IX ENDICT
Encontro de iniciação científica
Campus Toledo
2023

Gerador de Chaves Presentes em Instruções de Máquina por Teoria de Satisfabilidade do Módulo

José Luis Bastos Donin[1]
donin@alunos.utfpr.edu.com
Orientador:
Daniel Cavalcanti Jeronymo

[1]*Universidade Tecnológica Federal do Paraná – Campus Toledo*
Programa de IC

Introdução

O objetivo deste projeto é aplicar as Teorias de Módulos de Satisfatibilidade (SMT) à geração automática de chaves para acesso a sistemas através de análise binária. O objetivo é o desenvolvimento de um sistema que automatize o processo de engenharia reversa através do Binary Analysis and Reverse Engineering Framework (BARF) que traduz as instruções x86 para uma linguagem intermediária independente de plataforma chamada Reverse Engineering Intermediate Language (REIL). Z3-solver é um provador de teoremas criado pela Microsoft com algoritmos especializados para verificação e execução simbólica. O algoritmo realiza uma análise no arquivo executável .exe no Windows ou arquivo elf no Linux fornecido pelo usuário sem o código-fonte, o usuário deve especificar os pontos de entrada e saída dos endereços de memória a serem analisados pelo framework, as instruções entre esses pontos são traduzidas das instruções de assembly para a linguagem intermediária

Função de validação em C

```
int valida(int valCode[5]) {
    int sum, i;

        sum = valCode[0] + valCode[1] + valCode[2] + valCode[3]
+ valCode[4] ;


        if(sum %2 == 0){
            printf("\nWelcome!");
            return 1;
        }
        else{
            printf("\nInvalid Key!");
            printf("\nsum: %d\n", sum);
        }

        printf("\nValidation key: ");
        for( i=0; i<5;i++){
            printf(" %d", valCode[i]);
        }
        printf("\n");

        return 0;
}
```

Saída do programa em C
Validation key: 1 2 3 4 5
Invalid key!

Saída do script em Python após adicionar as restrições:
Key: 3 8 2 3 4

Saída do programa C com chave z3-solver:
Validation key: 3 8 2 3 4
Welcome!

Script Z3-Solver em Python

```
solver.add(((digit1)+(digit2)) %2 != 0 )
solver.add(((digit3)+(digit4)) %2 != 0 )
solver.add(((digit4)+(digit5)) %2 != 0 )
solver.add((digit1) != 1)
solver.add((digit2) != 1)
solver.add((digit3) != 1)
solver.add((digit4) != 1)
solver.add((digit5) != 1)
solver.add((digit1) != 0)
solver.add((digit2) != 0)
solver.add((digit3) != 0)
solver.add((digit4) != 0)
solver.add((digit5) != 0)

if solver.check() == sat:

    model = solver.model()

    digit_1 = model[digit1].as_long()
    digit_2 = model[digit2].as_long()
    digit_3 = model[digit3].as_long()
    digit_4 = model[digit4].as_long()
    digit_5 = model[digit5].as_long()


    print(f"Key: {digit_1} {digit_2} {digit_3}
    {digit_4} {digit_5}").
```

Conclusão

De acordo com os resultados preliminares, é possível gerar uma chave de validação adicionando manualmente as restrições ao z3-solver, ele é capaz de gerar uma chave de validação desde que as restrições sejam satisfatórias. O próximo passo é aplicar esses resultados ao adaptar o script usando barf e openreil para automatizar o processo de geração de chaves pela análise das instruções assembly traduzidas para openreil. Os principais desafios enfrentados atualmente são a falta de compatibilidade com as versões atuais do python, pois é necessário adaptar as tecnologias disponíveis para versões atualizadas.

Referências

1 ARCE, C. H. A. BARF: A multiplatform open source Binary Analysis and Reverse engineering Framework. [s.d.].

2 BJØRNER, N. et al. Programming Z3. Em: Engineering Trustworthy Software Systems. Cham: Springer International Publishing, 2019. p. 148–201.

3 REIL: A platform-independent intermediate representation of disassembled code for static code analysis Thomas Dullien. [s.l: s.n.].

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Logos/apoios: