Purdue University

Center for Education and Research in Information Assurance and Security

SOFTWARE TAMPERPROOFING

Mikhail Atallah

Hoi Chang

Tim Korb

John Rice

CERIAS Conference, April 27, 2001

TAMPERPROOFING

- The goal is to prevent unauthorized use of a program P. The mechanism is to put in code to check authorization and that prevents the program from operating properly if **ANY** change is made in it.
- This might prevent the piracy of MS Word or the clandestine use of a nuclear bomb design code.

MECHANISMS 1 & 2

1. Insert **authorization checks**: passwords, biometrics, machine/system prints, … If any of these checks fail then appropriate action is taken, e.g., crash machine, notify owner, corrupt computations, etc.
2. Insert **guards**: Compute check sums on the code of P.

PROBLEMS

1. The authorizations and guards can be located (even in binary code) by their special natures. Authorizations ask for information and guards use program statements as data.
2. Thus an attacker can remove or bypass the authorizations and/or guards.

LINE OF DEFENSE 3

3. Insert **multiple guards**: They guard each other as well as the program P. Make a complex network of guards that protect one another so that they have to all be removed before the guarding fails.

PROBLEM: A determined attacker might be able to find and remove them all.

LINES OF DEFENSE 4 & 5

4. **Obfuscate** the authorization and guard codes so they are hard to identify; e.g., hide 1789 and 4969 in their product 8889541; it is very hard to find the true code in the final product.

PROBLEM: Obfuscated code may be hard to understand but it is “strange” so one can eventually identify and remove it.

5. **Insert repairing guards**. They correct errors introduced into P; if they are deleted then P does not work properly.

LINE OF DEFENSE 6

6. Mix (**tangle**) these codes with pieces of P's code and obfuscate it all together. Then one cannot remove the obfuscated code without corrupting P.

PROBLEM: Obfuscated code is tough to untangle but the toughness depends on the length. These code fragments tend to be fairly short.

LINE OF DEFENSE 7

7. Introduce **dummy code** which does not affect P's operation. Tangle this in with the authorization, guard and P's code, then obfuscate. One can make this as hard to untangle as one wants. One can automate the generation of appropriate dummy code and doing the obfuscation.

SUMMARY

- There must be defenses for all kinds of attacks on the integrity of P. The above describes just the main theme of the defense; we list other attacks we can defend against. Some of these may be machine/system dependent in their details

OTHER TYPES OF ATTACKS

1. **Code analysis**: Read P using analysis/debugging tools.
2. **Trace analysis:** Trace the paths & values in P; simulate it.
3. **Reverse engineer** the obfuscated parts of P.
4. **Copy attack:** The guards check Copy #1 while executing Copy #2.
5. **Multiple copy attack:** Compare 10K copies of P to isolate and identify various functionalities.
6. **Subprogram spy attack:** Replace a standard utility with a spy.

DEMONSTRATION

- The **OnGuard Tool** works on Intel binary code using Visual C++ output of the compiler to insert guards into P.
- The **GUI** prototype inserts markers into source code about types and locations of guards desired. It can also insert standard or customized authorization code.

DEMO DETAILS

- P ~ 100 lines and has 6 guards: 2 check sums and 4 repairs (2 small and 2 larger).
- Basic obfuscation (basic block shuffling) and watermarking (garbage instructions between basic blocks) are included.
- There is no increase in object file size.
- The guarding is mostly automated