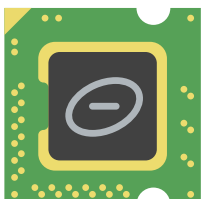
Damn Vuln IoT SoC: A Modular Platform To Learn Hardware Bug Exploitation

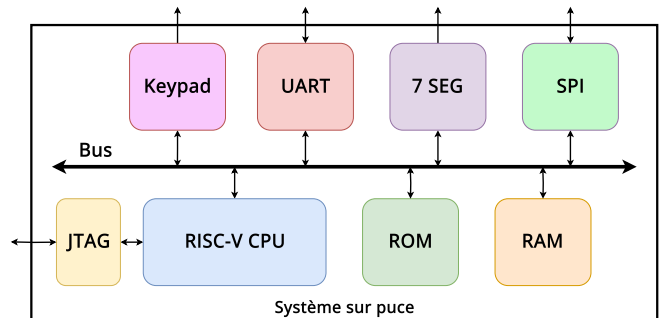
Adam HENAULT, Philippe TANGUY

Master CSSE
Université Bretagne Sud
Lab-STICC (UMR 6285)

SSTIC 2024

Qu’est-ce qu’un SoC ?





Quels sont les enjeux de sécurité ?

Problématique
Erreurs de description matériel dans un système sur puce.

Impacts

- Vulnérabilités pouvant être exploitées depuis le logiciel.
- Vulnérabilités pouvant être exploitées depuis le matériel.
- Difficultés à corriger d'éventuelles failles de sécurité.

Liste CWE (MITRE) répertorie la plupart des bugs.

CWE-1298 : Hardware Logic Race Conditions

```
module dma # (...)(...);
...
   input [7:0] [16-1:0] pmpcfg_i;
   input logic [16-1:0][53:0] pmpaddr_i;
   ...
   //// Save the input command
   always @ (posedge clk_i or negedge rst_ni)
      begin: save_inputs
      if (!rst_ni)
         begin
         ...
         end
      else
         begin
            if (dma_ctrl_reg == CTRL_IDLE || dma_ctrl_reg == CTRL_DONE)
            begin
            ...
            end
         end
      end // save_inputs
      ...
      // Load/store PMP check
      pmp #(
         .XLEN ( 64 ),
         .PMP_LEN ( 54 ),
         .NR_ENTRIES ( 16 )
      ) i_pmp_data (
         .addr_i ( pmp_addr_reg ),
         .priv_lvl_i ( riscv::PRIV_LVL_U ),
         .access_type_i ( pmp_access_type_reg ),
         // Configuration
         .conf_addr_i (pmpaddr_i ),
         .conf_i (pmpcfg_i ),
         .allow_o ( pmp_data_allow )
      );
endmodule
```

```
module dma # (...)(...);
...
   input [7:0] [16-1:0] pmpcfg_i;
   input logic [16-1:0][53:0] pmpaddr_i;
   ...
   reg [7:0] [16-1:0] pmpcfg_reg;
   reg [16-1:0][53:0] pmpaddr_reg;
   ...
   //// Save the input command
   always @ (posedge clk_i or negedge rst_ni)
      begin: save_inputs
      if (!rst_ni)
         begin
         ...
         pmpaddr_reg <= 'b0 ;
         pmpcfg_reg <= 'b0 ;
         end
      else
         begin
            if (dma_ctrl_reg == CTRL_IDLE || dma_ctrl_reg == CTRL_DONE)
            begin
            ...
            pmpaddr_reg <= pmpaddr_i;
            pmpcfg_reg <= pmpcfg_i;
            end
         end
      end // save_inputs
      ...
      // Load/store PMP check
      pmp #(
         .XLEN ( 64 ),
         .PMP_LEN ( 54 ),
         .NR_ENTRIES ( 16 )
      ) i_pmp_data (
         .addr_i ( pmp_addr_reg ),
         .priv_lvl_i ( riscv::PRIV_LVL_U ), // we intend to apply filter on
         // DMA always, so choose the least privilege .access_type_i ( pmp_access_type_reg ),
         // Configuration
         .conf_addr_i (pmpaddr_reg ),
         .conf_i (pmpcfg_reg ),
         .allow_o ( pmp_data_allow )
      );
endmodule
```

Pourquoi Damn Vuln IoT SoC ?

Objectif

- **Instrumenter** un outil de génération de système sur puce afin d'introduire de façon **automatique** des **vulnérabilités matérielles**.
- Développer une **plateforme éducative** afin d'illustrer les problématiques de la sécurité matérielle pour des cours ou pour des CTF.

Produits	Bugs Logiciels	Bugs Matériels	Modularité	Accessibilité
Attify Exploitation Learning	✓	✗	✗	✗
DIVA IoT Board	✓	✗	✗	✗
HaHa Board	✓	✓	✗	✗
HACK@DAC 2018	✗	✓	✗	✓
Damn Vuln IoT SoC	✓	✓	✓	✓

Pourquoi Damn Vuln IoT SoC ?

Objectif

- **Instrumenter** un outil de génération de système sur puce afin d'introduire de façon **automatique** des **vulnérabilités matérielles**.
- Développer une **plateforme éducative** afin d'illustrer les problématiques de la sécurité matérielle pour des cours ou pour des CTF.

Produits	Bugs Logiciels	Bugs Matériels	Modularité	Accessibilité
Attify Exploitation Learning	✓	✗	✗	✗
DIVA IoT Board	✓	✗	✗	✗
HaHa Board	✓	✓	✗	✗
HACK@DAC 2018	✗	✓	✗	✓
Damn Vuln IoT SoC	✓	✓	✓	✓

Pourquoi Damn Vuln IoT SoC ?

Objectif

- **Instrumenter** un outil de génération de système sur puce afin d'introduire de façon **automatique** des **vulnérabilités matérielles**.
- Développer une **plateforme éducative** afin d'illustrer les problématiques de la sécurité matérielle pour des cours ou pour des CTF.

Produits	Bugs Logiciels	Bugs Matériels	Modularité	Accessibilité
Attify Exploitation Learning	✓	✗	✗	✗
DIVA IoT Board	✓	✗	✗	✗
HaHa Board	✓	✓	✗	✗
HACK@DAC 2018	✗	✓	✗	✓
Damn Vuln IoT SoC	✓	✓	✓	✓

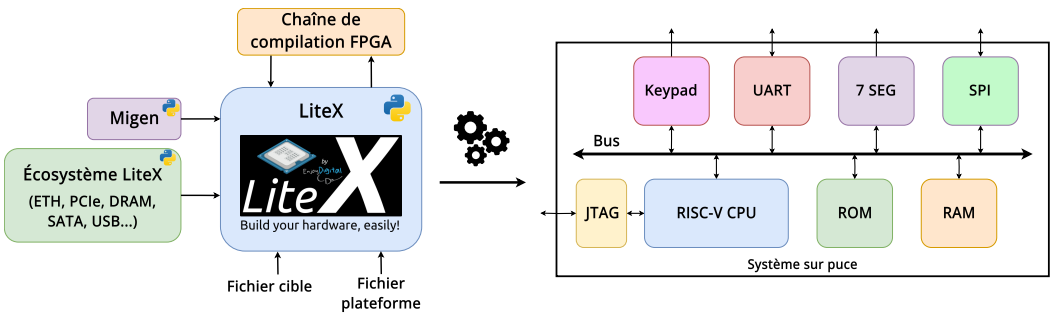
Les objectifs de Damn Vuln IoT SoC

- Catalogue de composants vulnérables et configurables.
- Plateforme modulaire.
- Open source.
- Pas de matériel spécifique (FPGA).

Présentation de LiteX

Informations

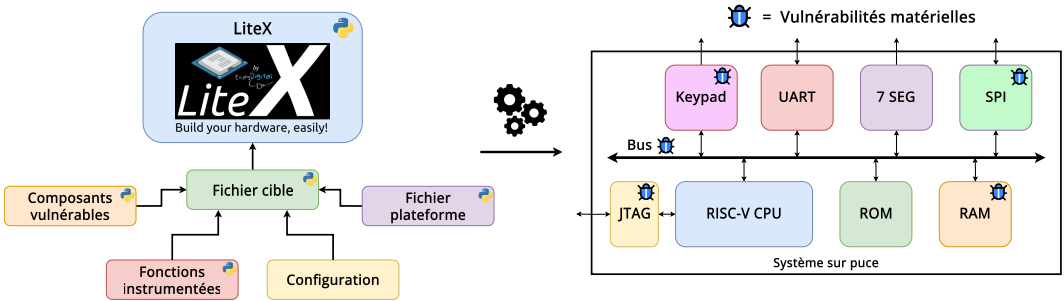
- Open source et développé en Python.
- Écosystème open source (LiteSPI, LiteScope, LiteSATA, LitePCIe...).
- Prévu pour la simulation avec Verilator.



Architecture de Damn Vuln IoT SoC

Étapes de génération du matériel

- Configuration des composants vulnérables.
- Instrumentation des fonctions de LiteX à partir du fichier de configuration.
- Génération du système sur puce contenant les vulnérabilités.



Instrumentation de LiteX

Création d'un overlap d'adresse entre 2 registres (CWE-1316)

- Récupération de la configuration.
- Allocation d'un même espace mémoire pour 2 registres.

```
# Alloc ------------------------------------------------------------------------------------
def alloc(self, name):
    config_parser = Config.getInstance()
    for n in range(self.n_locs):
        if n not in self.locs.values():
            if (name == "leds" or name == "jtag") and config_parser.getValue('jtag_lock_overlap') is True:
                return 10
            return n
    self.logger.error("Not enough Locations.")
```

Résultat de l’instrumentation

- Vulnérabilité exploitable depuis le logiciel.
- Accès au composant JTAG depuis les fonctions de contôle des LED.

```
...

/* jtag */
#define CSR_JTAG_BASE (CSR_BASE + 0x5000L)
#define CSR_JTAG_JTAG_LOCK_ADDR (CSR_BASE + 0x5000L)
#define CSR_JTAG_JTAG_LOCK_SIZE 1
static inline uint32_t jtag_jtag_lock_read(void) {
    return csr_read_simple((CSR_BASE + 0x5000L));
}
static inline void jtag_jtag_lock_write(uint32_t v) {
    csr_write_simple(v, (CSR_BASE + 0x5000L)); // Write to the address CSR_BASE + 0x5000
}

/* leds */
#define CSR_LEDS_BASE (CSR_BASE + 0x5000L)
#define CSR_LEDS_OUT_ADDR (CSR_BASE + 0x5000L)
#define CSR_LEDS_OUT_SIZE 1
static inline uint32_t leds_out_read(void) {
    return csr_read_simple((CSR_BASE + 0x5000L));
}
static inline void leds_out_write(uint32_t v) {
    csr_write_simple(v, (CSR_BASE + 0x5000L)); // Write to the address CSR_BASE + 0x5000
}
  return go(f, seed, [])
}
```

Liste de bugs supportés

Composants Damn Vuln IoT SoC

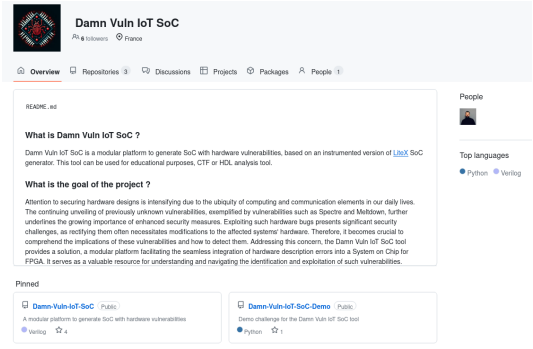
- 8 vulnérabilités matérielles.
- 2 composants vulnérables (JTAG et Keypad).

Correspondance CWE (non-exhaustif)

- CWE-1260: Improper Handling of Overlap Between Protected Memory Ranges.
- CWE-1189: Improper Isolation of Shared Resources on System-on-a-Chip.
- CWE-1191: On-Chip Debug and Test Interface With Improper Access Control.
- CWE-1256: Improper Restriction of Software Interfaces to Hardware Features.
- CWE-1316: Fabric-Address Map Allows Programming of Unwarranted Overlaps of Protected and Unprotected Ranges.

Conclusion

- Plateforme jeune et toujours en développement.
- Disponible sur GitHub (`https://github.com/Damn-Vuln-IoT-SoC`)
- Démonstration de différents challenges disponible sur GitHub.
- Utilisation de la plateforme pour développer un CTF.



Perspectives

- Améliorer la plateforme
  - Proposer plus d'instrumentations de fonction.
  - Étendre le catalogue de composants vulnérables.
- Expérimenter des outils d'analyse de vulnérabilité des SoC (SoCFuzzer[1], FEMU[2], SHarPen[3], etc.)

[1] Hossain et al., "SoCFuzzer: SoC Vulnerability Detection using Cost Function enabled Fuzz Testing", 2023

[2] Li et al., "FEMU: A firmware-based emulation framework for SoC verification", 2010

[3] Al-Shaikh et al., "SHarPen: SoC Security Verification by Hardware Penetration Test", 2023

Merci pour votre attention.

Question ?

Merci à Mohamed AFASSI, Seydina Oumar NIANG et Rébecca SZABO

Ainsi qu'à la communauté LiteX

Lab-STICC

Université Bretagne Sud

ubs