Could Software Watermarks Express Both Rules and Assurances?

Prof. Clark Thomborson

Presentation to the
ReTRUST Group
Villach, Austria
11th March 2008

Agenda

- What is security?
- What is software watermarking, and how is it used?
- Are we missing any cases?

What is Security?
(A Taxonomic Approach)

*The first step in **wisdom** is to know the things themselves; this notion consists in having a **true idea of the objects**; objects are distinguished and known by **classifying them methodically** and **giving them appropriate names**. Therefore, classification and name-giving will be the **foundation of our science**.*

Carolus Linnæus, ***Systema Naturæ***, 1735

(from Lindqvist and Jonsson, “How to Systematically Classify Computer Security Intrusions”, 1997.)

Standard Taxonomy of Security

1. **Confidentiality**: no one is allowed to read, unless they are authorised.
2. **Integrity**: no one is allowed to write, unless they are authorised.
3. **Availability**: all authorised reads and writes will be performed by the system.

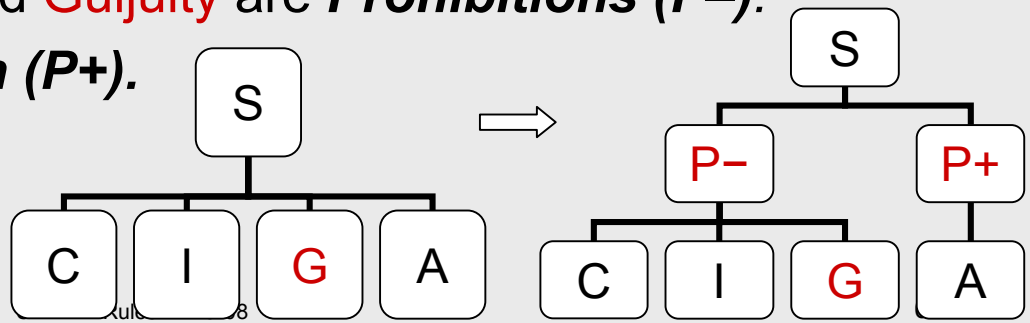
- **Authorisation**: giving someone the authority to do something.
- **Authentication**: being assured of someone's identity.
- **Identification**: knowing someone's name or ID#.
- **Auditing**: maintaining (and reviewing) records of security decisions.

A Multi-Level Hierarchy

- **Static security**: the Confidentiality, Integrity, and Availability properties of a system.
- **Dynamic security**: the technical processes which assure static security.
  - The gold standard: Authentication, Authorisation, Audit.
  - Defense in depth: Prevention, Detection, Response.
- **Security governance**: the “people processes” which develop and maintain a secure system.
  - Governors set budgets and delegate their responsibilities for Specification, Implementation, and Assurance.

Generalized Static Security

- Confidentiality, Integrity, and Availability are properties of read and write operations on data objects.
- What about executable objects?
  - Unix directories have “rwx” permission bits.
  - XXXX-ity: all executions must be authorised.
  - GuiJu FangYuan ZhiZhiYe ⇒ a new English adjective “Guijuity” (coined in Beijing, 2007).
- At the top of a taxonomy, we should have a clear and important distinction, not a long list of alternatives.
  - Confidentiality, Integrity, and Guijuity are ***Prohibitions (P–)***.
  - Availability is a ***Permission (P+).***

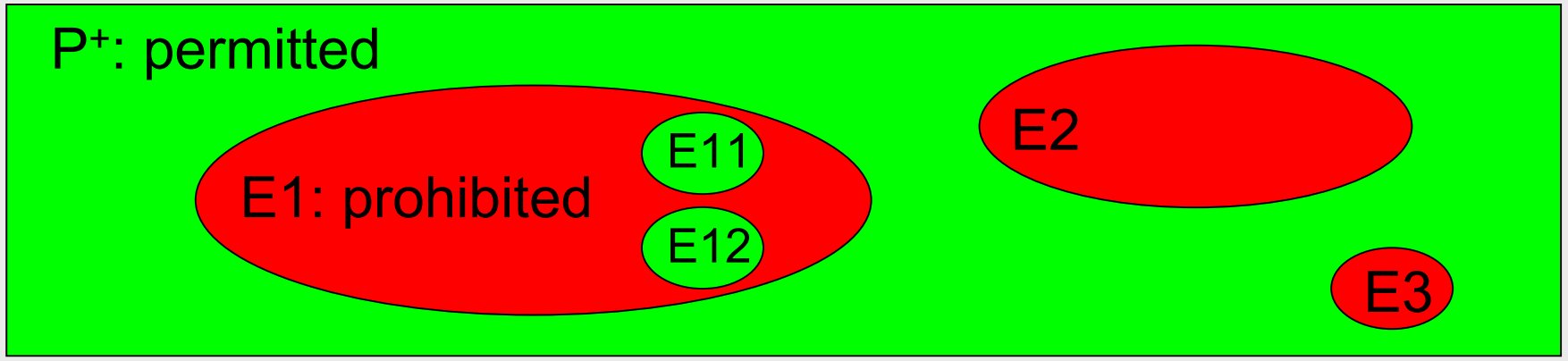


Prohibitions and Permissions

- **Prohibition**: prevent an action.
- **Permission**: allow an action.
- There are two types of action-secure systems:
  - In a **prohibitive system**, all actions are **prohibited** by default. Permissions are granted in special cases, e.g. to authorised individuals.
  - In a **permissive system**, all actions are **permitted** by default. Prohibitions are special cases, e.g. when an individual attempts to access a secure system.
- Prohibitive systems have permissive subsystems.
- Permissive systems have prohibitive subsystems.

Recursive Security

- Prohibitions, i.e. “Thou shalt not kill.”
  - General rule: An action (in some range P^-) is **prohibited**, with exceptions (**permissions**) E1, E2, E3, ...
- Permissions, i.e. a “licence to kill” (James Bond).
  - General rule: An action in P^+ is **permitted**, with exceptions (**prohibitions**) E1, E2, E3, ...
- Static security is a hierarchy of controls on actions:

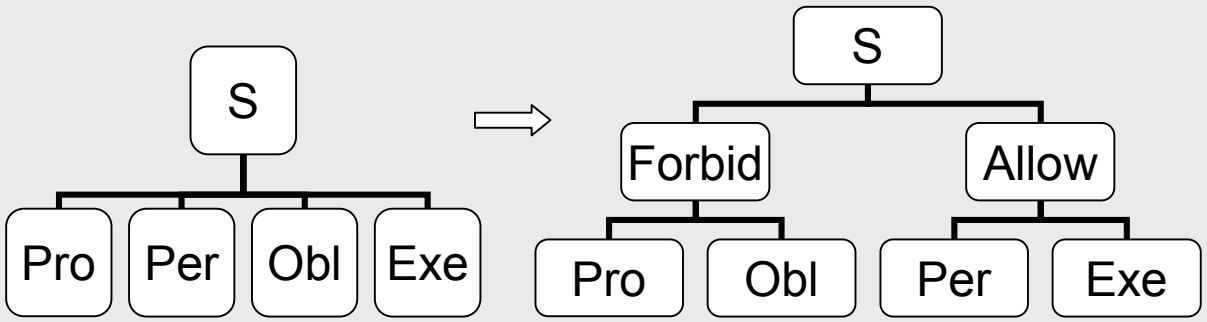


Is Our Taxonomy Complete?

- Prohibitions and permissions are properties of **hierarchical systems**, such as a judicial system.
  - Most legal controls ("laws") are prohibitive: they prohibit certain actions, with some exceptions (permissions).
- Contracts are non-hierarchical (agreed between **peers**), and consist mostly of requirements to act (with some exceptions):
  - **Obligations** are promises to do something in the future.
  - **Exemptions** are exceptions to an obligation.
- Obligations and exemptions are not well-modeled by action-security rules. Inaction security!
  - Obligations arise occasionally in the law, e.g. a doctor's "duty of care" or a trustee's fiduciary responsibility.

Forbiddances and Allowances

- **Obligations** are forbidden inactions; **Prohibitions** are forbidden actions.
  - When we take out a loan, we are obligated to repay it. We are forbidden from never repaying.
- **Exemptions** are allowed inactions; **Permissions** are allowed actions.
  - In the English legal tradition, a court can not compel a person to give evidence which would incriminate their spouse (husband or wife). This is an exemption from a general obligation to give evidence.
- We have added a new level to our hierarchy.



A Taxonomy of Security

- Three types of security: Static, Dynamic, Governance.
- **Static**: the rules.
  - Prohibitions, permissions, obligations, exemptions.
- **Dynamic**: how the rules are enforced.
  - The gold standard (Authentication, Authorisation, Audit).
  - Defense in depth (Prevention, Detection, Response).
- **Governance**: how the rules are made.
  - Governors set budgets and delegate responsibilities for Specification, Implementation, and Assurance.
  - We have defined a system consisting of a Secure Subsystem and its Governors.
  - Governors may themselves be regulated.

- Research question #1: Can governors govern themselves?
  - Sed quis custodiet ipsos custodes?
  - Can systems secure themselves, or are there only secure subsystems?
- Research question #2: Can the dynamic layer be more clearly defined?

Reviewing our Agenda

1. What is security?
2. What is software watermarking, and how is it used?
3. Are we missing any cases?

Developing Use Cases

- We can find use cases at the dynamic and governance layers of our hierarchy.
  - A rule (static security) is not a use: we need an actor, a system, and a desired action (or set of actions).
  - We can also look for misuses: malicious actors who take advantage of a system.
  - There are also “confuses” – authorised users who cause damage by mistake.
- Several years ago, I developed dynamic-use cases for various software protection technologies.
  - My purpose was to explain the functional differences between these technologies.
  - Let’s focus on the software watermarking entries...

Defense in Depth for Software

1. Prevention:
   a) Deter attacks on forbiddances (use obfuscation, encryption, watermarking, cryptographic hashes, or trustworthy computing).
   b) Deter attacks on allowances (use replication, or resilient algorithms).
2. Detection:
   a) Monitor subjects (user logs), relative to a user ID. Use biometrics, ID tokens, or passwords.
   b) Monitor actions (execution logs, intrusion detectors), relative to a code ID: cryptographic hashing, watermarking.
   c) Monitor objects (object logs), relative to an object ID: hashing, watermarking.
3. Response:
   a) Ask for help: Set off an alarm (which may be silent – steganographic), then wait for an enforcement agent.
   b) Self-help: Self-destructive or self-repairing systems.

Watermarks are used at all three layers! (Is there only one type of watermark, or are we using the same word for different things?)

Software Watermarking

Key taxonomic questions:

- **Where** is the watermark embedded?
  - ⇒ **How** is the watermark embedded?
- **When** is the watermark embedded?
- **Why** is the watermark embedded?
  - ⇒ **What** are its desired properties?

Software Watermarking Systems

- An embedder $E(P; W; k) \rightarrow P_w$ embeds a message (the watermark) W into a program P using secret key k, yielding a watermarked program P_w
- An extractor $R(P_w ; \ldots) \rightarrow W$ extracts W from P_w
  - In an invisible watermarking system, R (or a parameter) is a secret.
  - In visible watermarking, R is well-publicised (ideally obvious).
- The attack set A and goal G model the security threat.
  - For a robust watermark, the attacker's goal is a false-negative extraction, usually by creating an attacked object $a(P_w)$, with $R(a(P_w); \ldots) \neq W$ such that P_w is valuable.
  - For a fragile watermark, the attacker's goal is a false-positive: $R(a(P_w); \ldots) = W$ such that $P_w \neq P$ is valuable.
  - A protocol attack is a substitution of R' for R, causing a false-negative or false-positive extraction

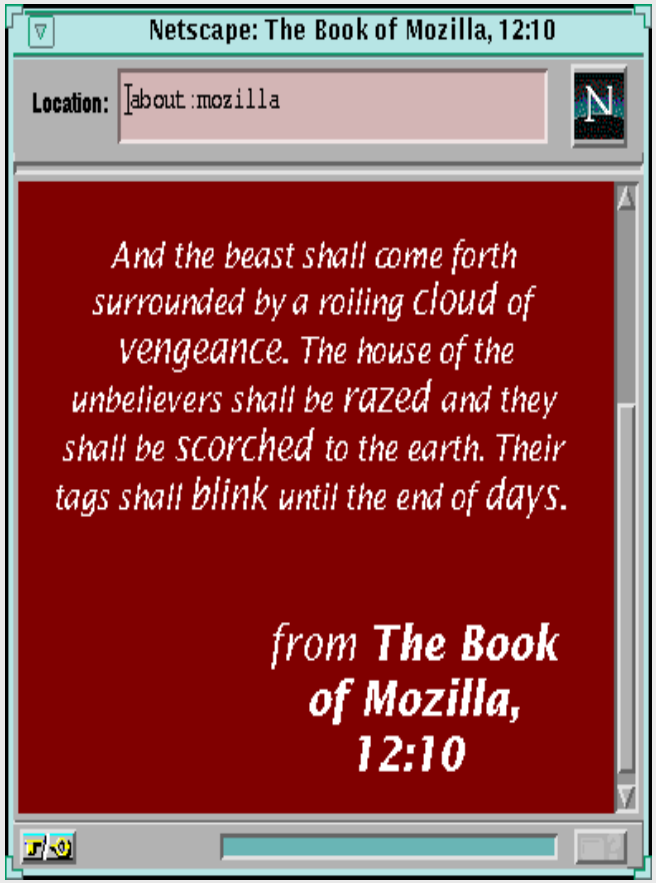
Where Software Watermarks are Embedded

- Static code watermarks are stored in the section of the executable that contains instructions.
- Static data watermarks are stored in other sections of the executable
- Static watermarks are extracted without executing (or emulating) the code.
  - A watermark extractor is a special-purpose static analysis.
  - Extraction is inexpensive, but we don't know of any robust static code watermarks. Attackers can easily modify the watermarked code to create an unwatermarked (false-negative) version.

Dynamic Watermarks

- **Easter Eggs** are revealed to any end-user who types a special input sequence.
- Other dynamic behaviour watermarks:
  - **Execution Trace Watermarks** are carried in the instruction execution sequence of a program, when it is given a special input sequence (possibly null).
  - **Data Structure Watermarks** are built by a program, when it is given a special input.
  - **Data Value Watermarks** are produced by a program on a surreptitious channel, when it is given a special input.

Easter Eggs



- The watermark is visible – if you know where to look!
- Not very robust, after the secret is published.
- See www.eeggs.com

Dynamic Data Structure Watermarks

- The embedder inserts code in the program, so that it creates a recognisable data structure when given specific input (the key).
- Details are given in our POPL'99 paper, and in two published patent applications.
  - Assigned to Auckland UniServices Ltd.
  - I would very much like to find licensed uses for this technology!
- Implemented at http://www.cs.arizona.edu/sandmark/ (2000-)
- Experimental findings by Palsberg et al. (2001):
  - JavaWiz adds less than 10 kilobytes of code on average.
  - Embedding a watermark takes less than 20 seconds.
  - Watermarking increases a program's execution time by less than 7%.
  - Watermark retrieval takes about 1 minute per megabyte of heap.

Thread-Based Watermarks

- A dynamic watermark is expressed in the thread-switching behaviour of a program, when given a specific input (the key).
  - The thread-switches are controlled by non-nested locks.
  - NZ Patent 533208, US Patent App 2005/0262490
  - Article in IH'04; Jas Nagra's PhD thesis, 2006
- The embedder inserts tamper-proofing sequences which closely resemble the watermark sequences but which, if removed, will cause the program to behave incorrectly.
  - This is a "self-help" response mechanism.

SW Watermarking
(Review of Taxonomic Questions)

- Where is the watermark embedded?
  - ⇒How is the watermark embedded?
- When is the watermark embedded?
- Why is the watermark embedded?
  - ⇒What are its desired properties?

Active Watermarks

- We can embed a watermark during a design step ("active watermarking": Kahng et al., 2001).
  - IC designs may carry watermarks in place-route constraints.
  - Register assignments during compilation can encode a software watermark, however such watermarks are insecure because they can be easily removed by an adversary.
- Most software watermarks are "passive", i.e. inserted at or near the end of the design process.

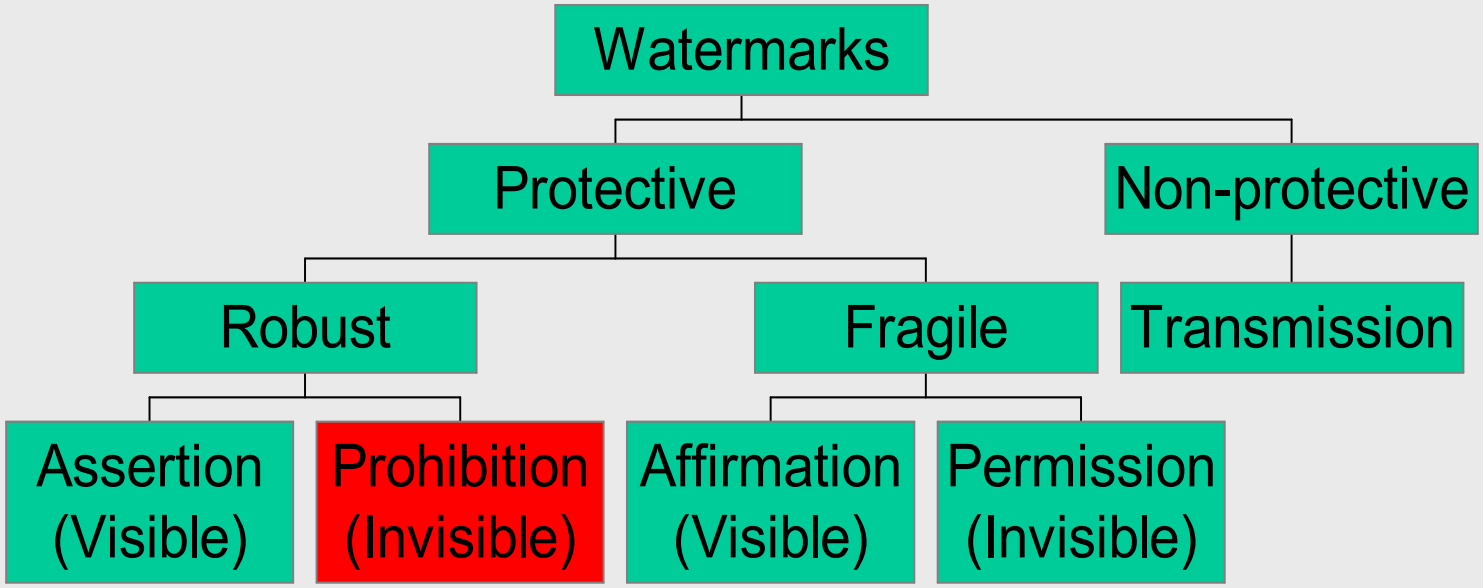
Why Watermark Software?
(Thomborson & Nagra, 2002)

- Invisible robust watermarks: useful for prohibition (of unlicensed use)
- Invisible fragile watermarks: useful for permission (of licensed uses).
- Visible robust watermarks: useful for assertion (of copyright or authorship).
- Visible fragile watermarks: useful for affirmation (of authenticity or validity).

A Fifth Function?

- Any watermark is useful for the transmission of information irrelevant to security (espionage, humour, …).
- Transmission Marks may involve security for other systems, in which case they can be categorised as Permissions, Prohibitions, etc.

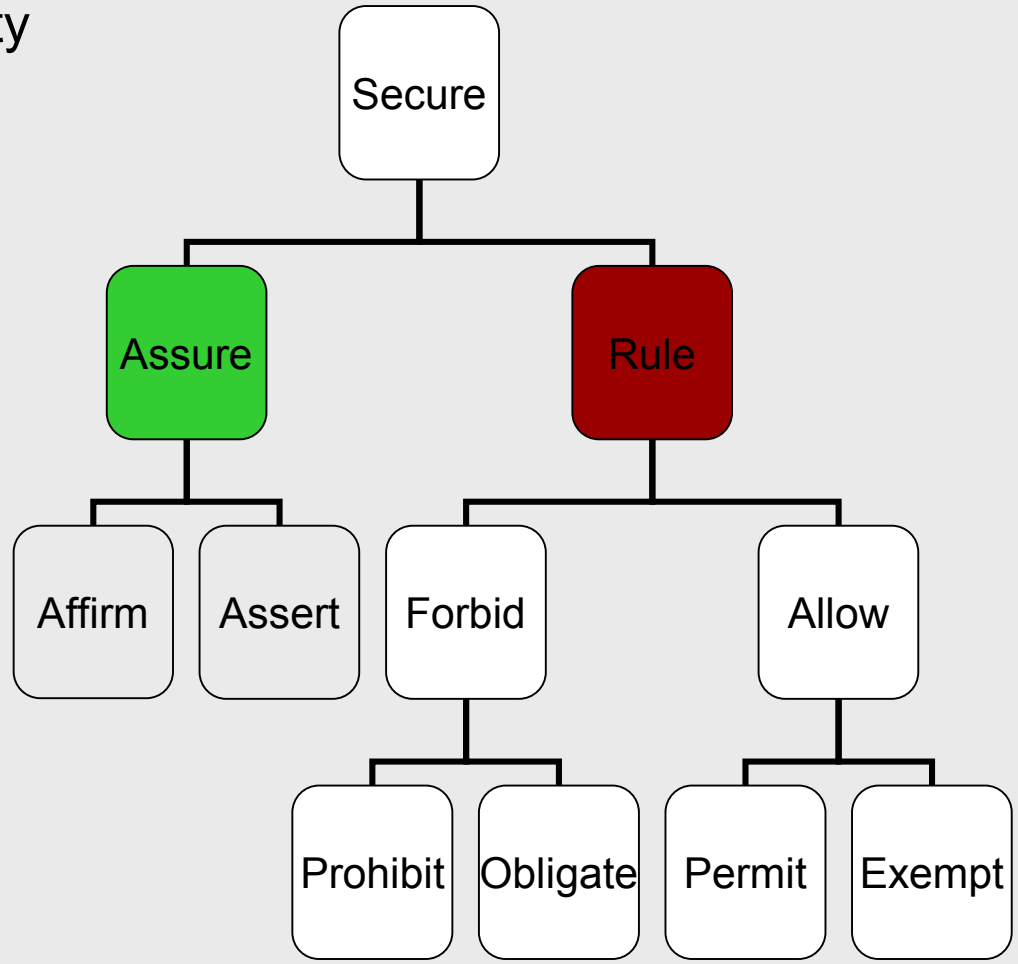
Our Functional Taxonomy for Watermarks [2002]



But: there are no "assertions" and "affirmations" in our theory of static security! Hmmm....

Future and Past Actions

- The Rules of static security define what a system should do in the future.
- Assertions (e.g. of authorship) are Assurances about a past action.
- Affirmations (e.g. of authenticity) are Assurances about a past inaction.
- Audit records are Assertions.
- Identifications and Authentications are Affirmations.
- Maybe we *can* clean up the second layer in my security taxonomy!



Summary/Review

1. What is security?
   - Three types: static, dynamic, governance.
   - Secure subsystems must have governors.
2. What is software watermarking, and how is it used?
   - We have identified five types of watermarks.
   - Invisible & robust watermarks have attracted the most interest to date.
3. Research question #3: Are we missing any cases?
   - Assertions and affirmations should be analysed carefully... if implemented as watermarks they'd be visible & robust, but why should we have a covertext?
   - Are there different types of covertexts?