RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Inria

cea

université de BORDEAUX

Analyse automatisée d'une bibliothèque crypographique

Détection de failles par canal auxiliaire par analyse statique et symbolique

Duzés Florian

Master Cryptologie et Sécurité Informatique

28 août 2025

Opérations dangereuses

Opérations influantes :

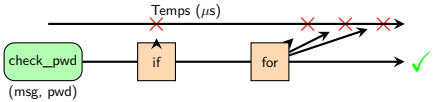
- Accès mémoire
- Décalage/rotation de valeurs
- Saut conditionnel
- Division/multiplication

Opérations dangereuses

Opérations influantes :

- Accès mémoire
- Décalage/rotation de valeurs (caché)
- Saut conditionnel
- Division/multiplication

```
1   bool check_pwd(msg, pwd){
2   if (msg.length != pwd.length){
3       return False
4   }
5   for(int i = 0; i < msg.length; i++){
6       if(msg[i] != pwd[i]){
7           return False
8       }
9   }
10  return True
11  }
```



Plus de problème ?

Plus de problème ?

Mauvaises nouvelles ?

2019 : Daniel, Bardin et Rezk, *Binsec/Rel : Efficient Relational Symbolic Execution for Constant-Time at Binary-Level*

Plus de problème ?

Mauvaises nouvelles !

2019 : Daniel, Bardin et Rezk, *Binsec/Rel : Efficient Relational Symbolic Execution for Constant-Time at Binary-Level*

2024 : Schneider et al., *Breaking Bad : How Compilers Break Constant-Time Implementations*

Spécialisations

Outil	Cible	Techn.	Garanties
ctgrind [Lan10]	Binaire	Dynamique	▲
ABPV13 [Alm+13]	C	Formel	●
VirtualCert [Bar+14]	x86	Formel	●
ct-verif [Bar+16]	LLVM	Formel	●
FlowTracker [RPA16]	LLVM	Formel	●
Blazer [Ant+17]	Java	Formel	●
BPT17 [BPT17]	C	Symbolique	▲
MemSan [Tea17]	LLVM	Dynamique	▲
Themis [CFD17]	Java	Formel	●
COCO-CHANNEL [Bre+18]	Java	Symbolique	●
DATA [Wei+20] ; [Wei+18]	Binaire	Dynamique	▲
MicroWalk [Wic+18]	Binaire	Dynamique	▲
timecop [Nei18]	Binaire	Dynamique	▲
SC-Eliminator [Wu+18]	LLVM	Formel	●
Binsec/Rel [DBR19]	Binaire	Symbolique	▲
CT-WASM [Wat+19]	WASM	Formel	●
FaCT [Cau+19]	DSL	Formel	●
haybale-pitchfork [Dis20]	LLVM	Symbolique	▲

Liste d'outils de vérification

Source : [Jan+21]

Cible

[C, Java] Code source
Binaire Binaire
DSL Surcouche de langage
Trace Trace d'exécution
WASM Assembleur web

Techn.

Formel Programmation formelle
[*] type d'analyse

Garanties (attaques temporelles)

● = Analyse correcte, ▲ = Limitée

L'outil idéal

Binsec

Binary Security[a] est une plateforme open source développée pour évaluer la sécurité des logiciels au niveau binaire.
Il est notamment utilisé pour la recherche de vulnérabilités, la désobfuscation de logiciels malveillants et la vérification formelle de code binaire. Grâce à l'exécution symbolique, Binsec peut explorer et modéliser le comportement d'un programme pour détecter des erreurs ; cette détection est réalisée en association avec des outils de fuzzing et/ou des solveurs SMT.

a. `https://binsec.github.io/`

Premiers scripts

Code : Instructions permettant de trouver le mot d'un passe d'un binaire d'exercice

```
1  starting from core with
2    argv<64> := rsi
3    arg1<64> := @[argv + 8, 8]
4    size<64> := nondet            # 0 < strlen(argv[1]) < 128
5    assume 0 < size < 128,       all_printables<1> := true
6    @[arg1, 128] := 0
7    for i<64> in 0 to size - 1 do
8      @[arg1 + i] := nondet as password
9      all_printables := all_printables && " " <= password <= "~"
10   end
11   assume all_printables
12 end
13
14 replace <puts>, <printf> by return end
15
16 reach <puts> such that @[rdi, 14] = "Good password!"
17 then print ascii stream password
18
19 cut at <puts> if @[rdi, 17] = "Invalid password!"
```

Simplification

Codes : Instructions permettant d'analyser le code précédent compilé vers Risc-V 32bits

```
1  #include <stdlib.h>
2  #include <stdint.h>
3  #include "Hacl_P256.h"
4
5  #define SIZE 4
6  uint64_t cin;
7  uint64_t x[SIZE]; uint64_t y[SIZE]; uint64_t r[SIZE];
8
9  int main(){
10   bn_cmovznz4(r, cin, x, y);
11 }
```

Adaptation

Codes : Instructions permettant d'analyser le code précédent compilé vers Risc-V 32bits

```
load sections .plt, .text, .rodata, .data, .got, .got.plt, .bss from file

secret global  r, cin, y, x

starting from <main>

with concrete stack pointer
halt at  0x0000000000000464
explore all

```

Premiers pas vers l'automatisation

Table : Tableau de résultats d'analyse Binsec pour architecture ARMv7 et ARMv8

opt\fonction analysée	cmovznz4				
Clang+LLVM	14.0.6	15.0.6	16.0.4	17.0.6	18.1.8
-O0	✓	✓	✓	✓	✓
-O1	✓	✓	✓	✓	✓
-O2	✓	✓	✓	✓	✓
-O3	✓	✓	✓	✓	✓
-Os	✓	✓	✓	✓	✓
-Oz	✓	✓	✓	✓	✓

✓ : *binary secure* ; ∼ : *binary unknown* ; × : *binary insecure*

Reherche de failles

Table : Tableau de résultats d'analyse Binsec pour architecture Risc-V

opt\fonction analysée	**cmovznz4** - 64 bits		**cmovznz4** - 32 bits	
Compilateur et architecture	gcc 15.1.0	clang 19.1.7	gcc 15.1.0	clang 19.1.7
-O0	∼	×	∼	×
-O1	✓	×	✓	×
-O2	✓	×	✓	×
-O3	✓	×	✓	×
-Os	✓	×	✓	×
-Oz	✓	×	✓	×

✓ : *binary secure* ; ∼ : *binary unknown* ; × : *binary insecure*

Cahier des charges

Objectifs du futur outil

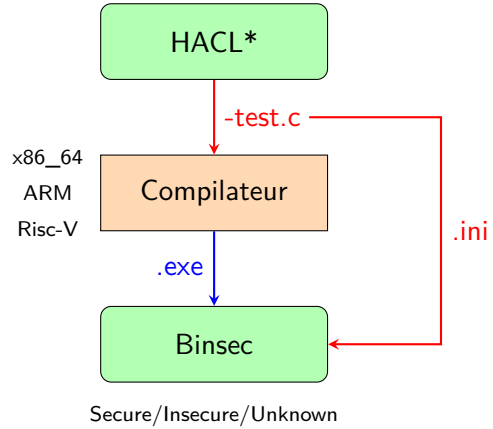
- Petits fichiers binaire
- Analyse complète de la bibliothèque
- Analyse correcte
- Automatique

Cahier des charges

Objectifs du futur outil

- Petits fichiers binaire
- Analyse complète de la bibliothèque
- Analyse correcte
- Automatique
- Couverture [architectures, compilateurs]

Figure : Flot de travail de l'outil



Conception générale

graphes

Spécifications architecturales

graphes

Constructions en modules

graphes

Andhrímnir

Besoins

Conception générale

graphes

Premières passes

graphes

Analyses

graphes

Conclusion

Références

[Alm+13] José Bacelar Almeida et al. *Formal Verification of Side-Channel Countermeasures Using Self-Composition*. 2013.

[Ant+17] Thomas Antonopoulos et al. *Decomposition Instead of Self-Composition for Proving the Absence of Timing Channels*. 2017.

[Bar+14] Gilles Barthe et al. *System-level non-interference for constant-time cryptography*. 2014.

[Bar+16] Gilles Barthe et al. *Computer-Aided Verification for Mechanism Design*. 2016.

[BPT17] Sandrine Blazy, David Pichardie et André Trieu. *Verifying Constant-Time Implementations by Abstract Interpretation*. 2017.

[Bre+18] Thomas Brennan et al. *Symbolic Path Cost Analysis for Side-Channel Detection*. 2018.

[Cau+19] Srinath Cauligi et al. *FaCT : A DSL for timing-sensitive computation*. 2019.

[CFD17] Jie Chen, Yu Feng et Isil Dillig. *Precise detection of side-channel vulnerabilities using quantitative cartesian hoare logic*. 2017.

[DBR19] Lesly-Ann Daniel, Sébastien Bardin et Tamara Rezk. *Binsec/Rel : Efficient Relational Symbolic Execution for Constant-Time at Binary-Level*. 2019. arXiv : 1912.08788. URL : http://arxiv.org/abs/1912.08788.

[Dis20] Craig Disselkoen. *haybale-pitchfork*. https://github.com/PLSysSec/haybale-pitchfork