Analyse de code binaire contre les attaques en faute

Workshop PROSECCO

06-11-18

Jean-Baptiste Bréjon

Encadrement :

Emmanuelle Encrenaz
Karine Heydemann
Quentin Meunier

Introduction

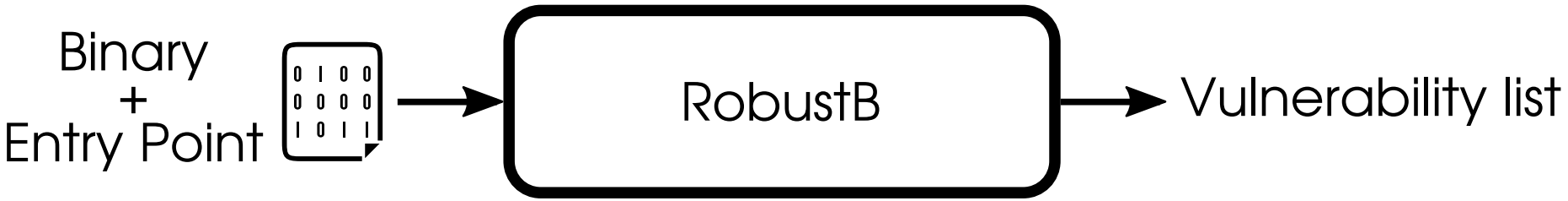
Objectif : évaluer la robustesse d'une zone de code binaire vis-à-vis d'attaques en faute

- Code binaire

  $\rightarrow$ permet de prendre en compte les effets du compilateur sur les protections implémentées au niveau du code source

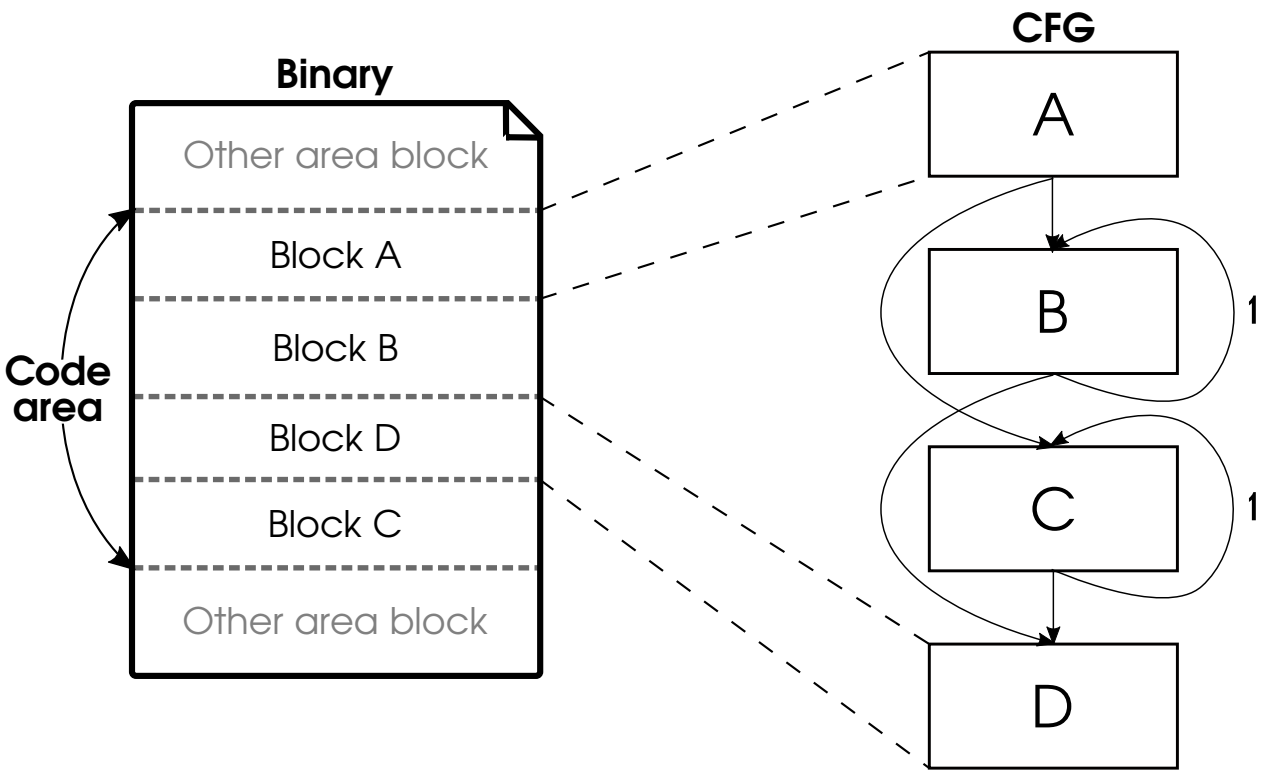
  $\rightarrow$ accès au placement de code
- Vérification formelle $\rightarrow$ Modélisation du code et des fautes
- Approche implémentée dans un outil $\rightarrow$ **RobustB**
- 2 modèles de fautes supportés
  - Saut d'instruction
  - Corruption de registre

Vue d'ensemble de la méthode d'analyse



- 1 - **Extraction** d'une représentation de la **zone à analyser**
- 2 - **Détermination** des **chemins d'exécution** possibles
- 3 - **Injection de faute** simple (corruption de registre, saut d'instruction) sur les chemins d'exécution possibles
- 4 - **Recherche de vulnérabilités** par vérification formelle d'une propriété de non-équivalence (SMT)
  - Satisfiable $\rightarrow$ la faute a induit une **vulnérabilité**
  - Insatisfiable $\rightarrow$ la faute n'a pas eu d'effet
- $\Rightarrow$ **Liste de vulnérabilités** avec leurs **localisations**

Extraction d'informations depuis le binaire



Analyse statique

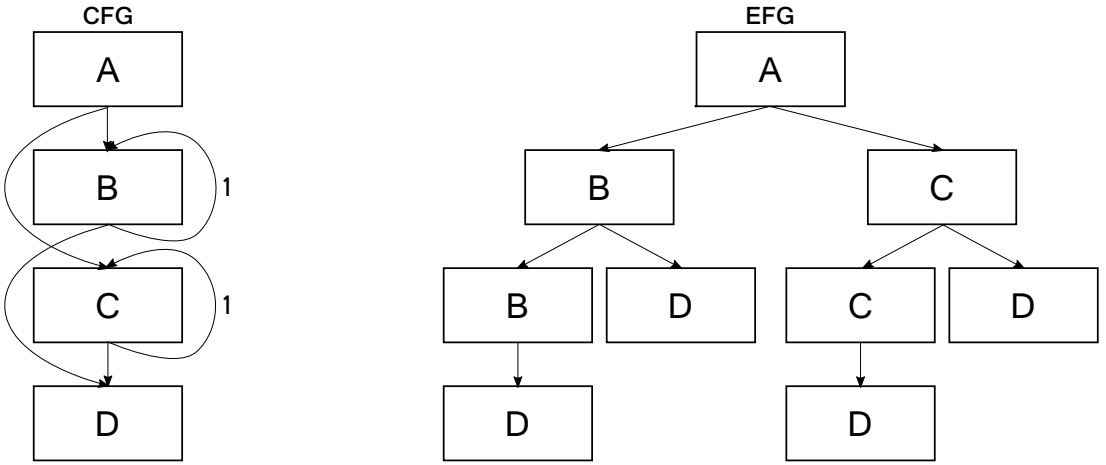
- Construction du CFG + ordre des blocs

Analyse dynamique/symbolique

- Extraction des contextes d'exécutions de la zone à analyser
- Extraction des bornes des boucles de la zone à analyser

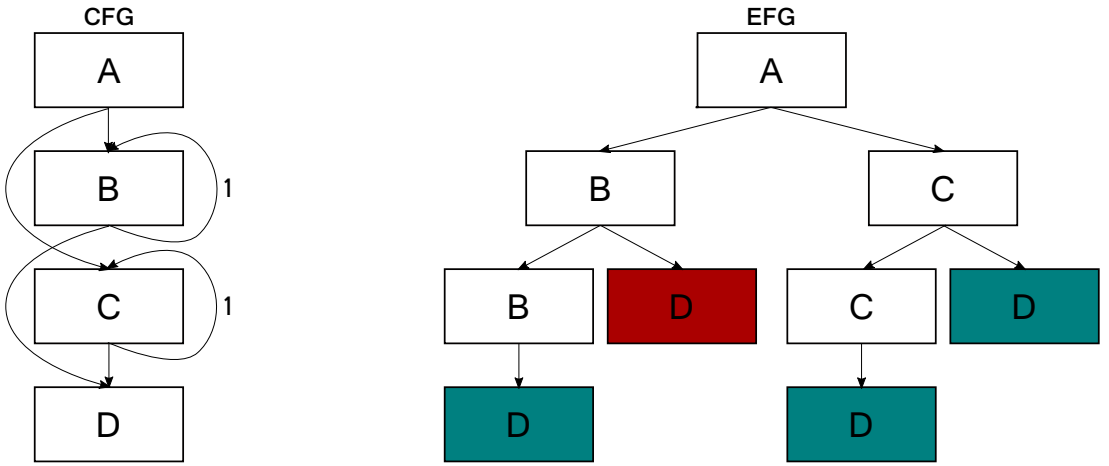
Détermination des chemins d'exécution possibles

- Déroulage structurel borné du CFG



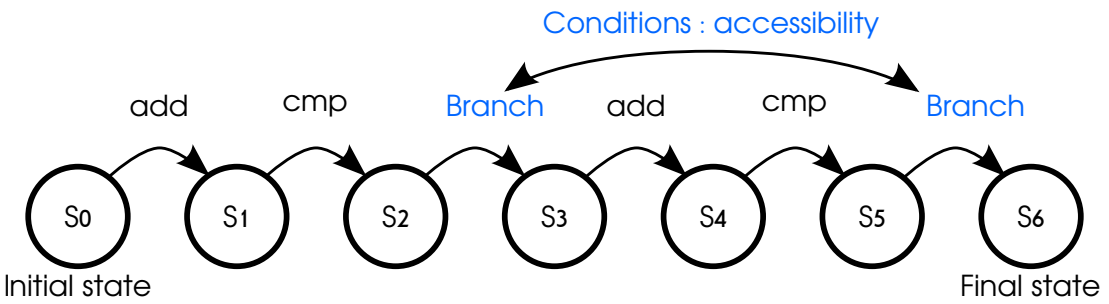
Détermination des chemins d'exécution possibles

- Déroulage structurel borné du CFG

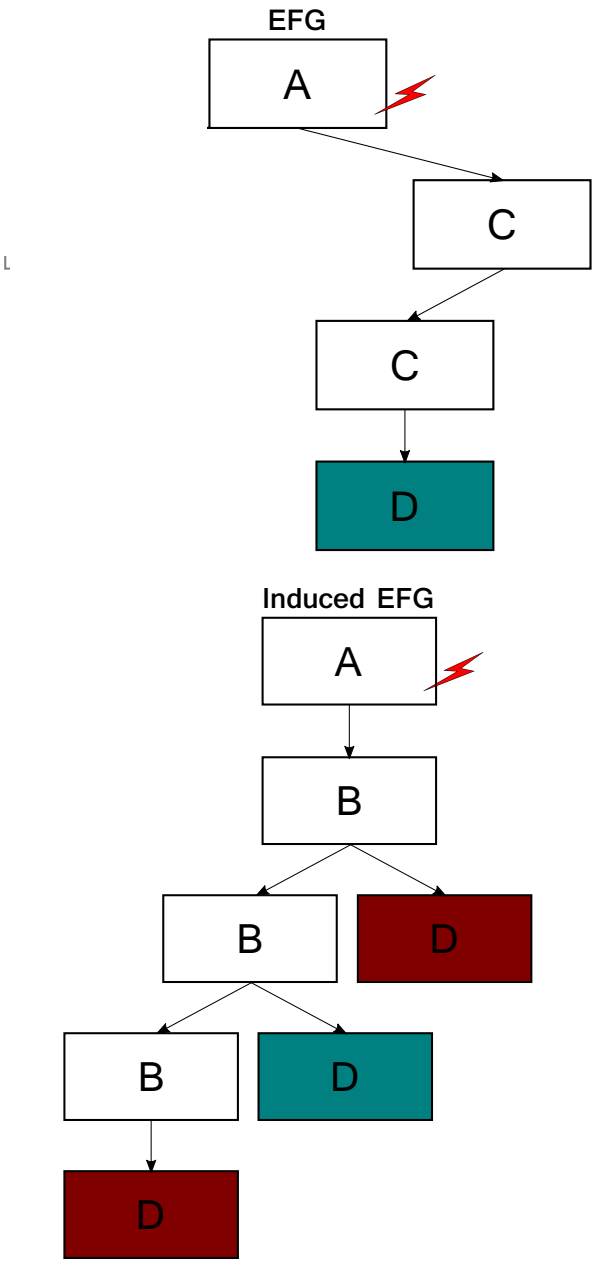


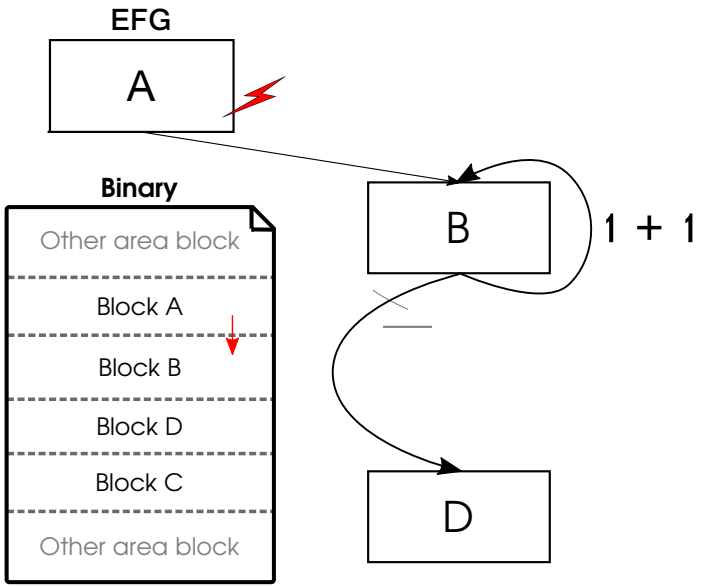
- Test de l'accessibilité des chemins d'exécution résultants (SMT) :

→ Modélisation de chaque instruction par leur effet sur une modélisation d'un état de la machine



Détermination des chemins d'exécution fautés

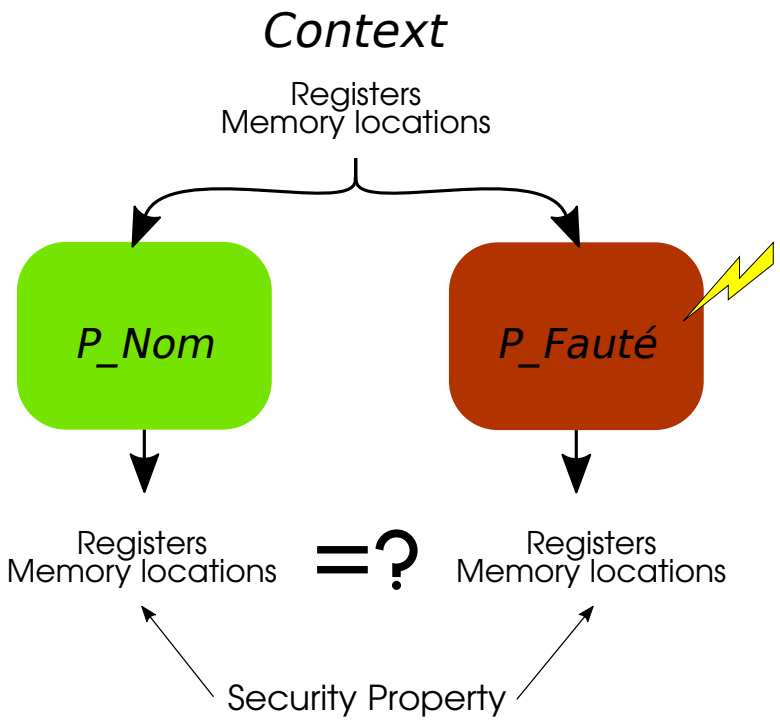




- Faute $\rightarrow$ déviation du chemin nominal
- Re-calcul des chemins d'exécution possibles après injection
- Déroulage du CFG après la faute :
  - Prise en compte du placement de code
  - Relâchement des contraintes de déroulage
- Accessibilité des chemins d'exécution calculés

Analyse de robustesse

- *P_Nom* $\rightarrow$ chemin d'exécution nominal
- *P_Fauté* $\rightarrow$ chemin d'exécution possible après une faute



- Même contexte de départ
- Une **vulnérabilité** est détectée si les **valeurs finales** d'éléments mémorisant **diffèrent** au point d'analyse
- Formule :

$$Access(P\_Nom) \wedge Access(P\_Fauté) \wedge Vuln$$

$\rightarrow$ **SAT** : La faute dans *P_Fauté* induit une vulnérabilité

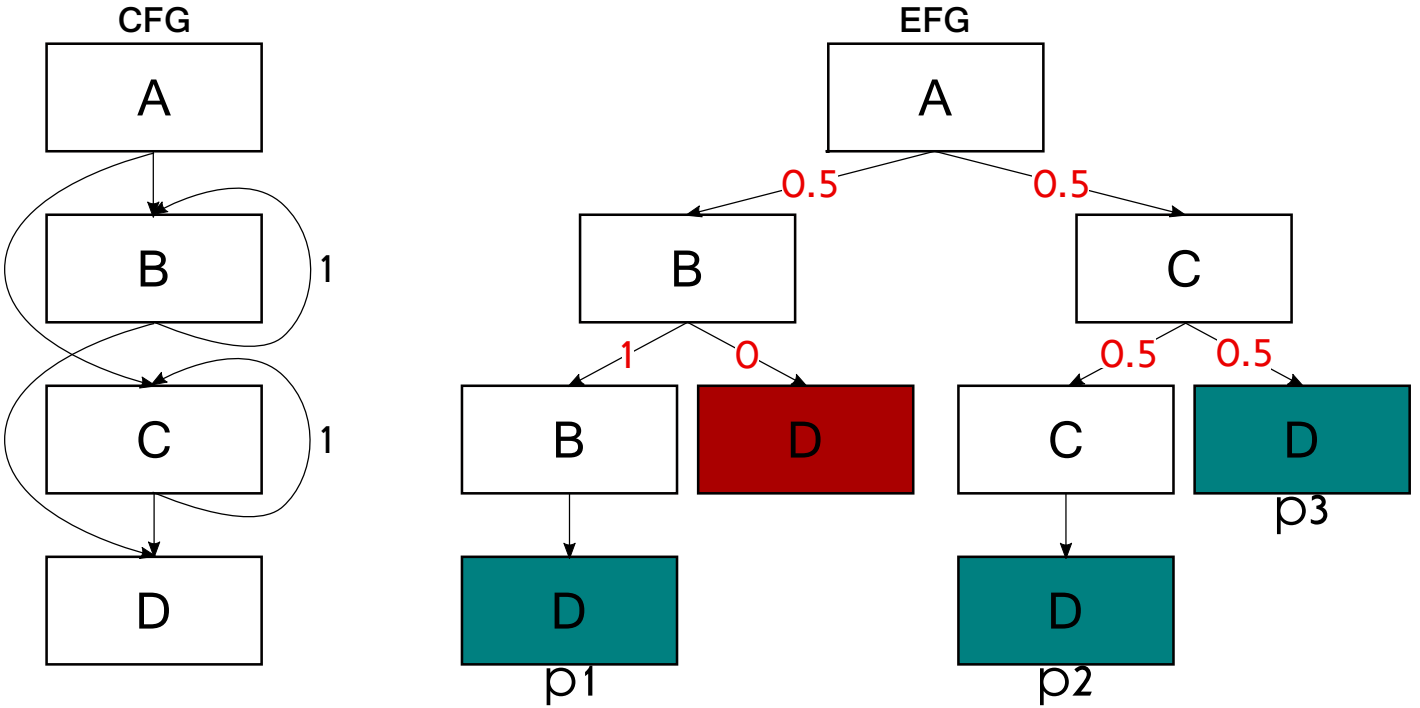
- Ex. de vulnérabilité : saut de l'**instruction @x** du **bloc y** sur le **chemin d'exécution z**
- La répétition de ce processus pour toutes les fautes sur les points d'injection produit une liste de vulnérabilités

Synthèse des résultats

- Les vulnérabilités sont difficiles à analyser
  - Niveau de dangerosité des vulnérabilités trouvées ?
  - Comment comparer les vulnérabilités de 2 codes similaires ?
- Besoin d'une vue synthétique
- Proposition de 3 métriques
  - **Degré de sensibilité d'une instruction**
  - **Score global du programme**
  - **Densité des vulnérabilités**

Déroulage du CFG et probabilités des chemins

Idée : une vulnérabilité apparaissant dans un chemin **rarement pris** est **moins importante** qu'une autre apparaissant dans un chemin **fréquemment pris**.



- Choix par défaut : équiprobabilité des chemins issus d'un branchement
- Idéalement : l'utilisateur définit les probabilités des branchements

Chemin	Blocs	P(chemin)
p1	A - B - B - D	0.5
p2	A - C - C - D	0.25
p3	A - C - D	0.25

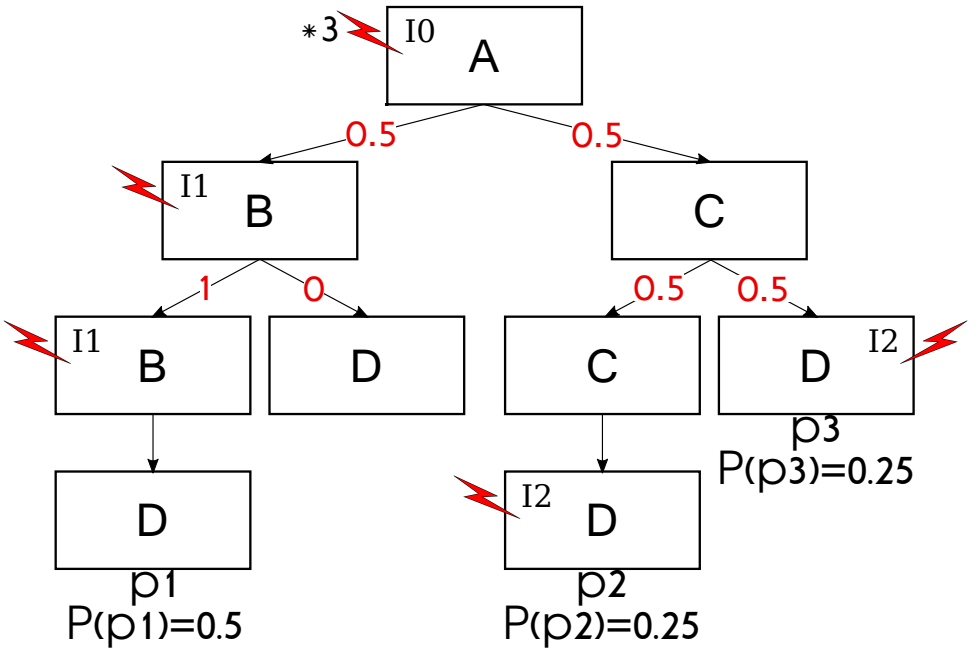
Sensibilité d'une instruction (IS)

IS(i) = score reflétant la sensibilité de l'instruction i

$$IS(i) = \sum_{p \in Paths} P(p \text{ is taken}) \times NV_i(p)$$

$NV_i(p)$ **: #Vulnerabilités pour l'instruction** i **sur le chemin** p

Inst	Score
`I0`	$\mathbf{1} = P(p1) + P(p2) + P(p3)$
`I1`	$\mathbf{1} = 2 * P(p1)$
`I2`	$\mathbf{0.5} = P(p2) + P(p3)$



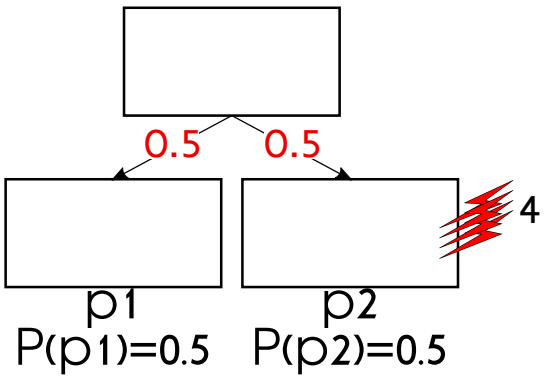
Permet l'identification des instructions à protéger en priorité

Surface d'attaque (AS)

AS = nombre moyen de vulnérabilités sur un chemin d'exécution.

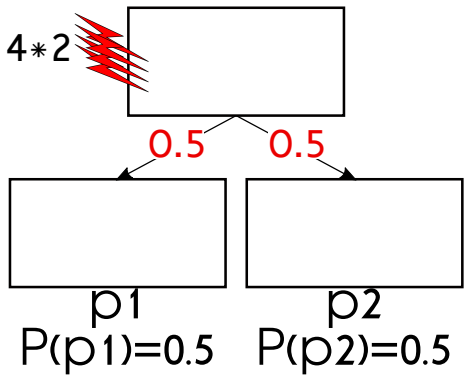
$$AS = \sum_{p \in Paths} P(p \text{ is taken}) \times NV(p)$$

NV(p)* : #Vulnerabilités sur le chemin *p



AS $= 4 * 0.5 =$ **2**

2 vulnérabilités en moyenne par chemin



AS $= 4 * 0.5 + 4 * 0.5 =$ **4**

4 vulnérabilités en moyenne par chemin

Plus la **surface d'attaque** est **élevée**, plus l'attaquant aura de **possibilités** d'injecter une faute aboutissant une **vulnérabilité**

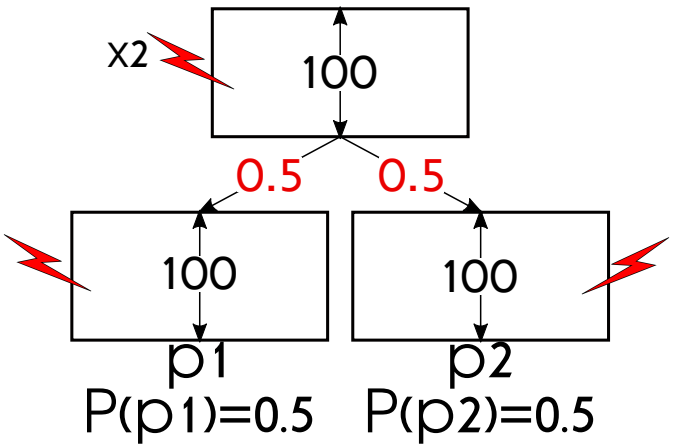
Surface d'attaque normalisée (NAS)

NAS = Densité moyenne des vulnérabilités

$$NAS = \frac{AS}{\sum_{p \in Paths} P(p \text{ is taken}) \times NI(p)} = \frac{AS}{ANI}$$

***NI(p)* : #Instructions sur le chemin** p

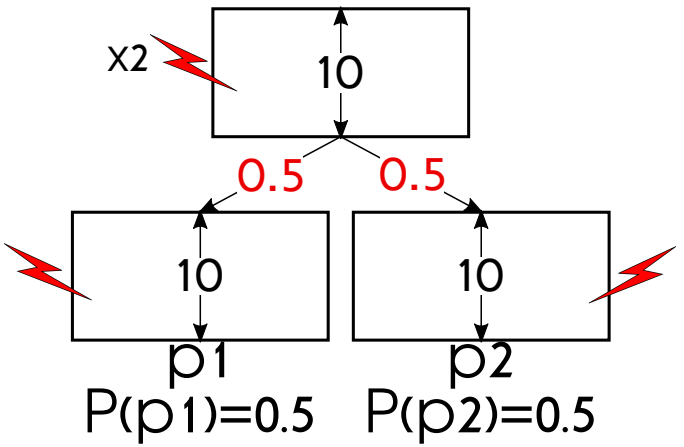
***ANI* : Nombre moyen d'instruction par chemin**



$AS = 2 * 0.5 + 2 * 0.5 = 2$

NAS $= 2/(100 + 100) =$ **0.01**

1% de chance qu'une injection de faute au hasard induise une vulnérabilité



$AS = 2 * 0.5 + 2 * 0.5 = 2$

NAS $= 2/(10 + 10) =$ **0.1**

10% de chance qu'une injection de faute au hasard induise une vulnérabilité

Use-case : VerifyPin

Description

- Appartient aux benchmarks **FISSC**(Fault Injection and Simulation Secure code Collection), dédiés à l'analyse d'injection de fautes
- Comparaison d'un PIN utilisateur avec un PIN prédéfini
  - Authentification "OK" si les PIN sont identiques, "KO" sinon
- Plusieurs versions du benchmark incluant différentes combinaisons de protections implémentées au niveau source

Analyse

- 4 versions : 1 version non-protégée, 3 versions protégées
- Deux niveaux d'optimisation, O0 et O2
- Modèle de faute : saut d'instruction
- Propriété : PIN entré faux et authentification "OK"

Résultats

- **Vulns** : nombre de vulnérabilités brutes
- **ANI** : nombre moyen d'instructions par chemin

Version	Opt level	#Vulns	AS	NAS	ANI
VerifyPin_0	O0	96	18.37	0.25	73.9
	O2	54	10.38	0.41	25.3
VerifyPin_4	O0	127	7.75	0.05	149.1
	O2	28	26	0.71	49
VerifyPin_5	O0	15	1	0.01	124.2
	O2	8	8	0.17	48
VerifyPin_7	O0	67	4.75	0.03	180.1
	O2	24	24	0.48	50

- Version la **moins vulnérable** (au sens des métriques AS et NAS) **VerifyPin**$_5$ O0
- Remarque : **NAS** est systématiquement plus élevée dans les versions O2 que dans les versions O0, contrairement au nombre de vulnérabilités brutes
- **VerifyPin**$_0$: surface d'attaque $\uparrow$ pour O0 mais densité des vulnérabilités $\uparrow$ pour O2
  $\rightarrow$ version O0 plus vulnérable face à un attaquant qui sait où attaquer
  $\rightarrow$ version O2 plus vulnérable face à un attaquant qui tire aléatoirement dans le temps

Résultats

- **Vulns** : nombre de vulnérabilités brutes
- **ANI** : nombre moyen d'instructions par chemin

Version	Opt level	#Vulns	AS	NAS	ANI
VerifyPin_0	O0	96	18.37	0.25	73.9
	O2	54	10.38	0.41	25.3
VerifyPin_4	O0	127	7.75	0.05	149.1
	O2	28	26	0.71	49
VerifyPin_5	O0	15	1	0.01	124.2
	O2	8	8	0.17	48
VerifyPin_7	O0	67	4.75	0.03	180.1
	O2	24	24	0.48	50

- Version la **moins vulnérable** (au sens des métriques AS et NAS) **VerifyPin**$_5$ O0
- Remarque : **NAS** est systématiquement plus élevée dans les versions O2 que dans les versions O0, contrairement au nombre de vulnérabilités brutes
- **VerifyPin**$_0$: surface d'attaque $\uparrow$ pour O0 mais densité des vulnérabilités $\uparrow$ pour O2
  $\rightarrow$ version O0 plus vulnérable face à un attaquant qui sait où attaquer
  $\rightarrow$ version O2 plus vulnérable face à un attaquant qui tire aléatoirement dans le temps

Résultats

- **Vulns** : nombre de vulnérabilités brutes
- **ANI** : nombre moyen d'instructions par chemin

Version	Opt level	#Vulns	AS	NAS	ANI
VerifyPin_0	O0	96	18.37	0.25	73.9
	O2	54	10.38	0.41	25.3
VerifyPin_4	O0	127	7.75	0.05	149.1
	O2	28	26	0.71	49
VerifyPin_5	O0	15	1	0.01	124.2
	O2	8	8	0.17	48
VerifyPin_7	O0	67	4.75	0.03	180.1
	O2	24	24	0.48	50

- Version la **moins vulnérable** (au sens des métriques AS et NAS) **VerifyPin**$_5$ O0
- Remarque : **NAS** est systématiquement plus élevée dans les versions O2 que dans les versions O0, contrairement au nombre de vulnérabilités brutes
- **VerifyPin**$_0$: surface d'attaque $\uparrow$ pour O0 mais densité des vulnérabilités $\uparrow$ pour O2
  $\rightarrow$ version O0 plus vulnérable face à un attaquant qui sait où attaquer
  $\rightarrow$ version O2 plus vulnérable face à un attaquant qui tire aléatoirement dans le temps

Résultats

- **Vulns** : nombre de vulnérabilités brutes
- **ANI** : nombre moyen d'instructions par chemin

Version	Opt level	#Vulns	AS	NAS	ANI
VerifyPin_0	O0	96	18.37	0.25	73.9
	O2	54	10.38	0.41	25.3
VerifyPin_4	O0	127	7.75	0.05	149.1
	O2	28	26	0.71	49
VerifyPin_5	O0	15	1	0.01	124.2
	O2	8	8	0.17	48
VerifyPin_7	O0	67	4.75	0.03	180.1
	O2	24	24	0.48	50

- Version la **moins vulnérable** (au sens des métriques AS et NAS) **VerifyPin**$_5$ O0
- Remarque : **NAS** est systématiquement plus élevée dans les versions O2 que dans les versions O0, contrairement au nombre de vulnérabilités brutes
- **VerifyPin**$_0$: surface d'attaque $\uparrow$ pour O0 mais densité des vulnérabilités $\uparrow$ pour O2
  $\rightarrow$ version O0 plus vulnérable face à un attaquant qui sait où attaquer
  $\rightarrow$ version O2 plus vulnérable face à un attaquant qui tire aléatoirement dans le temps

Résultats

- **Vulns** : nombre de vulnérabilités brutes
- **ANI** : nombre moyen d'instructions par chemin

Version	Opt level	#Vulns	AS	NAS	ANI
VerifyPin_0	O0	96	18.37	0.25	73.9
	O2	54	10.38	0.41	25.3
VerifyPin_4	O0	127	7.75	0.05	149.1
	O2	28	26	0.71	49
VerifyPin_5	O0	15	1	0.01	124.2
	O2	8	8	0.17	48
VerifyPin_7	O0	67	4.75	0.03	180.1
	O2	24	24	0.48	50

- Version la **moins vulnérable** (au sens des métriques AS et NAS) **VerifyPin_5** O0
- Remarque : **NAS** est systématiquement plus élevée dans les versions O2 que dans les versions O0, contrairement au nombre de vulnérabilités brutes
- **VerifyPin_0** : surface d'attaque $\uparrow$ pour O0 mais densité des vulnérabilités $\uparrow$ pour O2
  $\rightarrow$ version O0 plus vulnérable face à un attaquant qui sait où attaquer
  $\rightarrow$ version O2 plus vulnérable face à un attaquant qui tire aléatoirement dans le temps

Résultats

- **Vulns** : nombre de vulnérabilités brutes
- **ANI** : nombre moyen d'instructions par chemin

Version	Opt level	#Vulns	AS	NAS	ANI
VerifyPin_0	O0	96	18.37	0.25	73.9
	O2	54	10.38	0.41	25.3
VerifyPin_4	O0	127	7.75	0.05	149.1
	O2	28	26	0.71	49
VerifyPin_5	O0	15	1	0.01	124.2
	O2	8	8	0.17	48
VerifyPin_7	O0	67	4.75	0.03	180.1
	O2	24	24	0.48	50

- Version la **moins vulnérable** (au sens des métriques AS et NAS) **VerifyPin**$_5$ O0
- Remarque : **NAS** est systématiquement plus élevée dans les versions O2 que dans les versions O0, contrairement au nombre de vulnérabilités brutes
- **VerifyPin**$_0$: surface d'attaque $\uparrow$ pour O0 mais densité des vulnérabilités $\uparrow$ pour O2
  $\rightarrow$ version O0 plus vulnérable face à un attaquant qui sait où attaquer
  $\rightarrow$ version O2 plus vulnérable face à un attaquant qui tire aléatoirement dans le temps

Résultats

- **Vulns** : nombre de vulnérabilités brutes
- **ANI** : nombre moyen d'instructions par chemin

Version	Opt level	#Vulns	AS	NAS	ANI
VerifyPin_0	O0	96	18.37	0.25	73.9
	O2	54	10.38	0.41	25.3
VerifyPin_4	O0	127	7.75	0.05	149.1
	O2	28	26	0.71	49
VerifyPin_5	O0	15	1	0.01	124.2
	O2	8	8	0.17	48
VerifyPin_7	O0	67	4.75	0.03	180.1
	O2	24	24	0.48	50

- Version la **moins vulnérable** (au sens des métriques AS et NAS) **VerifyPin**$_5$ O0
- Remarque : **NAS** est systématiquement plus élevée dans les versions O2 que dans les versions O0, contrairement au nombre de vulnérabilités brutes
- **VerifyPin**$_0$: surface d'attaque $\uparrow$ pour O0 mais densité des vulnérabilités $\uparrow$ pour O2
  $\rightarrow$ version O0 plus vulnérable face à un attaquant qui sait où attaquer
  $\rightarrow$ version O2 plus vulnérable face à un attaquant qui tire aléatoirement dans le temps

Résultats

- **Vulns** : nombre de vulnérabilités brutes
- **ANI** : nombre moyen d'instructions par chemin

Version	Opt level	#Vulns	AS	NAS	ANI
VerifyPin_0	O0	96	18.37	0.25	73.9
	O2	54	10.38	0.41	25.3
VerifyPin_4	O0	127	7.75	0.05	149.1
	O2	28	26	0.71	49
VerifyPin_5	O0	15	1	0.01	124.2
	O2	8	8	0.17	48
VerifyPin_7	O0	67	4.75	0.03	180.1
	O2	24	24	0.48	50

- Version la **moins vulnérable** (au sens des métriques AS et NAS) **VerifyPin**$_5$ O0
- Remarque : **NAS** est systématiquement plus élevée dans les versions O2 que dans les versions O0, contrairement au nombre de vulnérabilités brutes
- **VerifyPin**$_0$: surface d'attaque $\uparrow$ pour O0 mais densité des vulnérabilités $\uparrow$ pour O2
  $\rightarrow$ version O0 plus vulnérable face à un attaquant qui sait où attaquer
  $\rightarrow$ version O2 plus vulnérable face à un attaquant qui tire aléatoirement dans le temps

Autres analyses réalisées

Différents cas d'utilisation

- Algorithme PRESENT $\rightarrow$ placement de code
- Code système FreeRTOS
- Protection de boucle $\rightarrow$ memcpy
  - protégé à la compilation
  - protégé au niveau du code source

AES sécurisé avec le compilateur PROSECCO (en cours)

Performance de l'outil

- Parallèlisation $\rightarrow$ les problèmes SMT sont lancés en parallèle
- Analyses réalisées sur une machine de 20 processeurs Intel Xeon Silver 4114 (2.20GHz), Hyper-threading $\times 2$

VerifyPin

- VerifyPin_0 O2 $\rightarrow$ 3m
- VerifyPin_7 O0 $\rightarrow$ 13h

$\rightarrow$ taille du code, nombre d'accès mémoire, nombre de chemins d'exécution

Schéma de protection de boucle appliqué à un memcpy

- ~1000 corruptions de registre $\rightarrow$ 1h20
- ~700 saut d'instruction $\rightarrow$ 20m

$\rightarrow$ Le nombre de variables libres influe sur la rapidité de l'analyse

Conclusion

- Un outil permettant d'analyser des portions de code binaire contre des fautes simples

Avantages

- Automatique (sauf pour la propriété $\rightarrow$ travaux de Vu Son Tuan)
- Vérification formelle
- Analyse en contexte
- SMT $\rightarrow$ exhaustivité (corruption de registres)

Limitations

- Petites portions de code
- Multiples fautes
- La rapidité de l'analyse dépend du nombre de chemins possibles ainsi que du nombre d'accès mémoire

Merci de votre attention !