Verifying Firmware for Confidential Computing with CPAchecker (Work in Progress)

Dirk Beyer, Po-Chun Chien, **Nian-Ze Lee**, and Thomas Lemberger

LMU Munich, Germany

2024-09-09 at the 9th International Workshop on CPAchecker

Confidential Computing

- An era of data: social media, banking, personal health information, etc.
- Threats of data breaches

Confidential Computing

- An era of data: social media, banking, personal health information, etc.
- Threats of data breaches
- Scenario: sensitive data to train ML models using cloud services

Confidential Computing

- An era of data: social media, banking, personal health information, etc.
- Threats of data breaches
- Scenario: sensitive data to train ML models using cloud services
  $\rightarrow$ Do not want to trust cloud operators

Confidential Computing

- An era of data: social media, banking, personal health information, etc.
- Threats of data breaches
- Scenario: sensitive data to train ML models using cloud services
  $\rightarrow$ Do not want to trust cloud operators

How can we protect data when they are in use, especially in a remote execution environment, e.g., cloud?

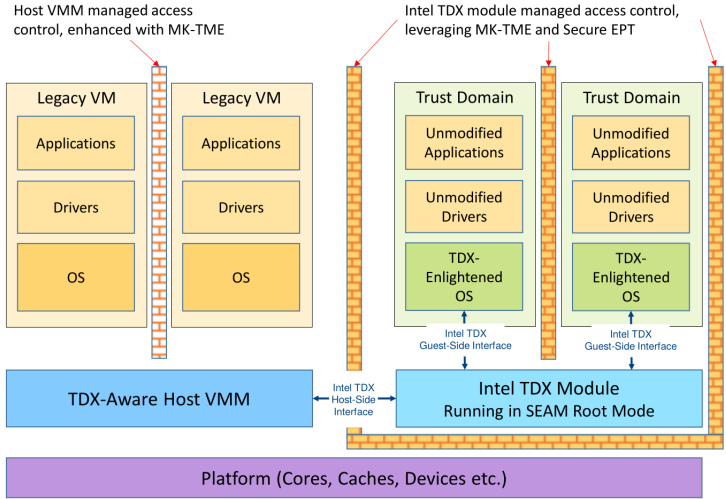
Using Trusted Execution Environment to Protect Data in Use

- Intel Trust Domain Extensions (TDX) [1]
  - Protect guest VMs against host VM manager (VMM) or OS
  - Encrypted memory and page tables, secure address translation, etc.

Using Trusted Execution Environment to Protect Data in Use

- Intel Trust Domain Extensions (TDX) [1]
  - Protect guest VMs against host VM manager (VMM) or OS
  - Encrypted memory and page tables, secure address translation, etc.
- Similar techniques
  - ARM Confidential Compute Architecture [2, 3, 4]
  - AMD Infinity Guard [5]

Intel TDX: Overview



Source: Figure 2.1 in Intel TDX Module v1.5 Base Architecture Specification

Intel TDX: Components

- Hardware extensions: new CPU mode, highest privilege
- Firmware components: TDX module
  - Loaded in special memory range
  - Executed with highest privilege

Intel TDX: Components

- Hardware extensions: new CPU mode, highest privilege
- Firmware components: TDX module
  - Loaded in special memory range
  - Executed with highest privilege
- Host VMM manages trust domains (TDs) via TDX module
  - Using application binary interfaces (ABIs), no direct access

Intel TDX: Components

- Hardware extensions: new CPU mode, highest privilege
- Firmware components: TDX module
  - Loaded in special memory range
  - Executed with highest privilege
- Host VMM manages trust domains (TDs) via TDX module
  - Using application binary interfaces (ABIs), no direct access
- ABI examples
  - VMM: create a TD, add a private page to a TD and encrypt the page, etc.
  - TD: accept a new page, read global metadata of a TDX module, etc.

Intel TDX: Components

- Hardware extensions: new CPU mode, highest privilege
- Firmware components: TDX module
  - Loaded in special memory range
  - Executed with highest privilege
- Host VMM manages trust domains (TDs) via TDX module
  - Using application binary interfaces (ABIs), no direct access
- ABI examples
  - VMM: create a TD, add a private page to a TD and encrypt the page, etc.
  - TD: accept a new page, read global metadata of a TDX module, etc.

Goal: verify ABIs of TDX module (implemented as C code plus assembly), assuming VMM and TDs can call any ABI with any inputs

Intel TDX: Properties of Interest

- Standard checks: assertion, memory safety, over/underflow, etc.
- Functional correctness: pre/post-conditions derived from TDX specification
- Information flow

Intel TDX: Properties of Interest

- Standard checks: assertion, memory safety, over/underflow, etc.
- Functional correctness: pre/post-conditions derived from TDX specification
- Information flow

Work in progress: create a benchmark set of TDX verification tasks for functional correctness (w.r.t. SV-COMP rules)

Intel TDX: Specification for ABI Function `TDG.SYS.RD`

Table 5.324: TDG.SYS.RD Input Operands Definition

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RDX	Field identifier – see 3.10 The LAST_ELEMENT_IN_FIELD and LAST_FIELD_IN_SEQUENCE components of the field identifier must be 0. WRITE_MASK_VALID, INC_SIZE, CONTEXT_CODE and ELEMENT_SIZE_CODE components of the field identifier are ignored. A value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		

Table 5.325: TDG.SYS.RD Output Operands Definition

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
RDX	If the input field identifier was -1, RDX returns the first readable field identifier. Else, in case of an error, RDX returns -1. On success, RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available.
R8	Contents of the field In case of no success, as indicated by RAX, R8 returns 0.
Other	Unmodified

Source: Intel TDX Module v1.5 ABI Specification

Firmware-Specific Constructs

- Byte/Bit-precise modeling of memory layouts
- Inline assembly
- Externally defined variables

Multiple Interpretations of Registers (Type Punning)

```
typedef union tdx_leaf_and_version_u {
  struct {
    uint64_t leaf : 16;
    uint64_t version : 8;
    uint64_t reserved0 : 8;
    uint64_t reserved1 : 32;
  };
  uint64_t raw;
} tdx_leaf_and_version_t;
```

Multiple Interpretations of Registers (Type Punning)

```c
typedef union tdx_leaf_and_version_u {
  struct {
    uint64_t leaf : 16;
    uint64_t version : 8;
    uint64_t reserved0 : 8;
    uint64_t reserved1 : 32;
  };
  uint64_t raw;
} tdx_leaf_and_version_t;
```

```c
typedef union md_field_id_u {
  struct {
    // Lower 32 bits
    union {
      struct {
        uint32_t field_code : 24;
        uint32_t reserved_0 : 8;
      };  // default field code
      struct {
        uint32_t element : 1;
        uint32_t subleaf : 7;
        ...
      } cpuid_field_code;
    };
    // Upper 32 bits
    struct {
      ...
    };
  };
  uint64_t raw;
} md_field_id_t;
```

Compiler Attributes for Memory Alignment

```c
// Size: 8 bytes
typedef struct Example_s {
  char c;
  int i;
} Example_t;

// Size: 5 bytes
typedef struct Example_packed_s {
  char c;
  int i;
} __attribute__((packed)) Example_packed_t;

// Size: 16 bytes
typedef struct Example_aligned_s {
  char c;
  int i __attribute__((aligned(8)));
} Example_aligned_t;
```

Example of Inline Assembly: Access Loader-Defined Variables

```
_STATIC_INLINE_ tdx_module_local_t* get_local_data(void) {
  uint64_t local_data_addr;
  _ASM_("movq %%gs:%c[local_data], %0\n\t"
        : "=r"(local_data_addr)
        : [local_data] "i"(
            offsetof(tdx_module_local_t, local_data_fast_ref_ptr)));
  return (tdx_module_local_t*)local_data_addr;
}
```

Challenges for Software Verifiers

- Preliminary investigation on a few examples

FW constructs	CPAchecker [6]	Cbmc [7]
Type punning	Partially supported	Supported
Compiler attribute	Partially supported	Partially supported
Inline assembly	Unsupported	Partially supported
Variable defined externally	Error	Error

Challenges for Software Verifiers

- Preliminary investigation on a few examples

FW constructs	CPAchecker [6]	Cbmc [7]
Type punning	Partially supported	Supported
Compiler attribute	Partially supported	Partially supported
Inline assembly	Unsupported	Partially supported
Variable defined externally	Error	Error

- CPAchecker: large `enum` value, compiler attribute, static assertion, anonymous `union`/`struct`
- Cbmc: nested compiler attributes

Challenges for Software Verifiers

- Preliminary investigation on a few examples

FW constructs	CPAchecker [6]	Cbmc [7]
Type punning	Partially supported	Supported
Compiler attribute	Partially supported	Partially supported
Inline assembly	Unsupported	Partially supported
Variable defined externally	Error	Error

Hardware modeling (e.g., inline assembly) is verifier-agnostic: avoid reinventing the wheel in every verifier by program transformation [8]

Modeling Inline Assembly via Program Instrumentation

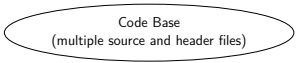
```
_STATIC_INLINE_ tdx_module_local_t* get_local_data(void) {
#ifdef TDXFV_NO_ASM
  return &local_data_fv;
#else
  uint64_t local_data_addr;
  _ASM_("movq %%gs:%c[local_data], %0\n\t"
        : "=r"(local_data_addr)
        : [local_data] "i"(
            offsetof(tdx_module_local_t, local_data_fast_ref_ptr)));
  return (tdx_module_local_t*)local_data_addr;
#endif
}
```

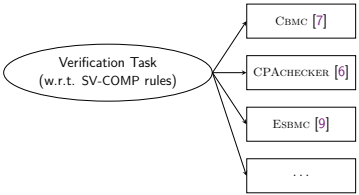
Modeling Inline Assembly via Program Instrumentation

```
_STATIC_INLINE_ tdx_module_local_t* get_local_data(void) {
#ifdef TDXFV_NO_ASM
  return &local_data_fv;
#else
  uint64_t local_data_addr;
  _ASM_("movq %%gs:%c[local_data], %0\n\t"
        : "=r"(local_data_addr)
        : [local_data] "i"(
            offsetof(tdx_module_local_t, local_data_fast_ref_ptr)));
  return (tdx_module_local_t*)local_data_addr;
#endif
}
```

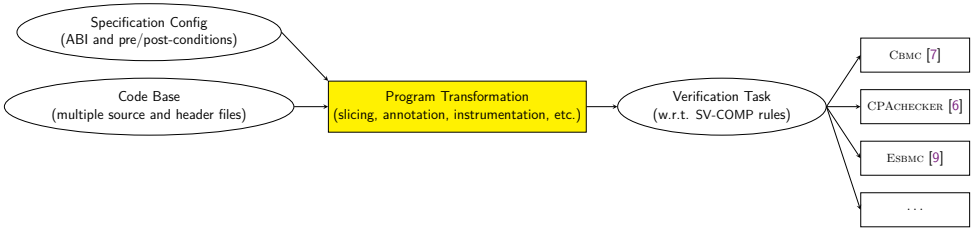
- Desired: `__VERIFIER_nondet_struct_X()` for initializing an arbitrary struct

Bridging Industrial Code Base and Academic Software Verifiers by Program Transformation



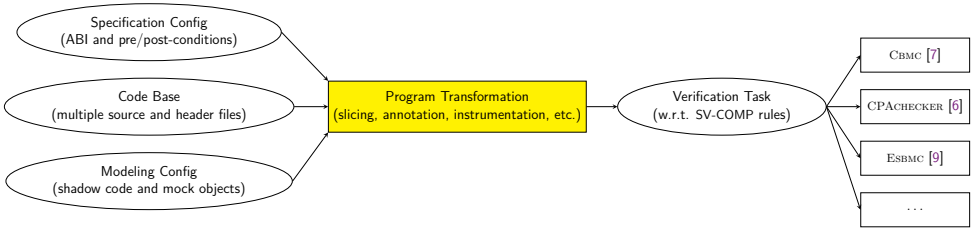


Bridging Industrial Code Base and Academic Software Verifiers by Program Transformation



- Slice off irrelevant code
- Annotate pre/post-conditions

Bridging Industrial Code Base and Academic Software Verifiers by Program Transformation



- Slice off irrelevant code
- Annotate pre/post-conditions
- Instrument shadow C code to model inline assembly

Firmware Verification: What We Have Learned So Far

- Manage and communicate with hardware
  - Data structure
    - Byte/Bit-precise and polymorphic types to simulate register/memory layouts
    - No dynamic allocation
  - Operation
    - Inline assembly to call instructions $\rightarrow$ hardware modeling (manually)
    - More data movement and less arithmetic
    - Statically-sized loops $\rightarrow$ BMC (but bounds are high)

Firmware Verification: What We Have Learned So Far

- Manage and communicate with hardware
  - Data structure
    - Byte/Bit-precise and polymorphic types to simulate register/memory layouts
    - No dynamic allocation
  - Operation
    - Inline assembly to call instructions $\rightarrow$ hardware modeling (manually)
    - More data movement and less arithmetic
    - Statically-sized loops $\rightarrow$ BMC (but bounds are high)
- Transformation as a way to bridge gaps and unify knowledge [8]

Firmware Verification: What We Have Learned So Far

- Manage and communicate with hardware
  - Data structure
    - Byte/Bit-precise and polymorphic types to simulate register/memory layouts
    - No dynamic allocation
  - Operation
    - Inline assembly to call instructions $\rightarrow$ hardware modeling (manually)
    - More data movement and less arithmetic
    - Statically-sized loops $\rightarrow$ BMC (but bounds are high)
- Transformation as a way to bridge gaps and unify knowledge [8]
- Working with industry

References I

[1] Intel Trust Domain Extensions, https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html, accessed: 2024-05-01

[2] Arm Confidential Compute Architecture, https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture, accessed: 2024-09-02

[3] Li, X., Li, X., Dall, C., Gu, R., Nieh, J., Sait, Y., Stockwell, G.: Design and verification of the Arm confidential compute architecture. In: Proc. OSDI. pp. 465–484. USENIX Association (2022), https://www.usenix.org/system/files/osdi22-li.pdf

[4] Fox, A.C.J., Stockwell, G., Xiong, S., Becker, H., Mulligan, D.P., Petri, G., Chong, N.: A verification methodology for the Arm confidential computing architecture: From a secure specification to safe implementations. Proc. ACM Program. Lang. **7**(OOPSLA1), 376–405 (2023). doi:10.1145/3586040

[5] AMD Infinity Guard, https://www.amd.com/en/products/processors/server/epyc/infinity-guard.html, accessed: 2024-09-02

[6] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). doi:10.1007/978-3-642-22110-1_16

[7] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). doi:10.1007/978-3-540-24730-2_15

References II

[8] Beyer, D., Lee, N.Z.: The transformation game: Joining forces for verification. Springer (2024), https://www.sosy-lab.org/research/pub/2024-Katoen60.The_Transformation_Game_Joining_Forces_for_Verification.pdf

[9] Gadelha, M.R., Monteiro, F.R., Morse, J., Cordeiro, L.C., Fischer, B., Nicole, D.A.: ESBMC 5.0: An industrial-strength C model checker. In: Proc. ASE. pp. 888–891. ACM (2018). doi:10.1145/3238147.3240481