Análise de Marcação e Fluxo de Controle para detectar Funcionalidades Escondidas em Instrumentos de Medição

Tiago M. do Nascimento[*†], Davidson R. Boccardo[*], Raphael C. S. Machado[*], e Luiz F.R.C Carmo[*]

[*]Inmetro – Instituto Nacional de Metrologia, Qualidade e Tecnologia
{tmnascimento, lfrust, drboccardo}@inmetro.gov.br
[†]UFRJ – Universidade Federal do Rio de Janeiro

***Resumo* — A crescente utilização de softwares embarcados em instrumentos de medição tem demandado novas abordagens de avaliação para validar sua confiabilidade. Uma ameaça a estes instrumentos é a presença de funcionalidades escondidas que possam ser acionadas para comprometer o funcionamento do instrumento de medição. Este trabalho propõe um método que detecta funcionalidades escondidas combinando técnicas de análise de programa: a análise do grafo de fluxo de controle, para verificar a presença de funcionalidades disjuntas ao programa principal, ou seja, uma funcionalidade escondida, e a análise de marcação que verifica se tais funcionalidades são realmente acionadas por meio das interfaces de entrada.**

Palavras-chave — análise de programas, engenharia reversa, avaliação e validação de instrumentos de medição.

I. Introdução

Antigamente os sistemas que possuíam softwares embarcados eram apenas utilizados em sistemas complexos como os sistemas industriais, aeronaves e navios. Hoje, porém, já são encontrados em instrumentos de medição para transações comerciais tais como medidores de energia elétrica, radares de velocidade, balança entre outros.

No entanto, ao se utilizar softwares embarcados em instrumentos de medição, a confiabilidade desses está intrinsecamente relacionada com a segurança do software nele embarcado. Assim esses instrumentos com software embarcado exigem novas abordagens de avaliação e validação. Esses instrumentos por serem sistemas cada vez mais sofisticados de hardware e software podem agregar um conjunto de novos desafios em seu processo de avaliação e validação.

Um risco associado à utilização de software embarcado em instrumentos de medição é a presença de funcionalidades escondidas que possam comprometer o correto funcionamento do instrumento. Estas funcionalidades escondidas normalmente são inseridas no instrumento de medição pelo fabricante para fins de depuração e teste. Contudo, muitas vezes tais funcionalidades são deixadas no software no momento da comercialização do instrumento de medição. Tal funcionalidade uma vez explorada pode permitir que um atacante subverta um mecanismo de controle de acesso, concedendo ao atacante total controle sobre o instrumento de medição [1].

Outra preocupação é quanto a presença de códigos maliciosos embarcados dentro do software ou que possam ser injetados por meio de vulnerabilidades contidas em funcionalidades escondidas [2]. Este trabalho propõe encontrar funcionalidades escondidas em softwares embarcados utilizando a combinação da análise de fluxo de controle com a análise de marcação. A análise de fluxo de controle verifica a presença de trechos de códigos disjuntos da aplicação principal e a análise de marcação verifica se tais trechos são realmente executados por meio de alguma interface de entrada do instrumento de medição. Assim, é possível verificar se valores passados ao instrumento de medição acionam as funcionalidades escondidas.

Na literatura é possível encontrar algumas ferramentas que fazem análise de programa combinando análise de marcação com fluxo de programas para detecção de funções vulneráveis [3], [4]. Porém essas ferramentas realizam tal identificação pela assinatura da função vulnerável, ou seja, sua sequencia de instruções ou bytes. Além disso, elas realizam análise de vulnerabilidades para a arquitetura x86 e não para softwares de arquitetura embarcada.

Nosso trabalho se difere destes, pois tem como objetivo detectar funcionalidades escondidas de software embarcado em instrumentos de medição independente de suas assinaturas. E também, nosso método proposto verifica se essas funcionalidades escondidas são ativadas por variáveis ou valores que vieram de interfaces de entrada do instrumento de medição.

Este artigo está organizado em três seções sendo a primeira seção esta que apresenta a introdução. A seção II apresenta a proposta com os conceitos básicos de análise de programas, análise de marcação e a proposta para detecção de códigos maliciosos e funcionalidades escondidas. Na seção III se encontram as conclusões.

II. PROPOSTA

Nesta seção serão vistos primeiramente os conceitos básicos de análise de programas, especificadamente análise de fluxo de controle e análise de marcação. Depois será detalhado como a combinação destas análises faz com que seja possível detectar funcionalidade escondidas embarcadas em um instrumento de medição.

A. Análise de Fluxo de Controle

A lógica do programa, seja ele na linguagem fonte ou binária, pode ser caracterizada por grafos de fluxo de controle e pelo grafo de chamadas.

Um grafo de fluxo de controle é uma representação de um procedimento do programa que usa notação de grafo direcionado para descrever todos os caminhos que podem ser percorridos por um software durante sua execução. Em um grafo de fluxo de controle cada nó representa um bloco básico, isto é, uma região de código sequencial sem qualquer salto de execução. Dessa forma, o destino de um salto denota o começo de um bloco, e cada aresta representa o fluxo de informação entre os blocos básicos baseados em saltos condicionais ou incondicionais. Na figura 1 é visualizado um exemplo de grafo de fluxo de controle juntamente com seu código fonte.

```
i = get_input();
two = 2;
if(i %2 == 0){
   j = i+two;
   l = j;
} else {
   k = two*two;
   l = k;
}
jmp l;
```

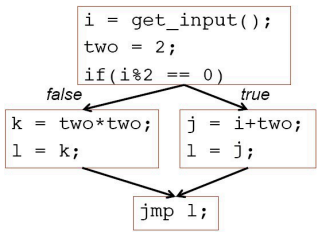

Fig.1. Código fonte e seu grafo de fluxo de controle.

Outro grafo importante é o grafo de chamadas. O grafo de chamadas é um grafo direcionado que representa as relações entre chamadas de funções ou sub-rotinas de um programa.

Para representação de um programa, em um grafo de chamadas, cada nó representa uma função e cada aresta que existir entre uma função 'f' e uma função 'g' indica que 'f' chama 'g'.

B. Análise de Marcação

A análise de marcação consiste em realizar marcações em variáveis ou constantes para fins de rastreamento. O uso dessa técnica permite monitorar valores de entrada suspeitos que possam por ventura executar alguma funcionalidade malicioso do software. A princípio, uma variável é marcada somente quando esta é influenciada por um valor de entrada, ou seja, o a saída dessa variável, ou seja seu valor, utiliza o valor de entrada em seu cálculo. Para rastreamento, toda variável que utiliza uma variável já marcada se torna marcada, e assim por diante. Com isso, é possível verificar se trechos de código são realmente acionados por um valor de entrada.

As Tabelas 1 e 2 exemplificam a análise de marcação dinâmica para o código da Figura 1. O código possui a variável de entrada *i*, pois esta recebe um valor de entrada através da função *get_input()*, assim esta variável é marcada. Em seguida, todas as variáveis que forem atribuídas o valor de *i* em seus cálculos, no caminho de execução do programa, também serão marcadas. Os resultados abaixo mostram a influência da entrada ser de valor par ou ímpar para a marcação das variáveis ao longo do grafo de fluxo de controle. Tabelas 1 e 2 mostram quais variáveis do programa são marcadas quando na variável *i* é atribuída um valor par ou ímpar, respectivamente..

Tab. 1. Análise com a variável *i* de valor par.

Variável	Valor	Marcada
i	6	Sim
two	2	Não
j	8	Sim
l	8	Sim

Tab. 2. Análise com a variável *i* de valor ímpar.

Variável	Valor	Marcada
i	7	Sim
two	2	Não
k	4	Não
l	4	Não

Dependendo do valor que *i* receba na entrada, o programa toma caminhos diferentes em seu fluxo de execução. Se em *i* for atribuído um valor par, o caminho de fluxo de execução será o da direita (ramo condicional verdadeiro) do programa, o que faz com que as variáveis *j* e *l* sejam marcadas, pois se utilizam da variável *i*. Em contrapartida, quando o valor atribuído a variável *i* for ímpar, o caminho do fluxo execução segue o ramo falso da instrução condicional, não marcando nenhuma das variáveis. Nota-se, contudo, que a instrução de salto incondicional '*jmp l*' pode transferir o fluxo de execução do programa para endereços de memória dependentes da

variável *i*, ou seja, a análise de marcação fornece indícios de que uma possível funcionalidade pode ser explorada utilizando uma entrada do usuário.

C. Detecção de Funcionalidades Escondidas

O método utilizado para encontrar funcionalidades escondidas foi baseado na combinação da análise do fluxo de controle com a análise de marcação. Para isso, foi programado um *script* escrito na linguagem *Python* para a ferramenta IDA [5], que é uma ferramenta de desmontagem (*disassembly*) de programas. A ferramenta IDA fornece uma interface com recursos para visualização dos grafos de fluxo de controle e do grafo de chamadas. Além disso, a ferramenta possui uma API [6] em que é possível interagir com as análises estáticas e dinâmicas já implementadas.

Então nosso método de combinação de análise, escrito em *Python*, é desenvolvido como uma extensão do *IDA* e utiliza uma máquina virtual de ARM [7] para depuração remota. Esta depuração remota será utilizada na análise de marcação. A arquitetura ARM foi escolhida pois instrumentos de medição se utilizam dela.

Quando se utiliza o método de combinação de análises em um programa, as variáveis de entrada deste programa são marcadas imediatamente. Em seguida o método continua a executar o programa dinamicamente rastreando essas variáveis marcadas para verificar se elas alteram o valor de outras variáveis. Ao alterar outras variáveis, estas passam a ser marcadas e rastreadas também.

Após isso, é possível observar o grafo de fluxo de controle com essas variáveis marcadas. Quando há blocos ou nós que não possuem arestas de ligações em um grafo de fluxo de controle, ou seja, blocos disjuntos, esses blocos são detectados como funcionalidades escondidas. Ao se visualizar um bloco disjunto, com suas variáveis marcadas, significa que nosso método identificou uma funcionalidade escondida. E, além disso, é possível determinar o caminho de execução e as instruções que realizam desvios até regiões disjunta do grafo de fluxo, para melhor entendimento de como são ativadas as funcionalidades escondidas.

Um exemplo de programa com funcionalidade escondida detectada pelo nosso método é visto na Figura 2.

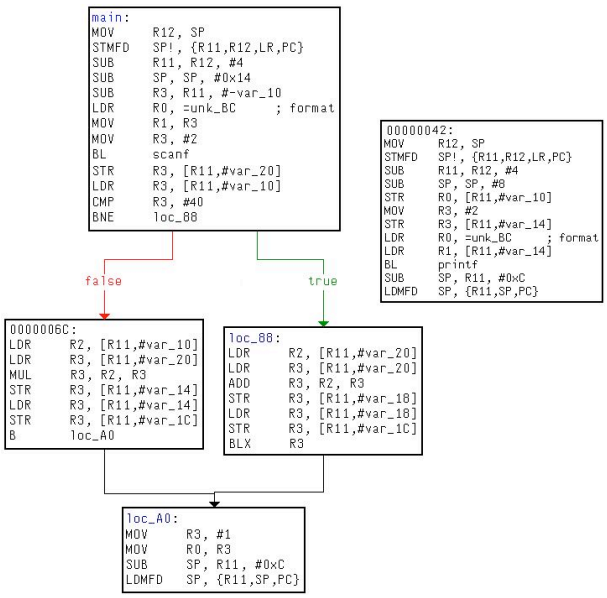

Fig. 2. Grafo de fluxo de controle com bloco disjunto.

Ao se observar o grafo de fluxo de controle de Figura 2, existe um bloco disjunto que não apresenta arestas de ligação, ou seja, não há um desvio direto para aquela região do programa. Este bloco é uma funcionalidade escondida.

Continuando a observar o grafo da Figura 2, pode-se ver que dentro do bloco *main* existe uma atribuição de valor ao registrador *R3* através de uma leitura pela interface de entrada. Se o valor de *R3* for igual a 40 o programa executa o bloco loc_88, se for diferente o programa executa o bloco *6C*. Dentro do bloco loc_88 pode-se encontrar a instrução *BLX R3* que realiza um desvio, em tempo de execução, para uma posição de memória que executa demais instruções, o bloco disjunto com endereço 42.

Sendo assim, o bloco disjunto 42 é uma funcionalidade escondida que é ativada através de um determinado valor, 40, atribuído ao registrador R3, ou seja, um valor passado como entrada do programa.

Portanto nosso método de combinação de análises desenvolvido neste trabalho mostra sua eficácia em detectar funcionalidades escondidas em um software embarcado em instrumento de medição.

III. CONCLUSÕES

Garantir a ausência de funcionalidade escondidas e ou vulneráveis em instrumentos de medição é imprescindível. Este trabalho incrementa o estado da arte combinando a análise de marcação já existente em outros trabalhos com a análise do fluxo de execução com o intuito de detectar funcionalidades escondidas que, por ventura, possam ser executadas diante de uma variável de entrada.

REFERÊNCIAS

[1] Smith, "DHS: Imported Tech Tainted with Backdoor Attack Tools", 2011, *http://www.networkworld.com/community/node/76262* (Último acesso março, 2013).

[2] The SANS Institute, "Glossary of Terms Used in Security and Intrusion Detection", 2009, *http://www.sans.org/security-resources/glossary-of-terms/* (Último acesso março, 2013).

[3] Ruoyu Zhang, Shiqiu Huang, Zhengwei Qi, Haibing Guan, "Static program analysis assisted dynamic taint tracking for software vulnerability discovery", *Computers and Mathematics with Applications* 63, p. 469–480, 2012.

[4] Newsome J., Song D., "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software", *Network and Distributed System Security Symposium*, NDSS, 2005.

[5] IDA, "IDA – multi-processor disassembler and debugger". *http://www.hex-rays.com/idapro/* (Último acesso março, 2013)

[6] IDAPython, "IDA Pro plugin that integrates the Python", *code.google.com/p/idapython* (Último acesso março, 2013).

[7] QEMU, "A generic and open source machine emulator and virtualizer", *www.qemu.org* (Último acesso março, 2013).