# *Software protection*

Bart De Win & Nessim Kisserli

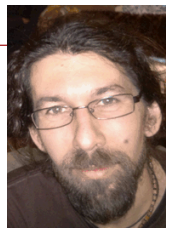
SecAppDev 2015

pwc

## *Who are we ?*

### Bart De Win

- 15+ years of Information Security Experience
  - Ph.D. in Computer Science - Application Security
- Author of >60 scientific publications
- ISC² CSSLP certified
- Senior Manager @ PwC Belgium:
  - Expertise Center Leader Software Assurance
  - (Web) Application tester (pentesting, arch. review, code review, ...)
  - Trainer for several courses related to secure software
  - Specialized in Secure Software Development Lifecycle (SDLC)
- OWASP OpenSAMM co-leader
- Contact me at bart.de.win@be.pwc.com



### Nessim Kisserli

- 15 years of Information Security Experience
  - Msc. in Information Security
- UNIX System Administrator
- Former researcher at KULeuven

- Senior Consultant @ PwC Belgium:
  - Application tester (pentesting, code review, ...)
  - SDLC

- Contact me at nessim.kisserli@be.pwc.com

# *Agenda*

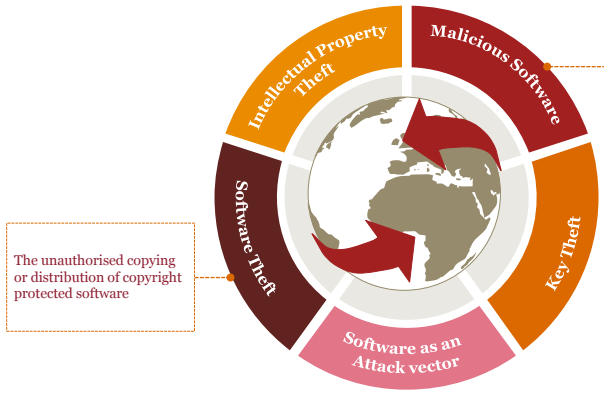
1. **Setting The Scene**
2. Software Protection Controls
3. Discussion



## *Setting the scope*

Once software is written,
what can the **manufacturer** do
*within the same software*
to protect it against abuse ?

## *Why do we need protection?*




Our software can be modified to include malicious code

The unauthorised copying or distribution of copyright protected software

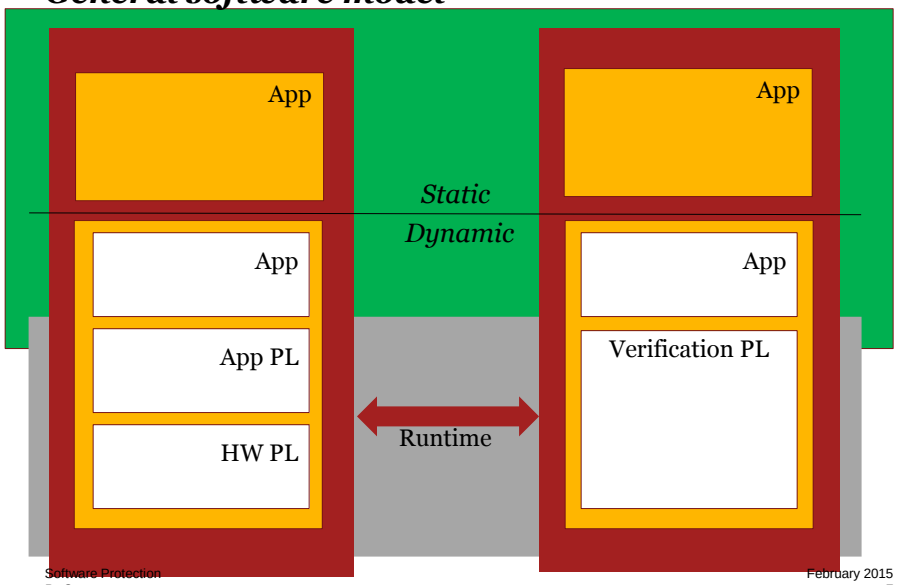


## *Real world examples*

Reverse engineering software to steal a proprietary image compression function.

Modifying mobile banking apps and redistributing them, introducing backdoors and illegally redistributing.

## General software model

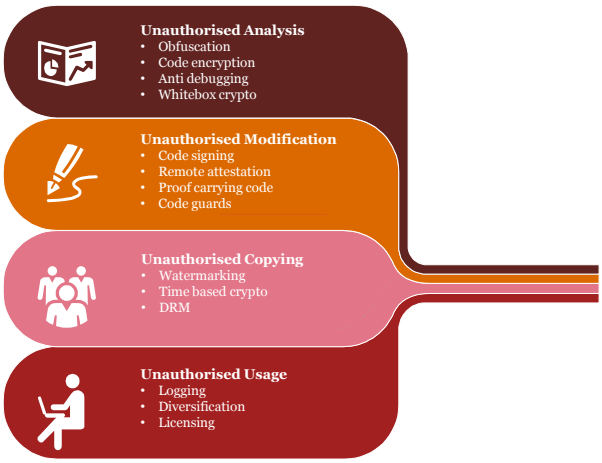





## Is it useful ?

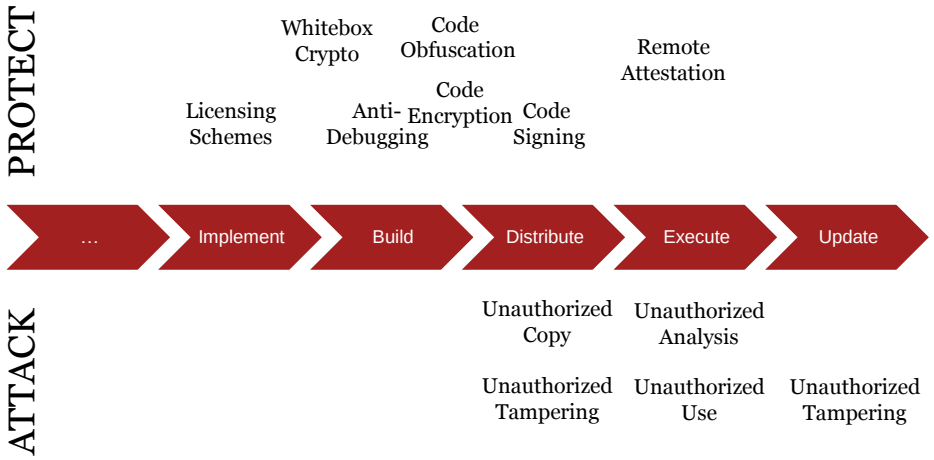
- Fundamentally impossible to protect software if you can't trust the execution platform (e.g., using a TPM)
- Still, software-only protection does make sense
  - If well scoped & targeted
  - To *reasonably* prolongue the time to break the protection

## *Threats and controls*
*A general overview*

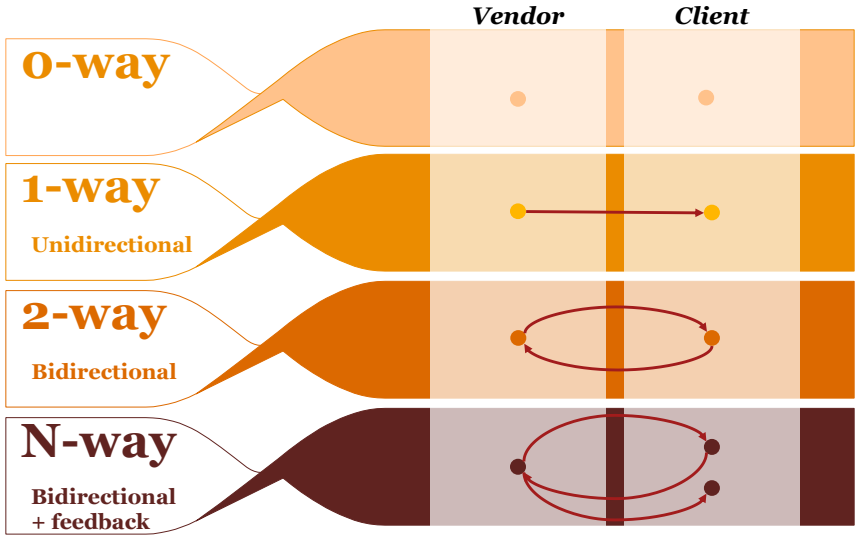





## *Software Protection Timeline*



## *Software protection models*






## *Agenda*

1. Setting The Scene
2. **Software Protection Controls**
   - **Unauthorized Analysis**
   - Unauthorized Modification
   - Unauthorized Copy
   - Unauthorized Use
3. Discussion

## *Obfuscation*

| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Build |
| *Commercial availability* | Yes |
| *Technology specificity* | Language-dependent |
| *Technical complexity* | Low-Medium |
| *Implementation cost* | Low |

What is it?

- Obfuscation is the process of making “source” code difficult for humans and/or machines to understand
  - For scripting languages (In the web context)
  - For Bytecode (Java, .Net CLR)
  - For Binaries (C, C++, ASM)

How is it applied?

- Modifying the “source” with semantic preserving transformations
- Applied in the last phase of the software build process



## *Obfuscation – Techniques*

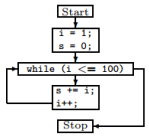
Typical techniques are the following:

- Name obfuscation: At source code level, change class and function names
- String encryption: encrypt the strings in the .data section
- Control flow obfuscation (control-flow flattening) : Break the structure of the CFG.
- Code virtualization: Translate the code into virtual opcodes that can be understood by a secure virtual machine.

## *Obfuscation – Control Flow Flattening*

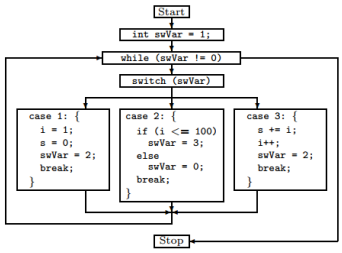
original

```
i = 1;
s = 0;

while (i <= 100) {


  s += i;
  i++;

}
```




control-flow flattening applied

```
int swVar = 1;
while (swVar != 0) {
  switch (swVar) {
    case 1: {
      i = 1;
      s = 0;
      swVar = 2;
      break;
    }
    case 2: {
      if (i <= 100)
        swVar = 3;
      else
        swVar = 0;
      break;
    }
    case 3: {
      s += i;
      i++;
      swVar = 2;
      break;
    }
  }
}
```






## *Obfuscation - Products*

Many commercial and open-source products available, e.g.:

- ProGuard/DexGuard
- Arxan
- Babel
- Irdeto (Cloakware)
- DashO
- …

Differ in supported techniques languages, and ease of configuration

## *Obfuscation - Discussion*

- Obfuscation can make it harder to reverse engineer the software and try to understand what it does (also for reflection), but it does not make it impossible.
- One important requirement is that the resulting code (after obfuscation) should still be executable without any reversing transformations.
- Obfuscation is sometimes used to hide malicious code, thereby also impact for anti-virus products
- Code obfuscation can be combined with certain licensing schemes. For example, the product key can be used to derive a secret key used to de-obfuscate the transformed code



## *Code Encryption*

| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Compilation |
| *Commercial availability* | Yes+ |
| *Technology specificity* | Independent |
| *Technical complexity* | Low |
| *Implementation cost* | Low |

What is it?

- It encrypts the binary code, and only decrypts the code when it is needed.

How is it applied?

- Mostly part of a bigger protection scheme (it comes as a feature of a software protector)

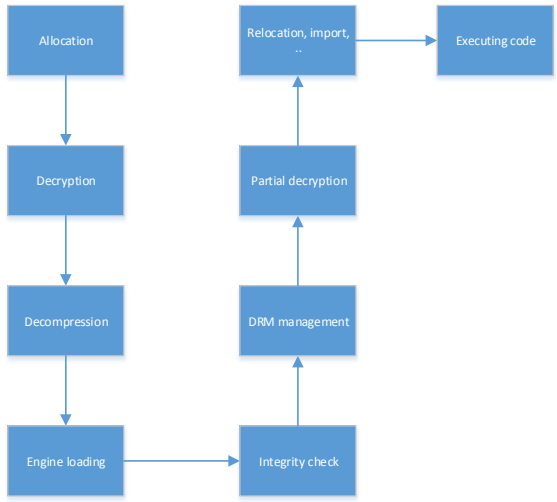
## *Code Encryption - Techniques*

- Software packer: Compresses the code and packs it into a binary. Decompressing and recreating the original file is done at runtime. The attacker can still obtain a memory dump

- Software cryptor: Performs code mutation, transforming them to something which can be executed.



## *Code Encryption – Overview*



## *Code Encryption – Discussion*

- Code encryption works, but it will only slow down a determined attacker
- The resulting code is dependent on some sort of "loading stub" to decrypt certain sections of the code.
- Can be more than simply encrypting binary sections, it can also deter standard attacks like trying to trace system calls, or library calls or even try to encrypt the process memory



## *Anti-debugging*

| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Distribution |
| *Commercial availability* | Yes |
| *Technology specificity* | Independent |
| *Technical complexity* | Low-Medium |
| *Implementation cost* | Low |

What is it?

- Software-level techniques used to "fool" the algorithms used by debuggers.

How is it applied?

There are two common ways:

- First, fool linear sweep, and recursive descent disassemblers
- Second, introduce dynamic at run time behaviour.

## *Anti-debugging – Example 1*

- Example 1: Self debugging

The windows API allows for programmers to connect to the app with a debugger, an example of such a call is the "DbgUIConnectToDbg"

A initial loader programme can create a new process with the DEBUG_PROCESS flag in the CreateProcess function.

Because the programme will already be "debugged", an additional debugger will not be able to attach to the programme.



## *Anti-debugging – Example 2*

- Example 2: Thread hiding

The windows anti debugging API allows the programmer to create certain classes with the *NtSetInformationThread* field set.

A debugger would not receive any events of a thread with the HideThreadFromDebugger class called on.

```
If (hThread == NULL)
      status = NtSIT(GetCurrentThread(),
                     0x11, // HideThreadFromDebugger
                     0,0);
```

## *Anti-debugging – Discussion*

- An arms race between attackers and defenders.
- Self contained “security”, no trusted third party required

- Some software protector solutions provide anti-debugging techniques that can be applied when building the software. Some protectors include anti patching heuristics.

- Most common techniques attempt to detect breakpoints on instructions or memory access, or try to protect against dumping certain memory regions.



## *White-box Cryptography (WBC)*

What is it?

- Allows a specific cipher and key combination* to be used for encrypting and decrypting data, in a hostile environment, open to analysis and tampering by an attacker without leaking the key.

How is it applied?

- The key, cipher and random data are merged and transformed into a complex series of lookup tables used in a programme. When executed, it encrypts or decrypts its input producing the same result as the black-box-equivalent cipher and key.


| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Implementation |
| *Commercial availability* | Yes |
| *Technology specificity* | Independent |
| *Technical complexity* | Medium-High |
| *Implementation cost* | Low-Medium |

## *WBC*

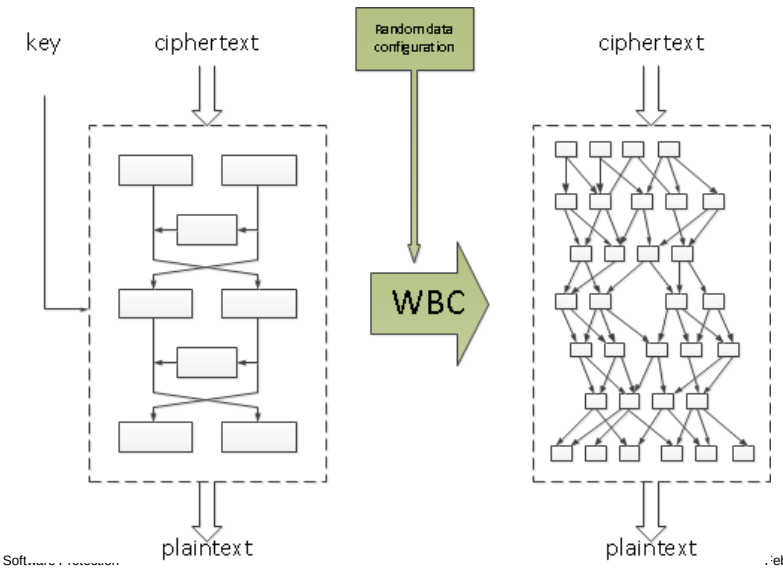
White-box cryptography is a solution to the white-box (or malicious host) attack model. Namely:

- Attacker has full access to the software implementation of a cryptographic algorithm
- Attacker has full control over the execution platform (CPU state, memory and register details, etc.)

The implementation must be its own protection.



## *WBC – Overview*



## *WBC – How it works*

We capture the result of key dependent operations into a lookup table and store them into the binary.

Next, the lookup data flow is randomized (random –input and output bijective encoding) – the resulting algorithm appears as the composition of a series of lookups on random values.

As a final step, some key independent encodings may be used to prevent code lifting



## *WBC – Advantages*

WBC does not depend on hardware modules. This means:

- Faster “manufacturing”
- Cheaper “manufacturing” – no need for certified factories
- Lower cost of production
- Better platform compatibility*

Implementations, even with the same key, are naturally diversified (built-in watermarking).+

By only providing an encryption call, vendors can “convert” a symmetric cipher into an asymmetric one!

## *WBC - Disadvantages*

- Size: The size of a white-box implementation is much larger than the equivalent black box implementation (e.g. 188 times)++
- Speed: White-box implementations are generally slower than black box techniques (e.g. 55 times slow-down)+
- They are fixed-key solutions: Dynamic key implementations will probably weaken security.
  - All current AES white-box implementations available in academic literature have been broken.
  - Commercial companies don't publish their implementations..



## *WBC – commercial solutions*

Used by many companies (mainly DRM):

- Microsoft
- Apple+
- Sony
- NAGRA
- Netflix..

WBC solutions provided by many companies:

- Arxan
- Philips
- whiteCryption (DES, AES, ECC, SHA, 3DES, RSA, ECDSA, DH, etc.)
- Irdeto (Cloakware)
- SafeNet
- Inside Secure (Metaforic) (AES, RSA, ECDSA)

## *WBC - commercial solutions*

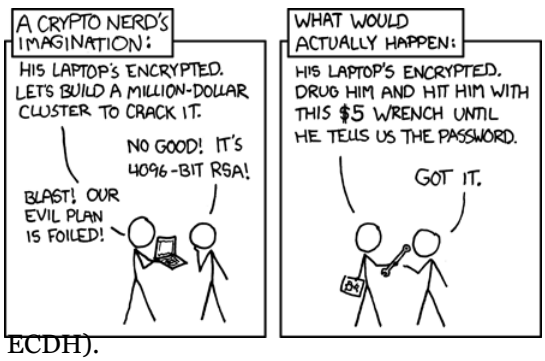
Inside Secure (Metaforic) White-box

- Provide AES, RSA, ECDSA white-box implementations for iOS, Android BB10, Linux, Windows, OSX.
- Use-case: protect sensitive data / credentials in Enterprise BYOD solutions by using white-box crypto.

whiteCryption's Secure Key Box (SKB) provides many whitebox implementations of ciphers, signature verification, key digest, and key agreement algorithms.



## *WBC – Discussion*




Exists for symmetric (DES, 3DES, AES) and asymmetric (RSA, ECC) algorithms, signatures (ECDSA) and key exchange protocols (DH, ECDH).

In general, even the best crypto is insufficient in practice:

No attacks have been observed to-date on commercial white-box crypto implementations. Attackers always choose a weaker vector. White-box crypto (adherence to Kerckhoff's principle) is, in a sense, the opposite of obfuscation (security through obscurity)

# Agenda

1. Setting The Scene
2. **Software Protection Controls**
   - Unauthorized Analysis
   - **Unauthorized Modification**
   - Unauthorized Copy
   - Unauthorized Use
3. Discussion



# Code Signing

| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Deployment |
| *Commercial availability* | Yes |
| *Technology specificity* | Independent |
| *Technical complexity* | Low |
| *Implementation cost* | Low |

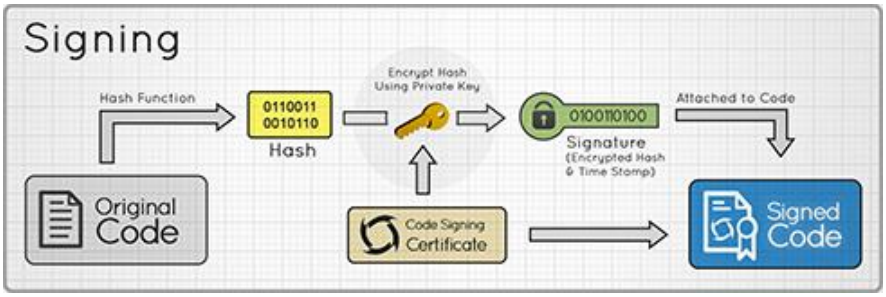
What is it?

- The process of digitally signing executables to confirm the software author and guarantee that the code has not been altered, making an assertion about the binary.

How is it applied

- Software is signed with a private key and distributed with the corresponding commercial software publishing certificate.
- Certificate requestors must first “prove” their identity.

## *Code Signing - Overview*






## *Code signing on mobile*

New execution environments allow for new code signing enforcement

- Android

By default, android requires that all apps are signed before they can be installed. The platform asserts the code itself + it asserts the permissions

- iOS

Apple requires that all iOS apps are signed by a certificate issued by Apple to a trusted developer. After the app verification process by apple, the app is re-signed before it can run an iOS device

## *Code Signing – Discussion*

- PKI is a trusted, well proven technology that is used every day to ensure secure communication, it's a natural extension to apply this is on software authentication
- Most of the platforms provide hooks for enforcement, and more of them will start to demand it
- The host is responsible for ensuring the software package is signed by a trusted authority
- Both Windows and Mac support signed applications



## *Code Guards*

| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Build |
| *Commercial availability* | Yes |
| *Technology specificity* | Independent |
| *Technical complexity* | Low-Medium |
| *Implementation cost* | Low-Medium |

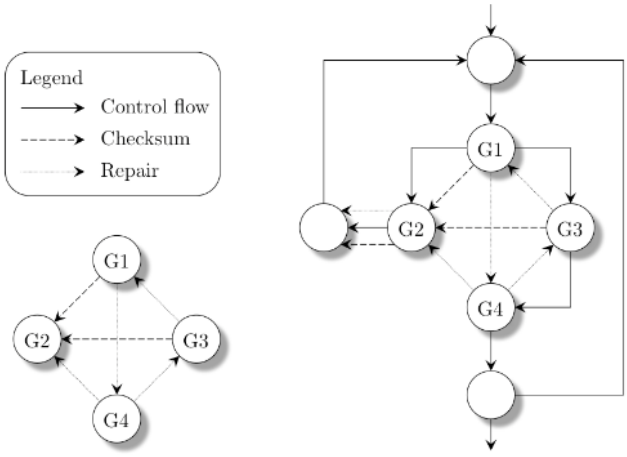
What is it?

- Small pieces of code which verify the integrity of an application's execution, and possibly each other's. Designed to detect unauthorized software modification.
- May optionally restore certain corrupted values.

How is it applied?

- During development, critical regions are identified for protection. In toolchain-assisted solutions, source code hints enable the compiler to build a protection profile for later instrumentation, generally at link-time.

## *Code Guards – Overview*






## *Code Guards – Practical application*

- In full COTS solutions, guards inserted without source code using binary rewriting.

- As guards can be identified, further techniques are used to maximize their effectiveness including:
  - Embedding large numbers of guards, including redundant ones
  - Delaying guard effects from time-of-check
  - Using anti-debugger, anti-analysis tricks
  - Obfuscation
  - Running guards in a separate process (VM/Hypervisor)

## *Code Guards - Commercial use*

- Arxan first commercialized the concept in 2001, with support for linux, windows, OS X, Android, IOS, Java and .NET.
  - Checksum (code guards)
  - Self-repair (repair guards)
    - Requires data/code redundancy in the binary.



## *Code Guards: Commercial Use 2*

- Microsoft integrated code guards into a proof of concept, comprehensive protection solution (XFI) on windows.
- Applied to legacy code (uses binary rewriting) .
- Protected image rendering programme which detected a malicious JPEG (malware) and aborted rendering.
- Code guards used to check Control Flow Integrity (CFI) by:
  - Checking transfers of control against whitelisted addresses
  - Checking integrity of register and stack values (against shadow copies), etc.

## *Code Guards – Discussion*

Disadvantages to code guards include:

- Increased software complexity, code size and runtime overhead
- Incompatibility with self-modifying code
- "Brittle" and risk "invalidation" by compiler: Generally inserted via binary rewriting.
- Developed and extensively studied in academic literature where numerous schemes exist.
- Generally deployed within commercial products as part of a wider, multi-layered protection mechanism.



## *Proof Carrying Code (PCC)*

| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Implementation |
| *Commercial availability* | Mostly academic |
| *Technology specificity* | Independent |
| *Technical complexity* | High+ |
| *Implementation cost* | High+ |

What is it?

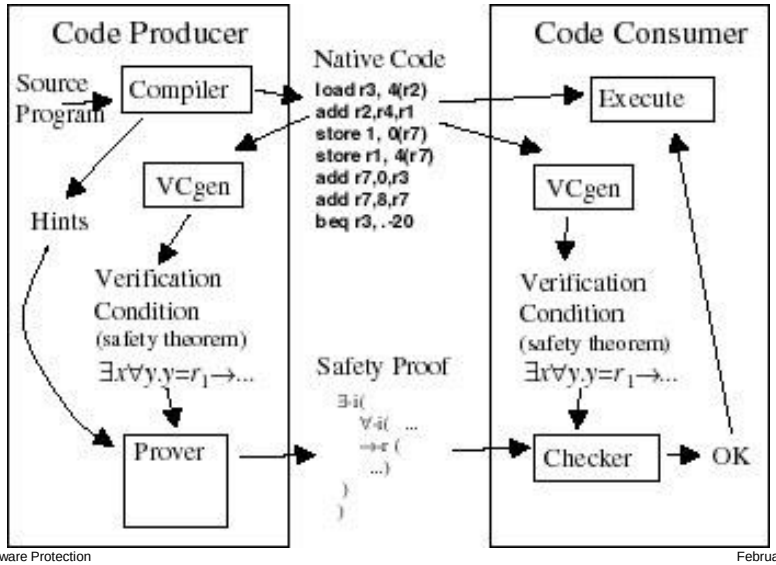
- A framework which allows untrusted code to be proved "safe" to execute.

PCC protects clients by guaranteeing certain properties during execution of otherwise untrusted code.

How is it applied?

- The producer (software vendor) mechanically proves certain **safety** properties about the code. A consumer (user/client) who verifies the correctness of the proof using a **checker**, is guaranteed claimed safety

## *PCC – Overview*






## *PCC – Techniques*

- Necula's original PCC. Axioms strongly tied to fixed Type System. Does not guarantee programme will only execute what its specification states and nothing else.
- Appel's Foundational PCC. No fixed Type System. Can guarantee adherence to specification and nothing else.

By safety we mean type and memory safety: private data stays inaccessible, important data cannot be overwritten, limit consumed resources, etc.

## *PCC – Discussion*

- We want the host to be able to verify properties about the application in some formal way.
- Strength of PCC: Requires a small TCB (proof verifier).
- It has been applied to ensure the JVM's JIT preserves type-safety on the resulting native code (Java only type-checks the bytecode, NOT native code).
- It still is an academic technique and is not formally applied in commercial software solutions.
- Does not guarantee the code has not been tampered with! Only that it still does not violate the safety policy.



## *Remote Attestation*

| | |
|---|---|
| *Network interaction* | 2-way |
| *SDLC stage* | Execute |
| *Commercial availability* | Yes |
| *Technology specificity* | Independent |
| *Technical complexity* | Medium |
| *Implementation cost* | Medium |

What is it?

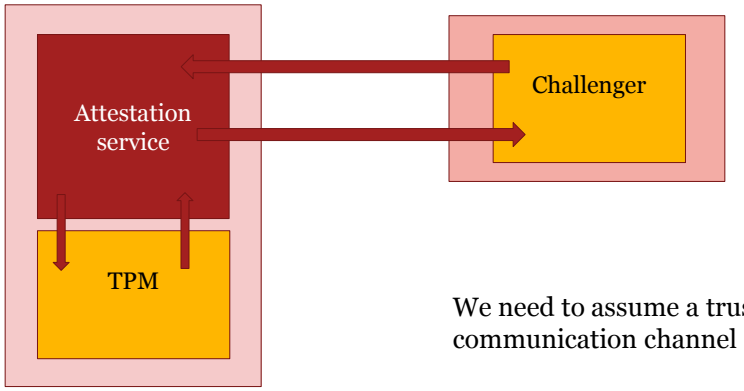
- Remote attestation is a method by which a host (client) authenticates it's hardware and software.
  The primary goal is to determine the level of trust, secondary goal is to detected unauthorized changes to software

How is it applied?

- It is part of the TCG standard, listed as one of the key features.

## *Remote attestation – Overview*

It is a two party system, requiring all the parties to be online when the protocol is being executed.




We need to assume a trusted communication channel



## *Remote attestation – The details*

Step 1: a challenger obtains a certificate from a trusted CA that claims you have a valid TPM on your machine

Step 2: The challenger sends a request to the computer to attest to its software state

Step 3: Your machine sends back a list of cryptographic hashes stored in the TPM (called PCR) and their state

Step 4: The challenger checks to see if the current state of the machine is a valid state. Depending on that check it determines to proceed

## *Remote Attestation – Discussion*

- Part of the TGC standard
- The architecture consists of two major components: Integrity measurement architecture and remote attestation protocol.
- Any remote attestation scheme relies on some sort of trusted hardware component.
- Windows 8.1 includes a remote attestation service
- OpenStack includes the OpenAttestation project (OAT), remote attestation services.



## *Agenda*

1. Setting The Scene
2. **Software Protection Controls**
   - Unauthorized Analysis
   - Unauthorized Modification
   - **Unauthorized Copy**
   - Unauthorized Use
3. Discussion

## *Watermarking & Fingerprinting*

What is it?

- A technique for embedding a unique fingerprint in each software copy (or set of copies) to identify the originator of unauthorized software disclosure (traitor tracing)

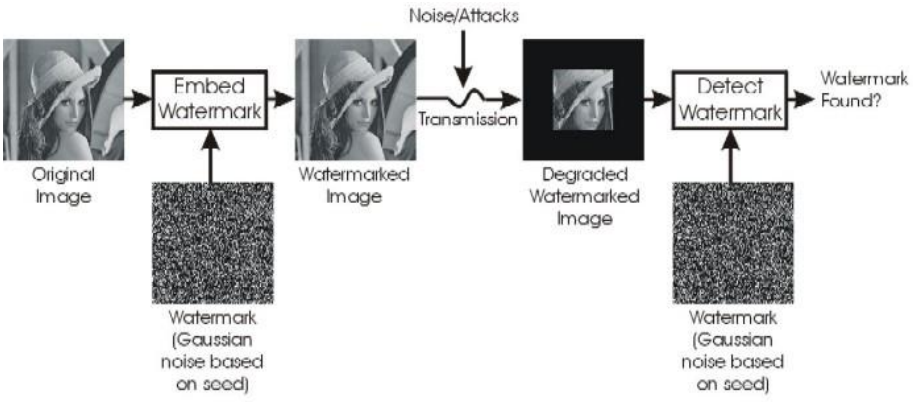
How is it applied?

- It's a way of creating an identifier form the application itself (relying on existing program attributes)

| | |
|---|---|
| *Network interaction* | O-way |
| *SDLC stage* | Various* |
| *Commercial availability* | Yes+ |
| *Technology specificity* | Independent |
| *Technical complexity* | Low-Medium |
| *Implementation cost* | Low-Medium |



## *Watermarking – Overview*



## *Watermarking & Fingerprinting – Distinctions for Software*

- Watermarking: Embedding an artefact in a programme
  - Perceptible or imperceptible
  - Generic or uniquely identifying
  - Keyed or unkeyed.

- Fingerprinting: Extracting an *identifying* watermark from a programme
- Ideal fingerprint system:
  - Minimal size cost and maximum stealth and resilience.
  - In practice, a tradeoff.



## *Watermarking & Fingerprinting – Techniques*

**Static** Watermarks:

- Data: strings in the code
- Code: Order of specific instructions, basic blocks, or procedures.

**Dynamic**: Given a particular sequence of inputs

- Easter Eggs: Programme enters a particular state (or produces output)
- Trace: Monitoring instructions executed or addresses used
- Data Structures: State of programme variables after executing the input

## *Watermarking & Fingerprinting – Example*

Idea: Embed code in the programme which builds a watermark.

- Extracted by knowledge of "secret key"*

Embedding:

- Fingerprint is embedded into a graph topology G which is split into several components $G_1$, $G_2$...$G_n$.
  - Each component $G_i$ converted into bytecode which builds it ($C_i$)
  - Bytecode embedded along the execution path taken given the secret key as input ($I_0$, $I_1$, $I_2$...)

Extraction:

- Run with secret input
- Fingerprint graph built on the heap is extracted and identified



## *Watermarking & Fingerprinting – Overview (CT)*

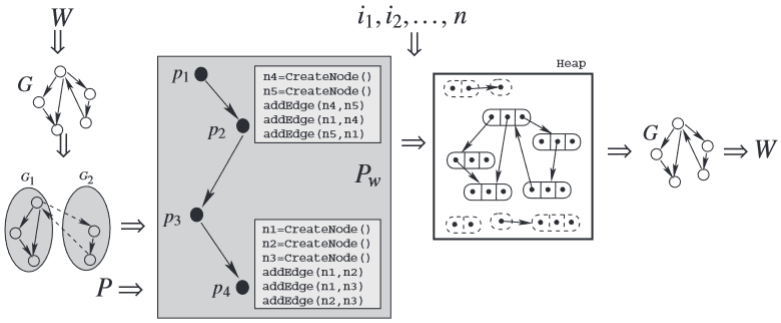



Fig. 3. Overview of the CT algorithm.

## *Watermarking & Fingerprinting – Graphs*

- Graphical Enumerations are used to transform a watermark integer into the nth enumeration of a graph.
- Many possible graph families (Directed Parent-Pointer Trees, Planted Planar Cubic Trees, etc.)

Permutation Encoding

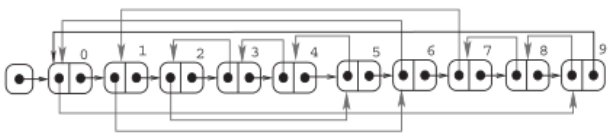
- Method of transforming an integer fingerprint into a permutation. E.g. 180398 becomes $\pi$ = ‹9,6,5,2,3,4,0,1,7,8›



## *Watermarking & Fingerprinting – Graphs (2)*

- Permutation encoded using a Permutation Graph (Singly linked circular list). Each element in list has 2 pointers (data and list).
  - list[i].dataPtr = π[i].
  - list[i].listPtr = (i+1) mod n

Example: Permutation Graph encoding an integer.




In practice: Limited stealthiness.

## *Watermarking & Fingerprinting – products*

Google Content ID (fingerprinting)*

- ProMedia Carbon, a universal transcoding solution used by many media companies generates a unique content-ID during media production.
- Media owners upload the ID with a "Usage Policy" describing how to handle matched content.
- Offending uploads to youtube are either blocked or enrolled in Ad-revenue generation schemes.

- Gracenote+ (Tribune) (fingerprinting)



## *Watermarking & Fingerprinting– Discussion*

- Used widely, in part due to its relative lightweight, non-intrusiveness and partly due to its unique value.
- Currently used more for media (content) than software. Software is often linked to a fingerprint of the execution environment (Hardware serial numbers, Mac address)*
- Closely related to steganography
- Must be robust in the face of transformations (e.g. obfuscations, change resolution, etc.).

## *DRM – Overview*

| | |
|---|---|
| *Network interaction* | 0- or 2-way |
| *SDLC stage* | After deployment |
| *Commercial availability* | Yes |
| *Technology specificity* | Independent |
| *Technical complexity* | Medium |
| *Implementation cost* | Medium |

What is it?

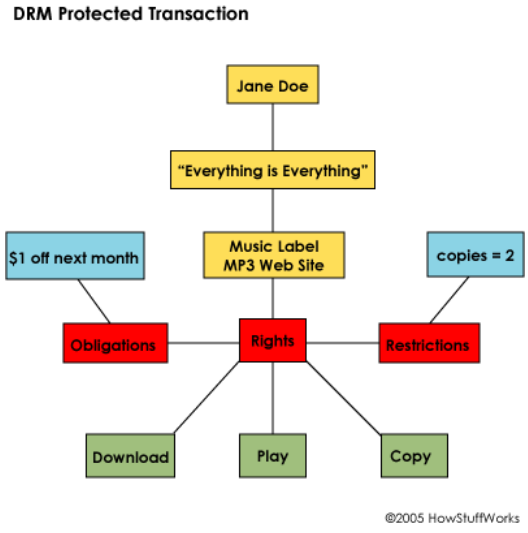
- Digital Rights Management (DRM) is a class of technologies used to enforce copyright over digital content after distribution. In essence it's a set of access controls (rights can vary per user).

How is it applied?

- By including tags in the content or using some form of data encryption
- By enforcing strict licensing



## *DRM – Overview*



## *DRM – Classical examples*

- Use persistent online authentication
- Make software unusable as soon as an illegal copy is detected
- Require some derivative of the product key to decrypt digital content
- Limit the number of installations
  - Bind a total of installations to a product key and verify this online



## *DRM – Real world example: FairPlay*

DRM technology introduced by Apple to protect their multimedia content

- Fairplay-protected files are regular mp4 containers with an encrypted AES AAC audio stream. The master key to decrypt the audio stream is included with the protected file, but the key itself is also encrypted with a 'user key' (unique per user).

- Keys are stored with the users' information on Apple's servers. Only authorized iTunes can obtain the 'user key' and play the songs (online). Each iPod has his own encrypted key storage to hold those 'user keys' (offline).

## *DRM – Real world example: Netflix (1)*

Netflix and EME (Encrypted Media Extension - W3C)

- Provides a specification for communicating between a web browser and a DRM agent, and allows for playing back DRM-wrapped digital content
- MPEG-DASH and MPEG-CENC (Common encryption) in the MPEG standard make it possible to play back protected (encrypted) content. Both of these are included in the HTML5 EME standard



## *DRM – Real world example: Netflix (2)*

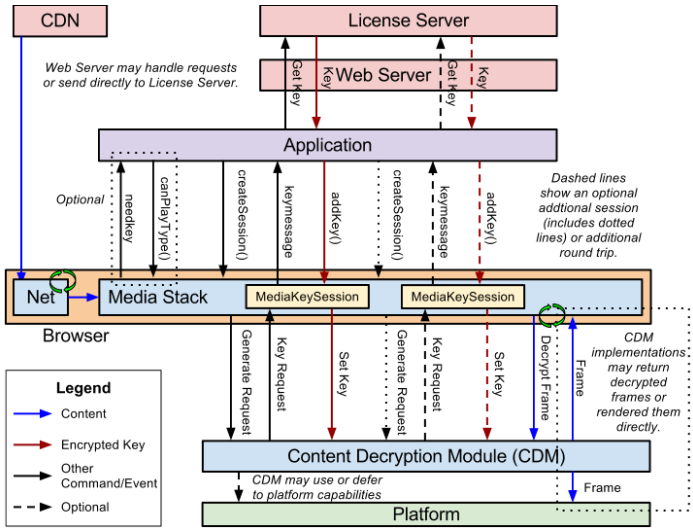
What does it include?

- It provides simple *'clear key decryption' to 'complex license key exchanges'*
- An API, no "full blown" DRM solution.

Model is the following components:

- Key System: A Content production DRM mechanism
- Content Decryption Module: A client side component to play back encrypted content
- License server: Interacts with CDM to provide decryption keys
- Packaging service: Encodes and encrypts media for end user.

## *DRM – Real world example: Netflix (3)*






## *DRM – Discussion*

- Collection of techniques used to accomplish content management
- Currently mostly a legal tool. Implementations are not fully resistant to bypass.
- DRM as a concept is technology independent
- Different targets need different DRM solutions (web video vs music streaming)

## *Timed-Release Encryption (TRE)*

What is it?

- A technique of encrypting content and publishing it such that it can only be decrypted at a specified later date. Can be used to ensure publicly disclosed (encrypted) votes are not "opened" until a predetermined date, bids in an auction, etc.

How is it applied?

- No definitive way of applying it. Keys -and possibly accompanying artifacts- are created and used as normal to encrypt the data to-be released. When present, the artifacts are also released to enable decryption at the desired time.

| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Distribution+ |
| *Commercial availability* | Mostly academic |
| *Technology specificity* | Independent |
| *Technical complexity* | Low- |
| *Implementation cost* | Low |



## *TRE - (Rivest time-lock puzzle)*

- Rivest, Shamir, and Wagner created and published a challenge in 1999 which should only be decryptable in 2033. Relies on computing

  $2^{(2^t)} \pmod n$

Solved by computing t successive squarings modulo n, a non-parallelizable calculation.

Values of n and t chosen assuming Moore's law will produce sufficiently powerful chips to complete the calculations by the desired decryption date*.

## *TRE – Discussion*

- Still academic for now.* HP Labs in Bristol created Time Vault a service for timed release of confidential information+
  - Practical implementations require TTP
  - Cleverly designed puzzles exploit Moore's law to rule out TTP. Probably low accuracy. More novelty for now.
- Useful real-world applications (elections, etc.)
- Identity-Based Encryption (IBE) schemes have become the basis for all proposed Time-release schemes.
- The puzzle and private keys can be generated independently of development.



## *Agenda*

1. Setting The Scene
2. **Software Protection Controls**
   - Unauthorized Analysis
   - Unauthorized Modification
   - Unauthorized Copy
   - **Unauthorized Use**
3. Discussion

## Licensing

| | |
|---|---|
| *Network interaction* | 2-Way |
| *SDLC stage* | Implementation |
| *Commercial availability* | Yes |
| *Technology specificity* | Independent |
| *Technical complexity* | Low |
| *Implementation cost* | Low-Medium |

What is it?

- A "software license" is a concept to govern the use and redistribution of software. It grants the user certain legal rights to use the software

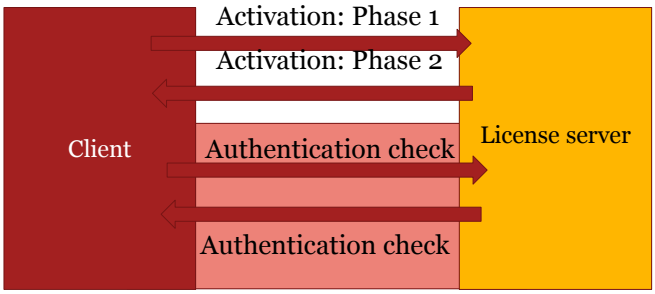
How is it applied?

- The translation results mostly in some form of *usage protection*. This in turn is then translated into a "license key scheme", that requires the user to verify his or her installation.
- *A common way is to use persistent online authentication.*

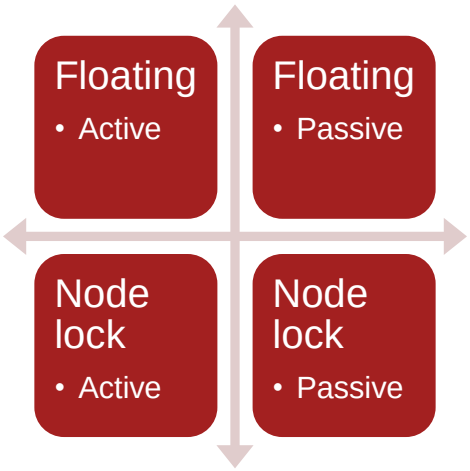


## Licensing - Overview

- The general concept.
  - Enter a product key on installation.
  - Verify the product key and perform "activation"
  - Create key file



## *Different licensing schemes.*






## *Licensing - Applications*

- Dongle
  - Hardware key containing serial number required for the software to run
- Product activation
  - Requiring the user to verify the license by entering a product key
  - Binding software to execution environment (via fingerprinting)
  - Restricting the number of times an application can be run
- Keyfile
  - A file with the activation key that is needed to run the software

## *Licensing – Discussion*

For licensing to be effective, it must be inherent to the functional flow of the application. If not, the license check could be patched out.

Thinking about licensing should be done early in the SDLC



## *Diversification*

| | |
|---|---|
| *Network interaction* | 0-way |
| *SDLC stage* | Build |
| *Commercial availability* | Yes |
| *Technology specificity* | Independent |
| *Technical complexity* | Medium |
| *Implementation cost* | Medium |

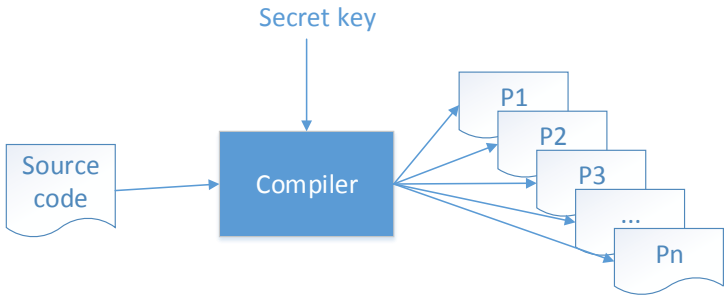
What is it?

- Transformation techniques to generate functionally identical yet distinct binary instances from source code. It offers probabilistic protection against Break Once Run Everywhere (BORE) attacks.

How is it applied?

- Transformations are applied at the source code, normally via a seeded diversifying compiler.

## *Diversification - Overview*






## *Diversification – Techniques*

Transformations include:

- Layout randomizations: Basic blocks are reordered
- Control Flow Flattening
- Opaque predicates
- Branch functions

## *Diversification – Example: Opaque predicates*

An opaque predicate is an expression used in a conditional clause but which always evaluates to the same a-priori known value, e.g.

```
x=5;
if (x % 2) > 9 {
    // never true this code will never  be executed. Good place
to add watermarking code/data.
} else {
    // always true
}
```

Trivial opaque predicates may be optimized away by the compiler or a sophisticated attacker's reverse engineering framework (including abstract interpretation)



## *Diversification – Example: Opaque predicates*

More realistic opaque predicate. Given:

- Distinct primes, P and Q;
- Two arbitrary, distinct positive numbers n, m;
- Any two variables x and y from the source code;

The following expression will always evaluate to false:

```
P * ((n| x)**2) != Q * ((m | y)**2)
```

## *Diversification – Challenges*

Diversification poses 2 main problems to software vendors:

- Software updates: Partial (delta) updates must be tailored to each instance. Full updates preferable.
- Bug reporting: Bug reports must be processed (normalized) before they can be used.



## *Patch diversification – case study*

Problem:

- Crackers* compare (binary diff) pre-patch and post-patch versions of binaries (patch Tuesday) to determine details of the patched vulnerability allowing them to develop exploit code (exploit Wednesday) to use on still unpatched systems.
- Crackers use the same collusion attack to port their cheats or cracks from older versions of games to newer patched versions.

## *Patch diversification – diversifying transforms*

Pool of diversifying transformations:

- Code layout randomization
- Partial control flow flattening
- Conditional branch flipping
- Two-way Opaque Predicate insertion



## *Patch diversification - Approach*

Starting with the patched programme, $V_1$, 18 diversifying iterations were run.

- Run BinDiff on $V_i$ and $V_{i-1}$
- Unmatched procedure: Keep same strategy (BinDiff thwarted)
- Matched procedure with different signature: Extend strategy to new signature
- Matched procedure with same signature: Try different strategy for same signature (BinDiff not thwarted)

## *Patch diversification – diversification strategy*

Selection of diversifying transforms is determined by a rules table. For example:

- Conditional branches may be flipped, in any iteration, in any procedure matched by BinDiff's "Hash Matching"* signature.
- After the first iteration, two-way opaque predicates may be inserted in or before basic blocks which are not executed (cold code) in any procedure matched by BinDiff's "Edges Flowgraph" signature.
- Log each procedure's diversification strategy



## *Patch diversification – diversifying transforms*

- Branch function and call function insertion*: Direct control transfers (jump or fall-through) are rewritten as calls to a branch function with a parameter which allows it to transfer control to the correct location.

## *Patch diversification – Results*

- No diversification: 99% match unchanged procedures.
- After 18 iterations, BinDiff matched fewer than 3% of the code. Code-size overhead is 30-40%.
- Execution overhead below 5% until iteration 16, then 115%.

- Thwarting diffing tools is possible with acceptably low execution overhead but large increase in patch size.
- Diversification could be identified with code normalization tools. Part of the constant arms-race escalation in software protection.



## *Diversification - Discussion*

- Performed by a diversifying compiler.
- May leverage programmer annotations or compiler flags only.
- Still mainly academic (including more “industry-minded” Microsoft Research) though commercial products exist (e.g. whiteCryption’s Secure Key Box is available in diversified form to prevent development of universal tampering schemes.)
- Not currently used as an end in-itself, rather a property of obfuscating, watermarking, or white-box crypto transformations used.

- Has applications in robustness and survivability which are currently ignored

## *Logging*

| | |
|---|---|
| *Network interaction* | 1-Way |
| *SDLC stage* | Deployment |
| *Commercial availability* | Yes* |
| *Technology specificity* | Independent |
| *Technical complexity* | Low |
| *Implementation cost* | Medium* |

What is it?

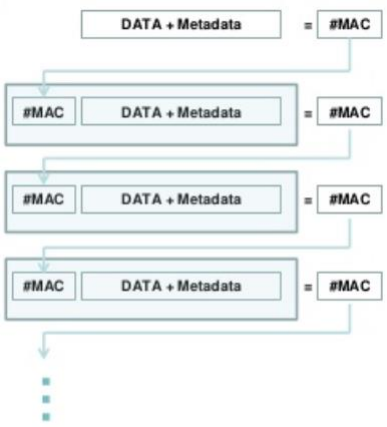
- The ability to log events on an untrusted host in such a way that events captured before a point in time (e.g. compromise) are tamper-evident

How is it applied?

- Log events are cryptographically linked on the host and intermittently transmitted to a TTP for secure storage and validation.



## *Logging – Overview*



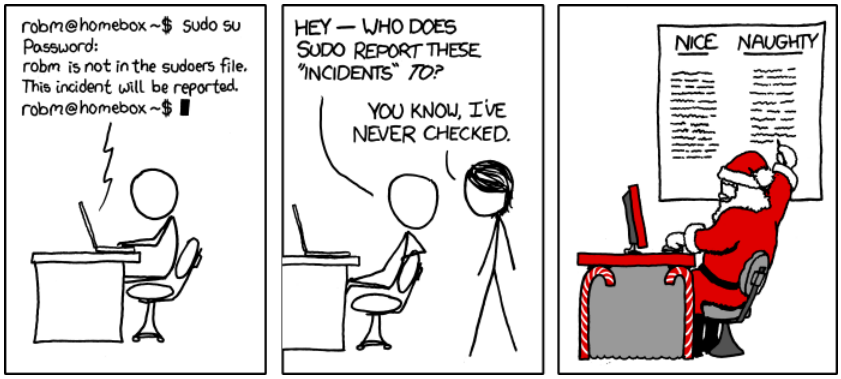
## *Logging – Properties*

There are many desirable properties

- Forward-security: Compromising the log system gives you no information about previous entries (secrecy). Attempts to modify them are detected (integrity)+
- Use of a TTP (or not)
- Seekability of log entries (i.e. fast verification of individual log entries without the need to verify an entire chain)*



## *Logging – The truth..*



## *Logging – Discussion*

- Online logging a possibility
  - Recommended to prevent attackers from physically destroying log files. Hybrid approach of regularly dispatching log entries rather than continuously.*
- Secure logging is a legal requirement of numerous regulatory bodies (e.g. PCI DSS, ISO27001, etc.). Rules for court-admissible evidence vary.
- Implemented in recent versions of journald (logging component of systemd) on most linux systems.+



## *Agenda*

1. Setting The Scene
2. Software Protection Controls
3. **Discussion**

## *Software Protection – Final thoughts*

What protection functionality is available in our arsenal?

- Obfuscation: code obfuscation, white-box crypto, code encryption
- Tamper proofing/detection: Anti-debugging, code guards, white-box crypto, logging, remote attestation
- Traitor tracing: watermarking, fingerprinting

Explicitly ignored: Execution environment protection.

Obfuscation vs. obfuscation:

- Barak's compiler O(P) -> P'
- code obfuscation transformations o(c) -> c'



## *Software Protection – Final thoughts*

- Software protection in a malicious host model is an attempt to find an Obfuscator O(P) -> P'
- Barak's impossibility result for Obfuscation suggests Software protection in a malicious host model is not possible*
- Reduced to performing "fuzzy" obfuscation without any underlying strong security guarantees.

## *Software Protection – Final thoughts and observations*

Conversion of multiple levels of protection:

- Perimeter defenses are moving into applications (when this makes sense): Runtime Application Self-Protection (RASP)
- Available commercial protection solutions combine multiple techniques for a defense-in-depth solution
- More pervasive deployment of hardened execution environments* and hardware support for security (GNX, TrustZone, NX)



## *Software Protection – Final thoughts*

Diversification, watermarking, obfuscation and white-box cryptography are closely linked concepts. Diversification is also a latent property* of the other transformations.

- Within commercial products, diversification is not currently an end in-itself. It remains a property achieved as a result of using obfuscation, watermarking, or white-box cryptography in an application.
- Most obvious commercial application is protection of high-value patch/update details+

## *Software Protection – Final thoughts*

No single protection technique is sufficient: Commercial products include:

- Static and dynamic analysis prevention: Obfuscation (varied), Anti-debugging
- Tamper Detection/prevention: Code guards (checking, repairing)
- Key protection: White-box cryptography implementations
- Traitor tracing: Explicit* (watermarking/fingerprinting) or implicit via diversification
- BORE Crack Prevention: Diversification (explicit or implicit via whitebox, obfuscation)



## *Software Protection – Final thoughts*

- No technical solution for piracy (though controls are an intrinsic part of the solution)
  - Move application to the cloud
  - Require hardware dongle (expensive applications)
- Large companies (e.g. Microsoft) combat piracy via:
  - Education, Lobbying, Traitor tracing, licensing, etc.

## *The different software solutions*

Different solutions apply different techniques to protect your software

- RASP (Runtime application self-protection)
- The Engima Protector
- Themida (using SecureEngine)
- WinLicense (using SecureEngine)
- Genode (ARM trustzone)
- HARES
- Intel GNX



## *Looking at the future?*

We are moving to “cryptographic” way of obfuscating the code

- HARES: Advanced obfuscators that make significantly raise the bar
- Intel SGX: Software guard extensions
- FHE: fully homomorphic encryption
  - Working on encrypted data without the need for decryption
- IO: Indistinguishability Obfuscation
  - Execute encrypted instructions without revealing them.

## *Software Protection – Conclusions*

Protection against 4 threats: illegal Analysis, Tampering, Copying, Use

Software-only protection often boils down to a rat race, but still useful.

Many protection techniques are commercially available ; Roll-your own is not necessarily a bad thing

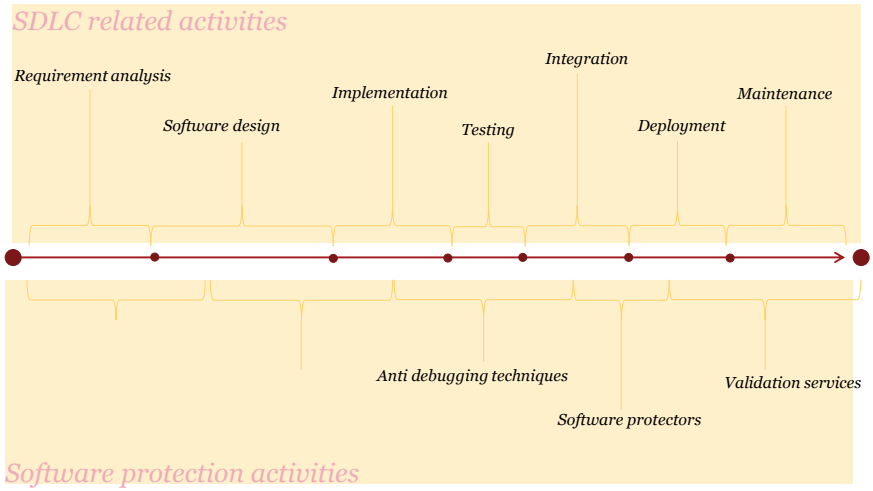
- Protection strength inversely proportional to popularity

Combination of different techniques makes the breaking of the *combination more difficult*.



## *BACKUP*

## *SDLC and software protection*






## *Applied techniques in commercial products (brainstorming)*

Metaforic Core

Uses:

- Code guards, whitebox, obfuscation,

Used by:

- licensing code (harden against analysis), embedded routers, mobile applications and medical implants (monitoring for tampering)

## *Applied techniques in commercial products (brainstorming)*

arxan:

Uses:

Used by: