First year review WP2 overview

Trento - September 24th, 2007

Goal

- To investigate software-only methodologies to implement the remote entrusting principle

Partners

- POLITO (WP leader) →

- Team:
  - Mario BALDI
  - Stefano DI CARLO
  - Paolo FALCARIN
  - Antonio DURANTE
  - Alberto SCIONTI
  - Davide D'APRILE

Partners

- POLITO (WP leader)
- UNITN

- Team:
  - Paolo TONELLA
  - Mariano CECCATO
  - Jasvir NAGRA
  - Milla DALLA PREDA
  - Amitabh SAXENA

Partners

- POLITO (WP leader)
- UNITN
- KUL

- Team:
- Bart PRENEEL
- Brecht WYSEUR
- Jan CAPPAERT
- Thomas HERLEA

Partners

- POLITO (WP leader)
- UNITN
- KUL
- SPIIRAS

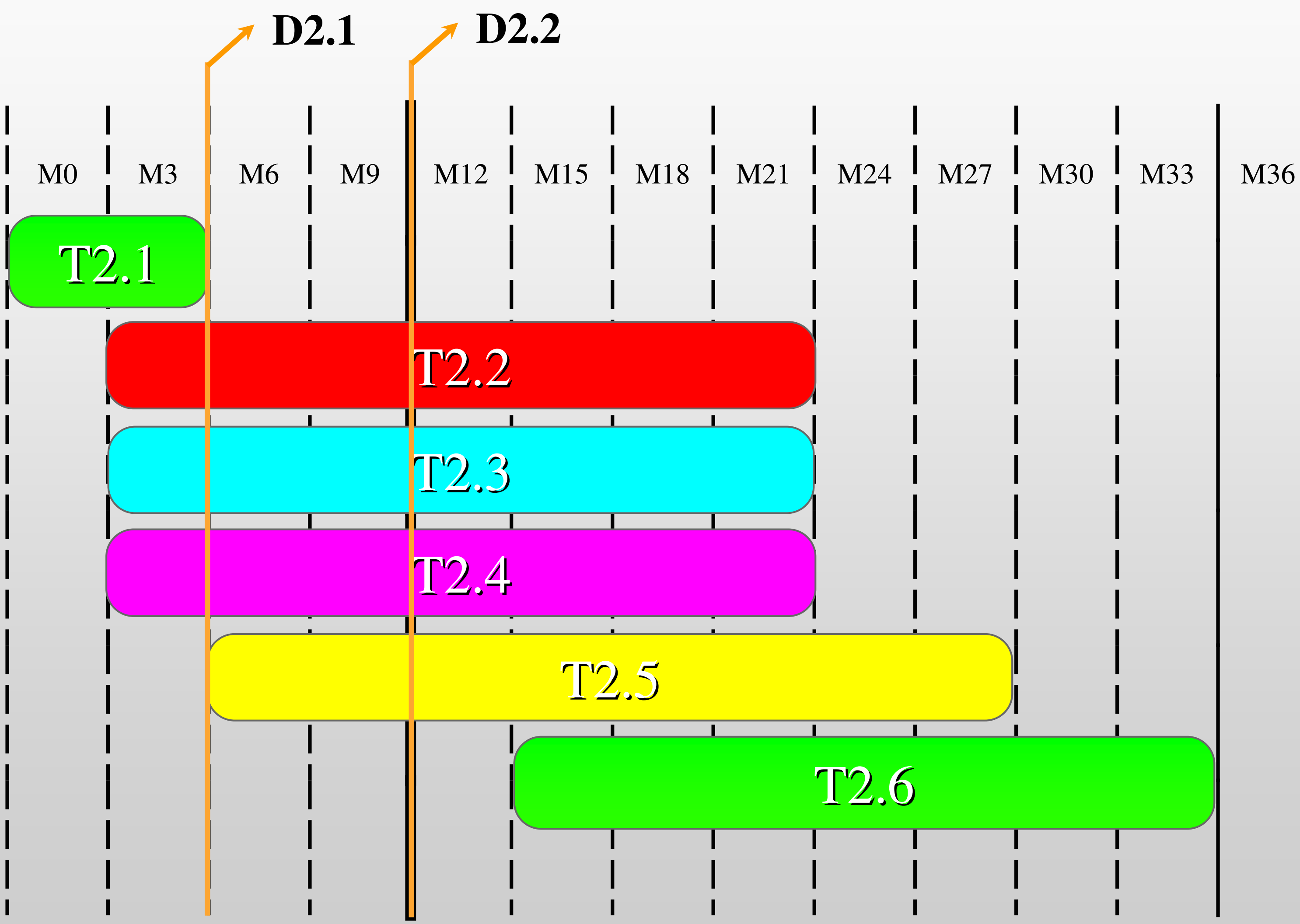
- Team:
  - Igor KOTENKO
  - Vasily DESNITSKY
  - Victor VORONTSOV
  - Vitaly BOGDANOV

Partners

- POLITO (WP leader)
- UNITN
- KUL
- SPIIRAS
- GEM

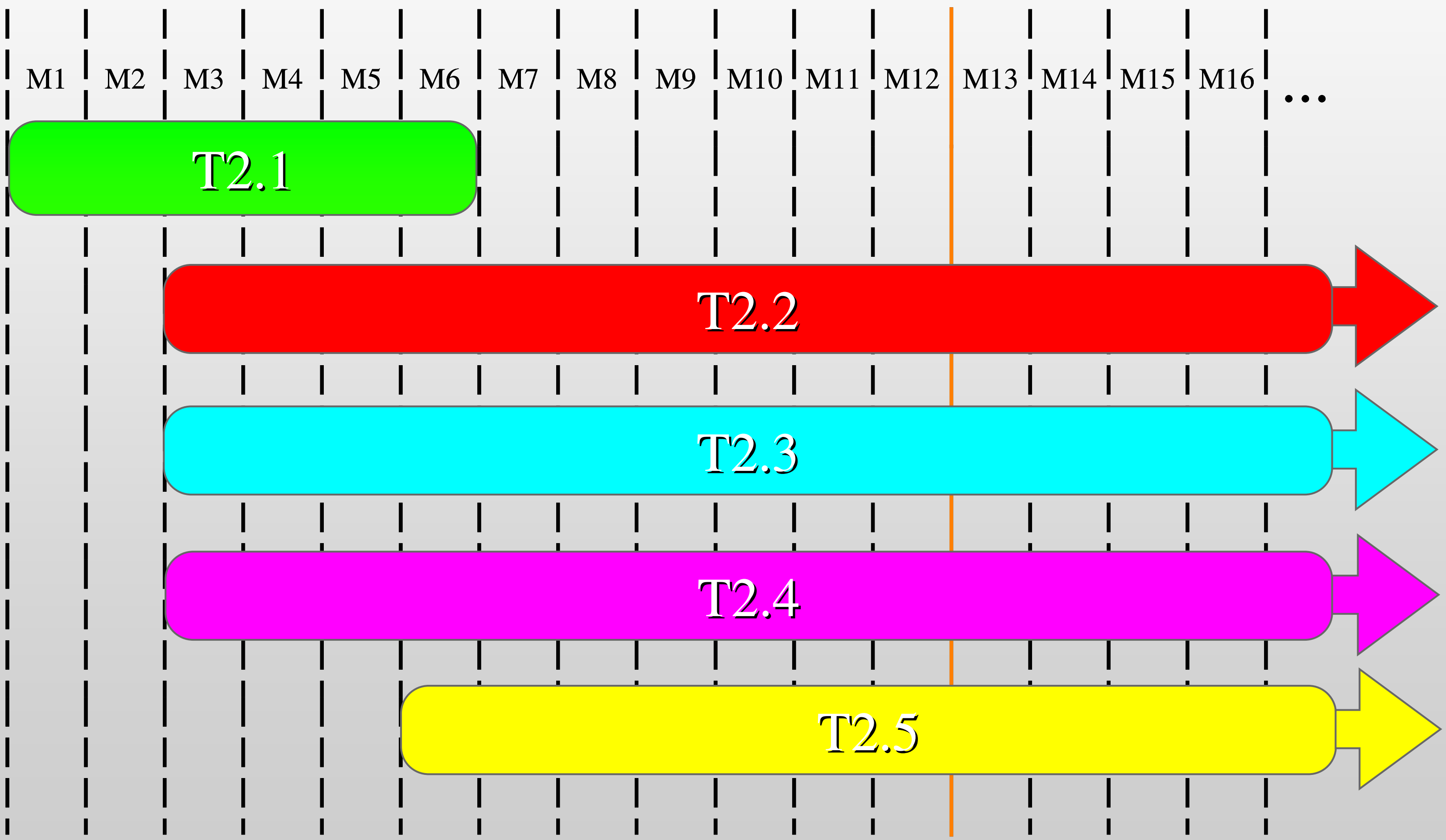
- Team:
  - Pierre GIRARDStéphane SOCIE

Tasks


D2.1
D2.2
M0
M3
M6
M9
M12
M15
M18
M21
M24
M27
M30
M33
M36
T2.1
T2.2
T2.3
T2.4
T2.5
T2.6

Tasks

T2.1


M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
M13
M14
M15
M16
…
T2.1
T2.2
T2.3
T2.4
T2.5

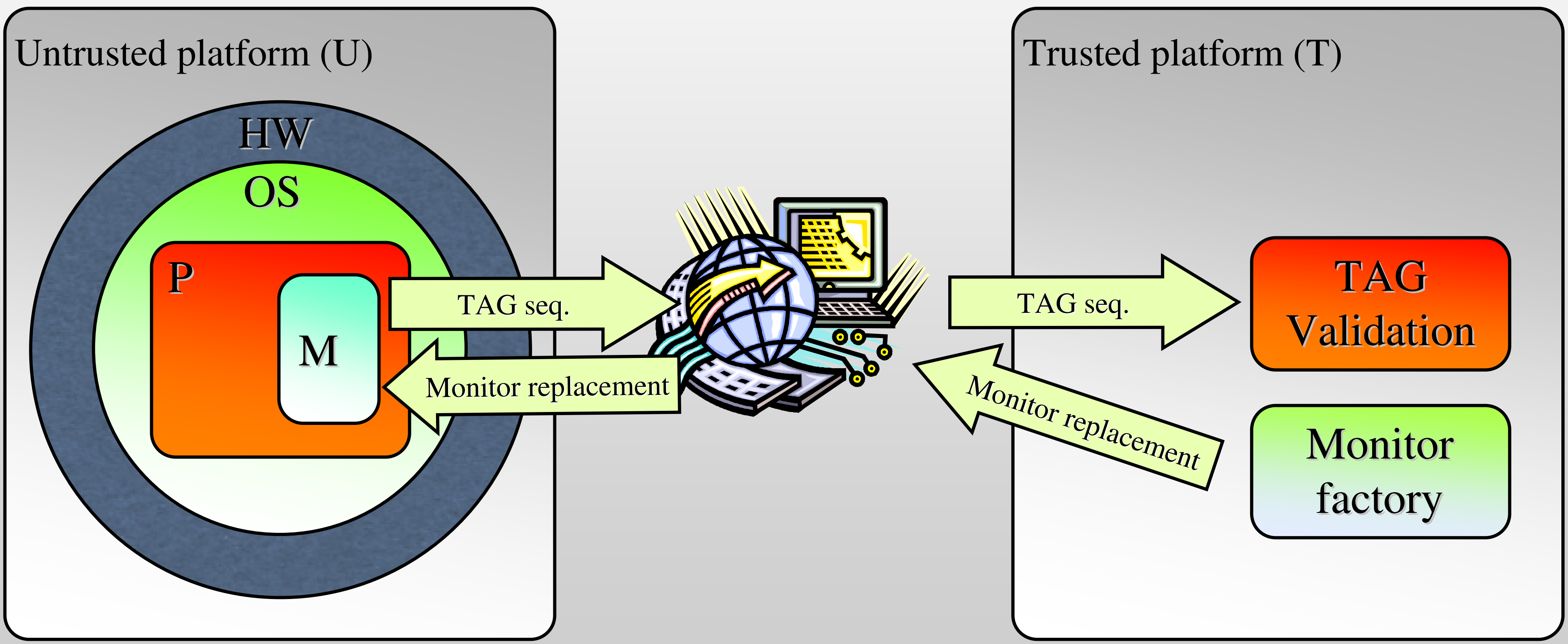
T2.1

Trust model

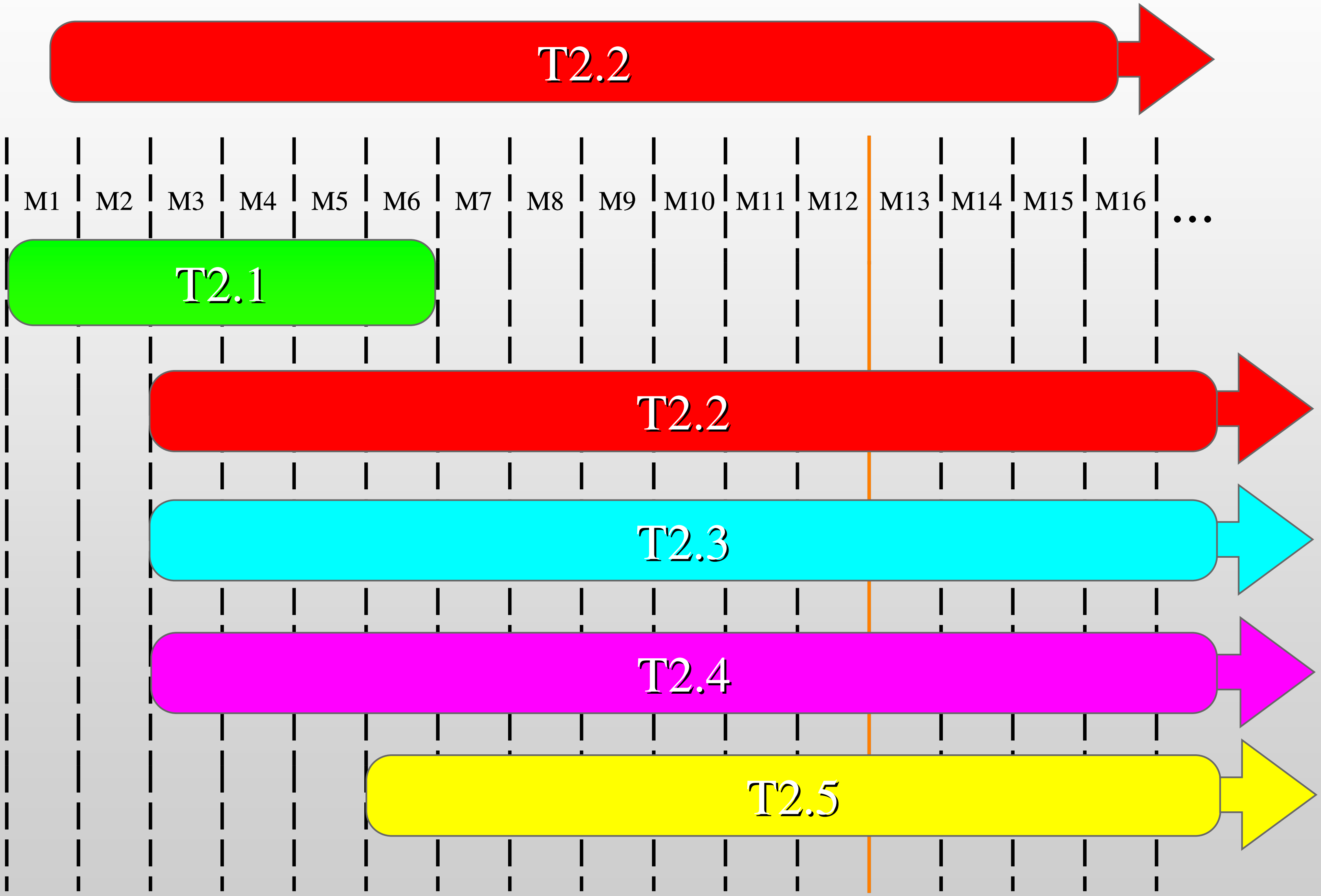
- Definition of the trust model for software only remote entrusting
- The output of this task is the deliverable D2.1:
  - Trust model and assumption for software based TR methods

T2.1

Trust model



Tasks


T2.2
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
M13
M14
M15
M16
…
T2.1
T2.2
T2.3
T2.4
T2.5

T2.2

Secure interlocking and authenticity checking

- Definition of software techniques to:
  - Securely combine the program P and the monitor M
  - Protect the authenticity of code and data of P

Invariants Monitoring (POLITO)

- A program invariant is a property that is true at a particular program execution point
- Invariant monitoring aims at detecting attacks to the state of a program P by continuously checking dynamically inferred invariants

T2.2

Invariants Monitoring (POLITO)

- Invariants monitoring workflow
  - Invariants definition (available tools: DAIKON by Michael D. Ernst)
  - Selection of the set I’ of relevant invariants to protect a subset S’ of the state of P

How to select S’ and I’ still an open challenge?

T2.2

Invariants Monitoring (POLITO)

- Definition of a monitor M able to periodically send information about S’ to T
  - T verifies if the selected list of invariants I’ is always respected
  - Any violation is detected as an attack

- Invariants monitoring is not 100% secure
- A prototype C++ application performing strings elaboration is available

Barrier slicing (UNITN)

- Aims at protecting the state of P by moving part of its code from U to T
- It presents an alternative architecture w.r.t. the presented trust model
- Trade-off between security and performance

Barrier slicing (UNITN)

- A set S of variables to protect is removed U
- Program slicing: the (executable) slice of P responsible of the variables in S is moved to T

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 - time);
x = x + delta * cos(direction);
y = y + delta * sin(direction);
Server.sendPosition(x,y);
if (track.isInBox(x, y)){
     gas = maxGas;
     lastFuel = time2;
}
else {
     gas = maxGas - (int) (time2-lastFuel);
     if (gas < 0) {
            gas = 0;
            if (speed > maxSpeed /10)
                   speed = maxSpeed /10;
            else if (speed < minSpeed/10)
                   speed = minSpeed/10;
     }
}
time = time2;
```

Barrier slicing (UNITN)

- Untrusted platform (U)
  - Use of variables in S replaced by queries and synchronization statements to T
- Trusted platform (T)
  - A barrier slicing running for each untrusted platform
  - Query and synchronization statements managed for each untrusted platform

Barrier slicing (UNITN)

- Example: barrier slice implemented on a simple car race game written in JAVA
- A complete description of the approach published
  - Mariano Ceccato, Mila Dalla Preda, Jasvir Nagra, Christian Collberg, Paolo Tonella, Barrier Slicing for Remote Software Trusting, 7th IEEE International Working Conference on Source Code Analysis and Manipulation
  - Jasvir Nagra, Mariano Ceccato and Paolo Tonella, “Distributing Trust Verification to Increase Application Performance”, in PDP2008, February 13-15, 2008, Toulose, France

White-Box Remote Procedure Call - WBRPC (UNITN, KUL)

- The name RPC implies the ability of a trusted platform T to execute an arbitrary program P on an untrusted platform U
  - In collaboration with Prof. Amir HERZBERG
- The key idea is the use of an **O**bfuscated **V**irtual **M**achine (OVM)

White-Box Remote Procedure Call - WBRPC (UNITN, KUL)

- WBRPC workflow
  - P is encrypted by T to get E(P)
  - E(P) and the inputs a are sent to U for execution under OVM

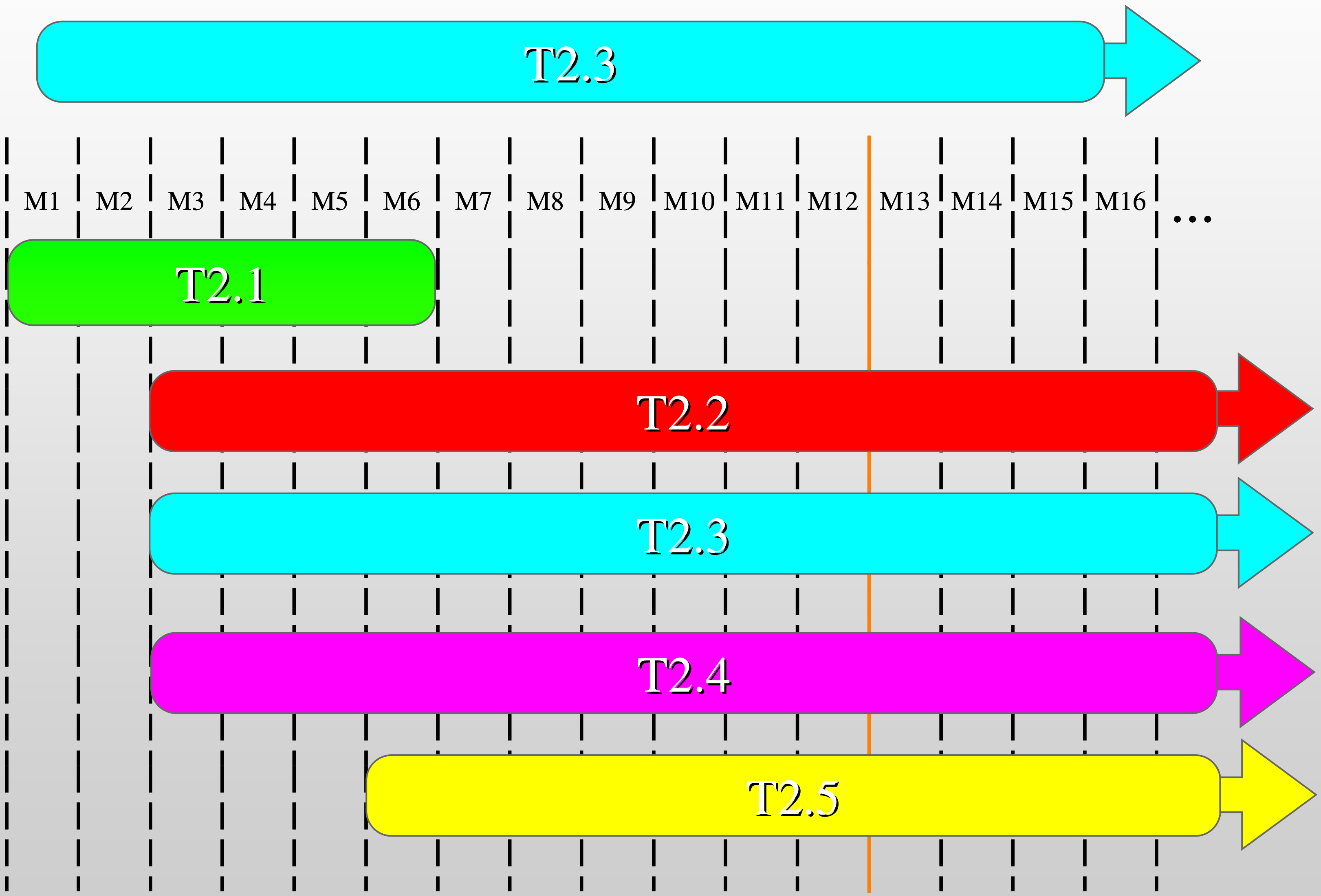
White-Box Remote Procedure Call - WBRPC (UNITN, KUL)

- OWM performs the following tasks:
  - Computes $P = D(E(P))$
  - Computes $y = P(a)$
  - Computes $z = E(y)$
  - Sends z to T
- T has the decryption key and computes $y = P(a) = D(z)$

White-Box Remote Procedure Call - WBRPC (UNITN, KUL)

- Under reasonable definition of obfuscator, we can show that OVM provides confidentiality of programs and integrity of execution of program

Tasks



T2.3
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
M13
M14
M15
M16
...
T2.1
T2.2
T2.3
T2.4
T2.5

T2.3

Dynamic replacement for increased tamper resistance

- Investigation of innovative methods exploiting the "time dimension" to increase overall tamper resistance of M
- The research activities of this task contribute to the deliverable D2.2:
  - Methods to dynamically replace the secure software module and to securely interlock applications with secure software modules

Dynamic replacement for increased tamper resistance

- Current (software-based) techniques co-bundle monitoring code with application code:
  - Position and behavior are hidden
- Threat (well-financed skilled attacker)
  - The user has full access on U and can exploit any type of reverse engineering facilities to break the monitoring code

Dynamic replacement for increased tamper resistance

- Continuos replacement of M
  - Selected software component and parameters of M are continuously replaced to make reverse engineering costs too high
- Dynamic replacement requires:
  - A mobility infrastructure
  - A binding support

Profiling Interfaces

(POLITO)

- MONO
  - C# on Linux
  - Interlocking and mobility from scratch
- JVMTI (profiling interface of JVM) JAVA on both Windows and Linux
  - Similar to MONO
  - Portable on any operating system
- Chat client prototype available

Aspect Oriented Programming
(POLITO)

- Prose (dynamic AOP tool)
  - JAVA based
  - Built-in mobility
  - Coarse-grain granularity for checking
- Chat client prototype available

Mutant C/C++ (POLITO)

- Uses self-modifying (mutant) code to implement the mobility infrastructure and the binding support
  - Native C/C++ code
  - High complexity (no built-in support for mobility)
  - Successfully applied on a small VNC client but a complete working prototype is still under construction

Tasks


T2.4
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
M13
M14
M15
M16
...
T2.1
T2.2
T2.3
T2.4
T2.5

Increased reverse engineering complexity for software protection

- This task addresses the challenging problem of pure software methods to protect the monitor M from tampering
  - The module behavior must be hidden to avoid trivial reverse engineering
  - Secret data inside the module (e.g., encryption keys) must be hidden in order to be not easily spotted

Obfuscation of Java byte code

(GEM)

- Use case definition in GEM context
  - Protect the secure link between an agent and a server
  - Avoid software modification
  - IP protection
  - Security of embedded software in PC based simulator

Obfuscation of Java byte code

(GEM)

- Classification of obfuscation transformation
  - Layout obfuscation
    - Remove debug information
    - Change identifier names
  - Data obfuscation
    - Change the way data is stored or encoded in the program

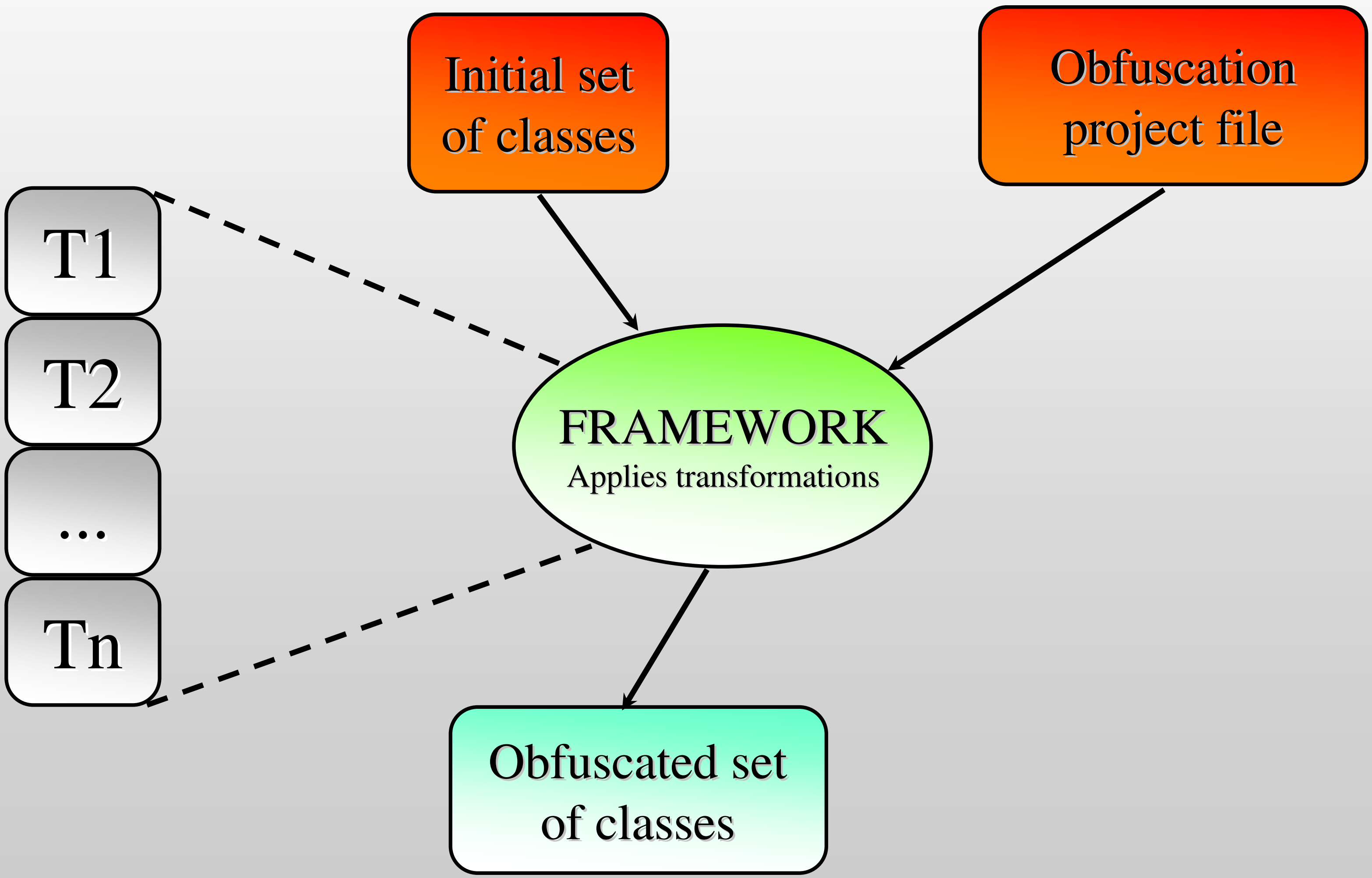
Obfuscation of Java byte code

(GEM)

- Control flow obfuscation
  - Change the way the program runs
- Preventive obfuscation
  - Try to find weakness in current de-obfuscation / decompilers to make them crash

Obfuscation of Java byte code

(GEM)



Crypto guards (KUL)

- A tamper resistance technique, in which code in an executable is encrypted
  - Guards are interleaved with the original application code
  - They create web of code dependencies
  - Decryption of code depends on other code

Crypto guards (KUL)

- During the execution of a program crypto guards make sure that:
  - The correct block of code is decrypted end executed
  - The block of code is encrypted back after the execution

C CFG flattening with TxL(KUL)

- Control Flow Graph (CFG) flattening aims at breaking down the structure of a program
  - The execution order of basic blocks is unknown (statically)
  - Dynamic analysis reveals order (if good coverage)

C CFG flattening with TxL(KUL)

- TxL: Turing eXtender Language
  - Suitable for source-to-source transformations
  - Transforms parse tree (different CFG)
- Control statements are removed form the program and transformed into a single switch-case statement

Snippets (KUL)

- Sequences of assembly inserted in assembly code after compilation (before linking):
  - They affect addresses and thus thwart offset-based cracks
  - They can break or duplicate patterns making pattern based cracks fail

Snippets (KUL)

- Our inserted snippets mostly consist of redundant code
  - Code that gets executed but does not affect the overall program behavior
- It is not trivial for compaction tools to remove all snippets completely

White-Box Cryptography(KUL)

- "Hide secret keys in software implementations of cryptographic algorithms (e.g., AES)"
  - Study of existing techniques
  - Research towards the construction of secure basic blocks and secure implementations
  - Development of a Theoretical Model

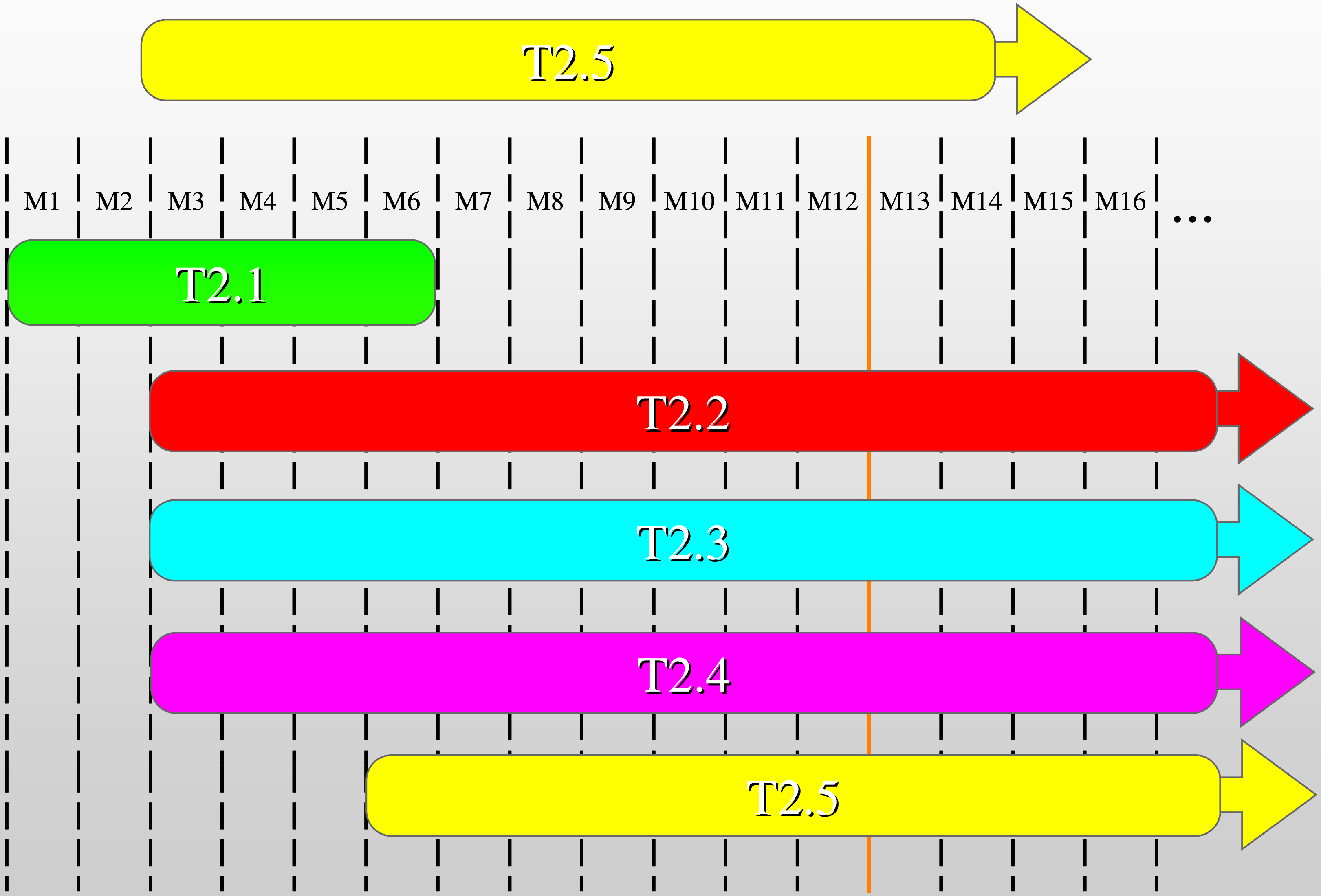
White-Box Cryptography (KUL)

- Publication of cryptanalysis of White-Box DES Implementations
  - Brecht Wyseur, and Wil Michiels, and Paul Gorissen, and Bart Preneel, "*Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings*", in Selected Areas of Cryptography, SAC 2007, August 16-17, Ottawa, Canada

Implementation (KUL)

- A real obfuscator
  - C source code to obfuscated C binary
  - Implemented techniques:
    - TXL transformations library (control flow graph flattening)
    - White-Box DES library
  - Future extensions:
    - Crypto guards (source -> binary)
    - Snippets (binary -> binary)

Tasks


T2.5
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
M13
M14
M15
M16
…
T2.1
T2.2
T2.3
T2.4
T2.5

T2.5

Design of entrusting protocol

- This task will cover the cryptographic and synchronization concerns of the communication protocol employed between the monitor and the trusted platform

T2.5

Design of entrusting protocol (SPIIRAS)

- Analysis of data flows to be involved in remote entrusting mechanisms
  - Analysis of structure of transmitted data, quantitative and time intensity assessments of data
- Development of target protocol security requirements

T2.5

Design of entrusting protocol (SPIIRAS)

- Analysis of existent network and cryptographic protocols
- Definition of network protocol facilities to fulfill the target protocol security requirements
- Analysis of formal methods for design of entrusting protocol

Future activities

- Assessment of the proposed software techniques (T2.2, T2.3 and T2.4) and investigation of alternative architectures
- Definition and implementation of a Monitor Factory (T2.3)
- Focus on the trust protocol design (T2.5)
- Early specification of the proof of concept for software only remote entrusting (T2.6)