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

Key Generator by Machine Code through Satisfiability Modulo Theories

José Luis Bastos Donin[1], Daniel Cavalcanti Jeronymo[2]

RESUMO

Neste projeto as Teorias de Satisfatibilidade Módulo (SMT) foram utilizadas para gerar automaticamente chaves de acesso através da análise de arquivos binários. Foi empregado o Framework de Análise e Engenharia Reversa Binária (BARF) para traduzir instruções de máquina da plataforma Intel x86 para a Linguagem Intermediária de Engenharia Reversa (REIL). As instruções traduzidas são analisadas pelo Z3-solver para determinar a satisfatibilidade das restrições. Quando satisfatórias, são geradas possíveis atribuições para as variáveis 'a', 'b' e 'c'. Os resultados preliminares indicam a possibilidade de gerar uma chave de validação adicionando manualmente restrições ao Z3-solver. Trabalhos futuros focam na automatização da geração de chaves através da análise de instruções traduzidas em REIL. Continua-se com a pesquisa em andamento, objetivando resolver desafios atuais que incluem problemas de compatibilidade com versões mais recentes do Python, necessitando adaptação tecnológica para versões atualizadas.

PALAVRAS-CHAVE: análise automática; assembly; keygen.

ABSTRACT

This project uses Satisfiability Modulo Theories (SMT) to automatically generate access keys through binary file analysis. The Binary Analysis and Reverse Engineering Framework (BARF) was used to translate machine instructions from the Intel x86 platform into a platform-independent intermediate language, Reverse Engineering Intermediate Language (REIL). The translated instructions are analyzed by the Z3-solver to determine the satisfiability of constraints. If satisfactory, it prints possible assignments for the variables ‘a’, ‘b’, and ‘c’. Preliminary results indicate the possibility of generating a validation key by manually adding constraints to the Z3-solver. Future work focuses on automating key generation through the analysis of instructions translated into REIL. Research is ongoing and current challenges include compatibility issues with newer versions of Python, necessitating technological adaptation to updated versions.

KEYWORDS: automatic analysis; assembly; keygen.

[1] Universidade Tecnológica Federal do Paraná, UTFPR, Toledo, PR, Brasil. E-mail: donin@alunos.utfpr.edu.br.
[2] Docente na Graduação em Engenharia de Computação. Universidade Tecnológica Federal do Paraná, UTFPR, Toledo, PR, Brasil. E-mail: danielc@utfpr.edu.br. ID Lattes: http://lattes.cnpq.br/5277068780093822.

Introdução

Uma chave é um conjunto de informação que determina a saída funcional de um algoritmo. As chaves são tipicamente utilizadas em diferentes contextos computacionais como forma de autenticação de acesso a recursos. O sistema de acesso, incluindo seu algoritmo, é tipicamente conhecido. Entretanto, a chave é mantida em segredo por quem tem acesso ao recurso (Kerckhoffs, 1883), a fim de validar sua autenticação.

Os algoritmos utilizados para implementar estes sistemas de acesso são tipicamente programados em linguagens compiladas, onde o código é transformado em instruções de máquina. A operação de compilação é representada pelo mapeamento da função $t : C \rightarrow I$, onde *C* é o conjunto de instruções da linguagem de programação e *I* é o conjunto das instruções de máquina de alguma arquitetura. A função *t* é não-injetiva, pelo fato de que múltiplas estruturas da linguagem de programação são mapeadas para os mesmos conjuntos de instruções de máquina. A característica não-injetiva da função t, especialmente quando associada a algoritmos que otimizam o processo de compilação, origina perdas de informação entre a linguagem de programação e as instruções de máquina. Isto ocorre pelo fato de que diversas características sintáticas existentes em linguagens de programação não tem correlato com instruções de máquina. Portanto, o processo de decompilação, isto é a função inversa t^{-1}, onde obtêm-se o código a partir das instruções de máquina, é difícil de ser realizado, senão impossível em sua completude (Cifuentes e Gough, 1995) e (Kraft, Malloy e Power, 2007).

Uma das práticas da engenharia reversa, no contexto da engenharia de software, é realizar o mapeamento inverso entre instruções de máquina e código. Uma das aplicações dessa prática é a criação de geradores de chaves, *key generators* ou *keygens*. Para isto, é identificado no arquivo binário a localização das instruções de máquina que implementam o algoritmo de autenticação. Este algoritmo pode ser considerado como uma função mapeadora $a : K \rightarrow \{0, 1\}$, onde *K* é a chave de acesso, um segredo do usuário, e a saída é um valor binário que representa o acesso negado ou permitido. O desafio então é encontrar o algoritmo inverso, portanto a função inversa a^{-1}, que identifique a chave correta *K* para acesso ao recurso. Este problema pode ser reescrito como encontrar as raízes da função de acesso tal que sua saída seja o acesso permitido, isto é, encontrar as raízes de $a(K) = 1$.

A teoria da satisfabilidade do módulo (*satisfability modulo theories* – SMT) decide a satisfabilidade de fórmulas pela combinação de diferentes teorias de fundamentação, a fim de encontrar a estrutura que satisfaça as sentenças e fórmulas de algum problema (Campbell e Stark, 2016; Saarikivi e Heljanko, 2015). A SMT é uma forma do problema de satisfação de restrições que poderia ser aplicada ao problema de encontrar as raízes da função de acesso.

A engenharia reversa de arquivos binários, que contêm instruções de máquina para execução da CPU, é um desafio devido à sua resistência à tradução direta para seu código de programação original. Este artigo apresenta um projeto que visa superar esse desafio através da utilização das Teorias de Satisfatibilidade Módulo (SMT) para a

geração automática de chaves de acesso a sistemas por meio da análise desses arquivos binários.

A proposta deste trabalho é automatizar o processo de engenharia reversa, traduzindo instruções de máquina da plataforma Intel x86 para uma linguagem intermediária independente de plataforma, a Linguagem Intermediária de Engenharia Reversa (REIL), através do uso do Framework de Análise e Engenharia Reversa Binária (BARF).

O Z3-solver, um provador de teoremas desenvolvido pela Microsoft, é empregado para verificar simbolicamente as instruções traduzidas e executá-las, determinando a satisfatibilidade das restrições impostas. O usuário fornece um arquivo executável PE32 (Windows) ou ELF (Linux), especifica os pontos de entrada e saída dos endereços de memória a serem analisados, e as instruções entre esses pontos são traduzidas para REIL via BARF.

Métodos

A metodologia proposta é um processo detalhado que envolve várias etapas. Primeiro, o Framework de Análise e Engenharia Reversa Binária (BARF) é utilizado. Este framework é uma ferramenta que permite a análise de binários em um nível detalhado. Ele é capaz de traduzir instruções de máquina específicas da plataforma x86 para uma linguagem intermediária de plataforma independente, chamada Linguagem Intermediária de Engenharia Reversa (REIL). Esta tradução permite uma análise detalhada do código binário, pois a REIL é uma linguagem que facilita a análise das instruções de máquina originais.

O processo começa importando o módulo BARF. Este módulo contém todas as funções e classes necessárias para a análise do binário. Em seguida, o usuário fornece um arquivo executável PE32 (Windows) ou arquivo ELF chamado “main”, este arquivo executável possui uma função que determina se uma chave digitada pelo usuário é válida. O código-fonte é desconhecido para o usuário, que só tem acesso às instruções assembly após a compilação do arquivo “main.c” que gera o arquivo executável.

Um arquivo ELF (Executable and Linkable Format), é um padrão comum para arquivos executáveis. Este formato foi desenvolvido pela Unix System Laboratories e é amplamente utilizado em sistemas operacionais Unix, como GNU-Linux. O formato ELF define a estrutura que os arquivos binários devem seguir para que sejam interpretados pelo sistema operacional.

Um arquivo executável PE32 é um arquivo de um programa executável para sistemas operacionais Windows, é uma variação do formato Executável Portátil ou Portable Executable em inglês, possui uma estrutura de dados que carrega as informações necessárias para o gerenciador do Windows executar o programa.

As instruções são carregadas pelo analisador de código a partir de um intervalo específico no arquivo ELF ou PE32. Este intervalo é escolhido com base nos requisitos específicos da análise. A expressão do registrador para “ebp” é obtida no modo “post”. O

registrador “ebp” é comumente usado em programas x86 para apontar para a base da pilha de chamadas atual.

Em seguida, as pré-condições são definidas em um intervalo para as variáveis ‘a’ e ‘b’, que são expressões de memória obtidas a partir de locais específicos em relação a ‘ebp’. Essas pré-condições garantem que ‘a’ e ‘b’ estejam dentro do intervalo de 2 a 100. As pós-condições são definidas com o valor desejado para a variável ‘c’, que é outra expressão de memória. Esta etapa é muito importante para garantir que os valores das variáveis estejam no limite esperado.

O Z3-solver é uma ferramenta de verificação simbólica, usado para determinar a satisfatibilidade de um conjunto de restrições, ou seja, se existe uma solução que satisfaça todas as restrições ao mesmo tempo.

Nesse contexto, o Z3-solver é empregado após a tradução das instruções de máquina para a Linguagem Intermediária de Engenharia Reversa (REIL) pelo Framework de Análise e Engenharia Reversa Binária (BARF). As instruções são verificadas simbolicamente pelo Z3-solver após traduzidas. O Z3-solver executa as instruções em um nível simbólico, não em valores concretos, permitindo uma análise genérica.

As restrições impostas no código são verificadas pelo Z3-solver. Essas restrições podem ser pré-condições e pós-condições que definimos no código, são usadas para especificar as propriedades que devem ser verdadeiras antes e depois da execução do trecho do programa, os resultados esperados e os estados finais que o código deve se encontrar após os testes com as restrições estipuladas. O Z3-solver verifica se existe uma solução que satisfaça todas estas restrições. Se tal solução existir, dizemos que as restrições são satisfatórias.

Por exemplo, se tivermos uma pré-condição que define que uma variável ‘x’ deve ser maior que 2, e uma pós-condição que define que o resultado ‘y’ deve ser menor que 10, o Z3-solver irá determinar se existe algum valor de ‘x’ e ‘y’ que satisfaça ambas as condições.

Em resumo, o Z3-solver é uma ferramenta que nos permite verificar a validade das restrições e proporciona uma maneira eficaz de análise do código binário, permitindo que os usuários entendam o comportamento do código e identifiquem possíveis problemas ou vulnerabilidades.

Após a análise realizada pelo z3-solver, caso haja a satisfabilidade das restrições estabelecidas, a chave válida para o programa é imprimida no terminal para que o usuário possa utilizar no programa “main”.

Conclusão

Em testes iniciais, foi possível gerar uma chave válida para um programa C sem o código fonte original ao adicionar manualmente restrições ao Z3-solver. Os resultados

preliminares indicam a viabilidade da geração automática de chaves válidas desde que as restrições impostas sejam satisfatórias.

O trabalho futuro se concentrará na aplicação desses resultados e na adaptação da abordagem proposta para automatizar a geração de chaves por meio da análise de instruções traduzidas em REIL. Desafios atuais incluem problemas de compatibilidade com versões mais recentes do Python, necessitando adaptação tecnológica para versões atualizadas.

Material suplementar

Este projeto utiliza o BARF Project para auxiliar na análise binária automática, fazendo uso de sua implementação do resolvedor Z3 e do Openreil. Todos os recursos necessários para a compreensão e replicação do projeto estão disponíveis para consulta e uso público nos respectivos repositórios do GitHub.

Agradecimentos

O primeiro autor agradece a UTFPR pela infraestrutura disponibilizada na forma de laboratórios para as conduções da pesquisa.

Disponibilidade de código

Todos os códigos pertinentes a este projeto estão disponíveis para consulta e uso público no seguinte repositório do GitHub: https://github.com/luisdonin/SMT-Automatic-Key-Generator. Este repositório contém todos os recursos necessários para a compreensão e replicação do projeto.

Conflito de interesse

Não há conflito de interesse.

REFERÊNCIAS

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

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

CAMPBELL, B., STARK, I. **Randomised testing of a microprocessor model using SMT-solver state generation, Science of Computer Programming, vol. 118, Mar. 2016.**

KERCKHOFFS, A. **"La cryptographie militaire", Journal des sciences militaires, vol. IX, pp. 5–83, Jan. 1883, pp. 161–191, Feb. 1883.**

KRAFT, N. A., MALLOY, B. A., POWER, J. F. **A tool chain for reverse engineering C++ applications, Science of Computer Programming, Volume 69, Issues 1–3, pp. 3 – 13, Dec. 2007.**

DULLIEN, T. **REIL: A platform-independent intermediate representation of disassembled code for static code analysis, 2016.**