilistic Turing Machine $\mathcal{O}$ is a D-**tamper verification technique** for the class of programs $\mathcal{P}$ in the context of C, if $\forall P \in \mathcal{P}$ the correctness definition of Definition 2 is satisfied, and the following soundness definition holds:*

(Soundness) – *The probability that a polynomial-time adversary S given the code of $\mathcal{O}(P)$ is able to defeat the tag verification (trigger a false positive at run-time) such that the defender's goals D are falsified in a way that a polynomial-time algorithm B given only black-box access to the program P cannot is negligible. Formally, $\forall i$ it needs to hold that*

$$\left| \begin{array}{l} Pr[(P', C', I', \gamma_i') \leftarrow S(\mathcal{O}(P), C, z) : A(E(P', C', I')) = 1 \wedge V(\gamma_i', z) = 1] \\ -Pr[(P'', C'', I'', \gamma_i'') \leftarrow B^P(1^k, z) : D(E(P'', C'', I'') = 0 \wedge V(\gamma_i'', z) = 1] \end{array} \right| \leq neg(k),$$

where the γ_i denote the i-th tag; the probabilities taken over the coin tosses of S and B; z denotes dependable auxiliary input; and k is a parameter in polynomial relation to $|P|, |I|$ and $|TAGS|$.

- Adversary's task: tamper with the program and subvert the verification predicate

19

Invariants Monitoring

- A practical verification scheme based on concept of *programming by contract* and *software invariants*.
  - Programming by contract – software elements are meant to satisfy well understood specifications or business goals [Bertrand Meyer]
  - Software invariants – describe conditions on software elements according to specifications
- Violation of invariants = disrespect of original specification

Set of variables for portion P* of program P

$\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$

Predicates valid over P*

$I = \{I_1, I_2, \ldots, I_m\}$

20

Conclusions

- A formal definition of tampering has been presented, based on the defender's goals
- Attacker and defender share sub-goals
- Modeling in terms of Execution Traces
  - Open problem: how to work with them in practice?
- Relationship with obfuscation (Tamper Protection)
  - TR does not need to be stronger than Obfuscation (in contrast to conclusion of [Canetti et al.])
  - Open problem: does obfuscation imply TR?
- Reactive protection (Tamper Detection): tags
- Example in practice: Invariants Monitoring

21

- Questions?

- Full paper at https://www.cosic.esat.kuleuven.be/publications/article-1280.pdf

22