Uma Estratégia Dinâmica para a Detecção de Anomalias em Binários WebAssembly

Calebe Helpa[1], Tiago Heinrich[1], Marcus Botacin[2], Newton C. Will[3], Rafael R. Obelheiro[4], Carlos A. Maziero[1]

[1] Departamento de Informática
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

[2]Texas A&M University – Texas – USA

[3]Departamento de Ciência da Computação
Universidade Tecnológica Federal do Paraná (UTFPR)
Dois Vizinhos – PR – Brasil

[4]Programa de Pós-Graduação em Computação Aplicada (PPGCAP)
Universidade do Estado de Santa Catarina (UDESC)
Centro de Ciências Tecnológicas – Joinville – SC – Brasil

{cph19,theinrich,maziero}@inf.ufpr.br, botacin@tamu.edu,
will@utfpr.edu.br, rafael.obelheiro@udesc.br

Abstract. *WebAssembly is a low-level binary format that provides a compilation target for high-level languages. Offering more security for users on the web, with a binary instruction format, WebAssembly is supported by over 95% of web browsers. However, the growth in the use of WebAssembly has raised concerns regarding its security and possibility of malicious use. Given that WebAssemby is a low-level instruction format, it is essential to identify the purpose of the developed codes, by extracting their features. The use of WebAssembly for cryptojacking attacks and malicious code obfuscation is somewhat frequent. In this context, this work presents a strategy for identifying anomalies in WebAssembly binaries, through feature extraction and static analysis. The strategy proposed here achieved an f1score of 99.3%, highlighting its potential.*

Resumo. *WebAssembly é um formato binário de baixo nível, que oferece um alvo de compilação para linguagens de alto nível. Oferecendo mais segurança para os usuários na Web, com um formato de instruções binárias o WebAssembly é suportado por mais de 95% dos navegadores Web. No entanto, o crescimento no uso do WebAssembly trouxe preocupações em relação à sua segurança e seu possível uso de forma maliciosa. Dado que o WebAssemby é um formato de instruções de baixo nível, torna-se essencial a identificação do propósito dos códigos desenvolvidos, por meio da extração de suas características. O uso de WebAssembly para ataques de cryptojacking e ofuscação de códigos maliciosos é frequentemente observado. Nesse contexto, esse trabalho apresenta uma estratégia para a identificação de anomalias em binários WebAssembly, através de extração de características e análise estática. A estratégia proposta neste artigo alcançou um f1score de 99.3%, evidenciando seu potencial.*

1. Introdução

O constante desenvolvimento de aplicações web e a crescente necessidade de processamento mais rápido contribuiu para o desenvolvimento do formato de instruções *WebAssembly* [Romano et al. 2022]. Atualmente, a principal linguagem utilizada para desenvolver aplicações web é o JavaScript (JS), porém, devido ao aumento da quantidade e complexidade das aplicações executadas, mais desempenho e segurança se tornam necessários. Nesse contexto surge o *WebAssembly*, um formato de instruções em binário que executa dentro de um ambiente de *sandbox*.

O *WebAssembly* (também chamado *Wasm*) tem como objetivo dar suporte para as aplicações web, oferecendo um suporte de processamento rápido e diminuindo o uso de memória para carregar a páginas web [Falliere 2018]. Concomitantemente, o *WebAssembly* possui garantia de funcionamento em diferentes navegadores [Romano et al. 2022]. Por utilizar um ambiente virtualizado (*sandbox*) para executar os *bytecode WebAssembly*, as aplicações são portáteis, permitindo a reutilização de código em diferentes plataformas.

Contudo, o *WebAssembly* possui limitações de *design* que tornam o ambiente propício a certos tipos de ataques. Por ser uma tecnologia recente, ainda existem vulnerabilidades de segurança e espaço para melhorias a serem explorados. Por exemplo, problemas como *buffer overflow* estão presentes em casos de interação entre JS e *WebAssembly* [Michael et al. 2022]. Por conta da utilização de um *buffer* implementado por meio de um vetor para ser utilizado como memória do programa, problemas relacionados ao armazenamento de dados da aplicação também estão presentes [Lehmann et al. 2020].

O *WebAssembly* também possui potencial para uso malicioso devido ao formato binário, que dificulta a identificação do proposito de cada programa. Desta forma, usuários maliciosos exploram o *design* do formato para realizar ataques *cryptojacking* e a ofuscação de códigos maliciosos [Naseem et al. 2021]. A fim de mitigar tais riscos, estudos recentes focam em corrigir falhas de *design*, adicionar novos recursos e melhorar recursos já implantados, avaliando os problemas relacionados à memória [Michael et al. 2022] e aperfeiçoando o *design* do compilador [Bosamiya et al. 2022].

Considerando a segurança de aplicações *WebAssembly* no ambiente web, um processo de detecção de anomalias para binários *WebAssembly* permitiria a identificação de vulnerabilidades e conteúdos maliciosos que possam estar presentes nos binários. Estratégias de Machine Learning (ML) possuem resultados promissores na área de segurança, permitindo a classificação de comportamentos e identificação de anomalias [Ceschin et al. 2020].

O objetivo deste trabalho é estudar o potencial de informações extraídas de binários *WebAssembly* para a identificação de anomalias, como vulnerabilidades do formato exploradas para realização de ataques ou erros inseridos durante o desenvolvimento. Para alcançar este objetivo, foi utilizado o formato Debugging With Attributed Record Format (DWARF), permitindo a utilização de um processo estático para a extração de informações dos binários *WebAssembly*. Posteriormente, para averiguar o potencial das informações extraídas em um processo de detecção de anomalias, algoritmos de aprendizado de máquina que realizam a classificação dos binários foram treinados utilizando estes dados. As contribuições do trabalho consistem de:

- Definição de características que sejam representativas dos binários *WebAssembly*

para realizar a detecção de anomalias; e
- Um modelo de detecção de anomalias para aplicações *WebAssembly* com dados estáticos.

Este trabalho está estruturado em seis seções. A Seção 2 apresenta os conceitos necessários para a compreensão do trabalho. A Seção 3 apresenta os trabalhos relacionados. A Seção 4 apresenta a proposta do trabalho. A Seção 5 apresenta os resultados obtidos e a Seção 6 discute as conclusões do trabalho.

2. Fundamentação Teórica

Esta seção apresenta a fundamentação teórica para a compreensão do trabalho, incluindo o formato *WebAssembly*, detecção de anomalias e análise estática.

2.1. WebAssembly

O *WebAssembly* é um formato binário, onde desenvolvedores podem optar em utilizar o formato textual WebAssembly Text (WAT) para a implementação de aplicações ou utilizar compiladores específicos que permitem a tradução de códigos em linguagens como C, C++, Go e Rust para *WebAssembly* [Hoffman 2019].

Um módulo consiste de uma aplicação *WebAssembly*, que contém definições de funções, variáveis globais, memórias lineares e tabelas de chamadas indiretas. Funções e variáveis, bem como outros elementos do programa são identificados por índices representados por números inteiros [Lehmann et al. 2020]. Um módulo *WebAssembly* geralmente possui três seções: `Preâmbulo` com informações de início do módulo, `Padrão` que contém todas as informações da aplicação, como funções, e `Customizada` que possui informações para depuração [Kim et al. 2022]. A Figura 1 apresenta em detalhes a estrutura de um binário *WebAssembly*.

Preâmbulo		Padrão											Customizada
Magic	Versão	Type	Import	Funções	Table	Memory	Global	Export	Start	Code	Element	Dados	Qualquer tipo de dado

Figura 1. Estrutura de um binário *WebAssembly*.

Apenas quatro tipos primitivos são suportados: *i32*, *i64*, *f32* e *f64*, representando um inteiro de 32 ou 64 bits ou um número em ponto flutuante de 32 ou 64 bits, respectivamente.

O *WebAssembly* utiliza um formato de instruções em código binário que pode ser depurado por meio de conversores que tornam o código de máquina legível. As ferramentas estão disponíveis e tornam mais prática a análise das seções do código *WebAssembly* [Falliere 2018]. O formato de instruções *WebAssembly* foi projetado com foco em garantir a segurança de seus usuários. Seus principais recursos para a segurança são:

Ambiente virtualizado: os módulos *WebAssembly* rodam em uma máquina virtual baseada no modelo de pilha. Toda interação de entrada, saída e acesso a recursos do sistema operacional deve ser realizada através de funções incorporadas pelo *WebAssembly*, que devem ser importadas pelo módulo. Portanto, o *WebAssembly* é capaz de estabelecer políticas de segurança aos desenvolvedores e consegue garantir aos usuários que seu ambiente e os recursos do sistema estão sendo acessados pelos módulos de forma limitada e controlada [Rossberg 2018].

Memória linear: A memória linear dos módulos *WebAssembly* é instanciada em *buffers* gerenciados. Desta forma, as operações de leitura e escrita são limitadas a determinadas áreas de memória [Kim et al. 2022].

Integridade de controle de fluxo (CFI): Através de um controle de fluxo estruturado gerado durante o processo de compilação, módulos *WebAssembly* são protegidos contra ataques como injeção de *shellcode* ou abuso de saltos irrestritos realizados de forma indireta [Kim et al. 2022]. No entanto, o CFI do *WebAssembly* não é eficiente como CFI modernos utilizados para a defesas de binários nativos, com inúmeras chamadas vulneráveis para uso malicioso [Lehmann et al. 2020].

2.2. Detecção de Anomalias

Identificar padrões de dados que não correspondem ao comportamento esperado do sistema é o objetivo de um Sistema de Detecção de Anomalia [Chandola et al. 2009, Castanhel et al. 2020]. Para ser capaz de realizar essa detecção, o sistema deve possuir conhecimento sobre as instruções presentes no meio sendo monitorado. Esse meio pode ser a rede de conexão, em que os dados monitorados serão os do tráfego de rede, ou o *host*, nesse caso buscando anomalias em relação ao comportamento considerado normal para o sistema [Liu et al. 2018, Lemos et al. 2022].

Para Sistemas de Detecção de Intrusão baseados em *host* são as subsequências de instruções que, quando consideradas em conjunto, definem um comportamento anormal do sistema. A fim de monitorar o comportamento de códigos ou módulos específicos, duas técnicas podem ser empregadas, análise estática ou dinâmica [Chandola et al. 2009, Castanhel et al. 2021].

Análise dinâmica é uma técnica de análise de *software* que permite avaliar o comportamento do programa durante a sua execução. A análise dinâmica é realizada por meio de testes e simulações, o que possibilita identificar erros de programação, comportamentos inesperados e falhas de segurança durante a interação entre o código e o ambiente em que ele é executado [Kirchmayr et al. 2016, Lemos et al. 2023].

Já a análise estática é realizada através da extração de características do código, porém sem efetuar sua execução, definindo então uma representação abstrata do comportamento do programa [Kirchmayr et al. 2016]. Essa técnica tem sido amplamente utilizada especialmente em sistemas críticos, como os usados em aviação e controle de tráfego aéreo.

2.3. Segurança no ambiente Web

O *web browser* é um recurso indispensável atualmente, sendo essencial para o acesso a conteúdos encontrados na web. O *web browser* é composto por uma interface, que permite que usuários interajam com a aplicação, e uma *engine*, responsável por conectar a interface com componentes como a rede e armazenamento. Toda a troca de mensagens, apresentação e processamento de conteúdo recebido por uma *web page* é manipulado e gerenciado pela *engine* [Grosskurth and Godfrey 2006].

Devido à natureza do *web browser*, recursos externos são acessados, recuperados e executados [Alcorn et al. 2014]. O principal recurso de controle de um *web browser* é o Same-Origin Policy (SOP). Este mecanismo de controle restringe a interação de recursos vindos de diferentes origens. No entanto, os dois principais vetores para a realização de

ataques no ambiente são: (i) vulnerabilidades de *software*, ou seja, erros de implementação com implicações de segurança; e (ii) recursos do navegador que podem ser explorados para fins maliciosos, como a execução de JS maliciosos [Bandhakavi et al. 2010].

3. Trabalhos Relacionados

Na literatura o enfoque de trabalhos voltados à análise de binários *WebAssembly* não está centrada em um processo de detecção de intrusão. O foco dos trabalhos acaba sendo voltado à identificação de problemas no processo de desenvolvimento ou depuração de códigos. A Tabela 1 apresenta um conjunto de estratégias para a análise de aplicações *WebAssembly*, através do uso de estratégias estáticas e dinâmicas.

Tabela 1. Estratégia para análise de aplicações *WebAssembly*.

Referência	Estratégia	Descrição
[Quan et al. 2019]	Estática	*EVulHunter* é uma ferramenta para análise de de módulos Wasm para plataforma *blockchain* EOSIS.
[Stiévenart and De Roover 2020]	Estática	Wassail é uma ferramenta para análise de funções em *WebAssembly*.
[Brito et al. 2022]	Estática	Uma abordagem para a avaliação de binários e identificação de vulnerabilidades.
[Heinrich et al. 2023]	Dinâmica	Uma estratégia para a identificação de anomalias utilizando chamadas geradas pelo ambiente *sandbox WebAssembly*.
[Stiévenart et al. 2023]	Dinâmica	Apresenta um estudo sobre *slicers* para aplicações WebAssembly.

Abordagens que focam na avaliação estática de binários *WebAssembly* estão voltadas para a identificação de vulnerabilidades nos códigos desenvolvidos que possam ser explorados ou ocasionar um erro em produção. Estas ferramentas também estão voltadas para a geração de fluxo das aplicações *WebAssembly*, visando a identificação de vulnerabilidades que podem ter sido portadas de outras linguagens [Quan et al. 2019, Brito et al. 2022, Brito et al. 2022].

Uma estratégia dinâmica voltada para a detecção de anomalias é apresenta em [Heinrich et al. 2023]. A proposta foca em utilizar interações do ambiente *sandbox WebAssembly* com o *host* (descritas como chamadas WebAssembly System Interface (WASI)) para realizar o processo de detecção de anomalias. Os resultados obtidos são promissores, com um *f1score* superior a 92% para todos os modelos testados. O trabalho também ressalta como a representação dos dados através de uma abordagem categórica pode contribuir para estratégias de identificação de anomalias, já que permite a adição de informações ao conjunto de dados que não estariam disponíveis sem este recurso.

Levando em consideração vulnerabilidades em aplicações *WebAssembly*, [Lehmann et al. 2020] destaca que valores inesperados entre interações de linguagens como *WebAssembly* e JS podem gerar *overflow/underflow* de inteiros, que posteriormente pode ser explorado para um ataque de *buffer overflow*. Além disso, vulnerabilidades podem

persistir ao portar código para *WebAssembly* [Stiévenart et al. 2022]. A sobrescrita de dados pode ser explorada por um usuário malicioso para ganhar controle adicional do ambiente [Lehmann et al. 2020, McFadden et al. 2018]

Já o uso do *WebAssembly* com o intuito de realizar operações maliciosas, como a ofuscação de código através de *WebAssembly* [Balakrishnan and Schulze 2005] e *crypto-jacking* são situações observadas na literatura [Bian et al. 2020].

Por fim, é possível apontar que o desenvolvimento de estratégias voltadas ao processo de detecção de intrusão ainda é limitado na área. O foco das ferramentas desenvolvidas não está associado à identificação de conteúdo malicioso. Desta forma, discussões sobre o potencial no uso de informações extraídas dos binários *WebAssembly* para a identificação de ameaças ainda é escasso.

4. Proposta

Esta seção apresenta a proposta do trabalho. A Seção 4.1 apresenta a estratégia estudada. A Seção 4.2 discute as características utilizadas dos binários no processo de detecção.

4.1. Estratégia

Este trabalho apresenta uma estrategia para a detecção de anomalias em aplicações *WebAssembly*. O objetivo da proposta consiste em avaliar a eficácia da utilização de informações extraídas dos binários *WebAssembly* para um processo de detecção de anomalias.

Para alcançar este objetivos, as seguintes premissas foram definidas: (i) definição de uma estratégia de extração de características dos binários; e (ii) definição de uma estratégia de avaliação dessas características para detecção de anomalias.

Para alcançar a primeira premissa será utilizada o formato de depuração DWARF para extrair características dos códigos binários *WebAssembly*. O DWARF *WebAssembly* permite que sejam acessadas informações encontradas nas seções preambulo, padrão e customizada de um binário *WebAssembly*. O formato DWARF contém informações de depuração para aplicações *WebAssembly*, descrevendo todas as operações, variáveis e funções contidas no modulo Wasm através das informações de entidades presentes no código-fonte.

Um binário *WebAssembly* pode já conter as informações do DWARF ou elas podem ser geradas, através de um processo de transformação. Após este processo o formato DWARF pode ser utilizado para a compreensão das operações e funcionalidades realizadas pelo binário *WebAssembly* [Delendik 2020].

A Figura 2 apresenta o processo de transformação para uma aplicação *WebAssembly*. Existem três etapas, estas sendo: (i) transformação de um modulo *WebAssembly* para código e endereços nativos; (ii) extração das informações do binário para o formato DWARF; (iii) associação das informações encontradas no formato DWARF com a transformação do binário *WebAssembly* em um novo binário executável.

Foi utilizado o utilitário `llvm-dwarfdump` para a extração das informações de cada binário *WebAssembly*. As informações são incorporadas na seção customizada no binário Wasm avaliado, após serem transformadas. O resultado deste processo é um *dataset* que será apresentado na Seção 5.2. Devido à natureza do formato DWARF, é possível

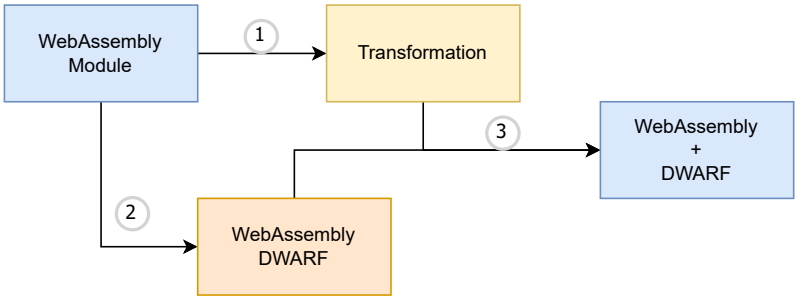

Figura 2. Exemplo da transformação de um binário *WebAssembly*.

assumir que para qualquer binário *WebAssembly* é possível extrair as informações usadas no *dataset*.

Após o processo de extração de informações do binário Wasm, diferentes técnicas podem ser utilizadas para a depuração da aplicação. Considerando um processo de identificação de anomalias, estas informações podem ser avaliadas e filtradas para um posterior processo de classificação.

Para a segunda premissa, uma estratégia de detecção de anomalia utilizando algoritmos de aprendizado de máquina foi definida. Foram selecionados sete algoritmos de aprendizado de máquina, dois algoritmos de uma classe e cinco algoritmos multi-classes. A escolha dos algoritmos usados para o processo de detecção e a decisão de comparar o desempenho de algoritmos de uma classe *versus* multi-classes é baseada em trabalhos correlatos anteriores [Galante et al. 2019, Castanhel et al. 2020, Lemos et al. 2023].

4.2. Características dos binários

Após a conversão dos binários *WebAssembly* para suportar o formato DWARF, informações podem ser extraídas dos mesmos. O formato DWARF oferece informações de um conjunto de marcações e atributos presentes em um binário Wasm. Estas informações permitem um posterior processo de análise da aplicação, para a identificação de erros de implementação ou, no caso deste trabalho, para a identificação de anomalias e detecção de intrusão.

Para a seleção dos atributos utilizados no processo de detecção foram identificados atributos disponíveis em aplicações *WebAssembly*. Os atributos selecionados consistem de informações representativas das funcionalidades que a aplicação exerce e informações genéricas do binário. No final deste processo, foram identificados 19 características relevantes para um processo de identificação de anomalia. Estas informações são genéricas para qualquer aplicação *WebAssembly*, possibilitando que futuras abordagens identifiquem os mesmos campos no binário Wasm. Os atributos selecionados extraem as seguintes informações de um binário Wasm:

(i) **Informações básicas** como tamanho do programa, o número de rótulos presentes no código e a linguagem do código fonte. Estas informações além de descrever a aplicação apresentam informações da linguagem utilizada antes do porte para *WebAssembly*;

(ii) **Tipos de dados** que inclui o número de declarações de tipos, tipos inteiros declarados, tipos inteiros sem sinal declarados, tipos palavra declarados e tipos ponteiro para arquivo declarados. Como o *WebAssembly* tem um conjunto restrito de tipos de dados nativo, é necessária a definição desses tipos nos módulos *WebAssembly*;

(iii) **Informações de sub-rotinas e subprogramas** considerado o número de variáveis declaradas, o número de declarações de sub-rotinas *inline*, o número de declarações de subprogramas e o número de parâmetros desses subprogramas;
(iv) **Estruturas em memória** frequência de membros declarados quando há o uso de estruturas ou classes; e
(v) **Informações de compartilhamento externo** inclui dados sobre a quantidade de atributos que determinam se as sub-rotinas são parte de um programa externo ou produzem informações acessíveis externamente e a quantidade de *tags* e atributos relacionados a utilização de chamadas de funções.

Estas características retratam informações sobre variáveis, parâmetro de funções, subprogramas, chamadas de arquivo, tipos de variáveis e linguagem caso a implementação tenha sido portada. Para o processo de detecção de anomalia, os algoritmos de aprendizado de máquina foram treinados e testados utilizando a frequência de aparição e codificação das variáveis presentes neste conjunto de características.

5. Avaliação dos Resultados

Esta Seção apresenta a avaliação dos resultados. A Seção 5.1 apresenta o objetivo da avaliação. A Seção 5.2 apresenta o conjunto de dados. A Seção 5.3 apresenta a discussão dos resultados.

5.1. Objetivos e experimentos

A proposta do trabalho consiste na avaliação do uso de informações estáticas de binários *WebAssembly* para a identificação de ameaças, averiguando a viabilidade no uso das informações presentes em binários *WebAssembly* através do formato DWARF.

Para os experimentos, dois grupos de algoritmos de aprendizado de máquina foram selecionados: o primeiro grupo consiste de algoritmos voltados para a classificação de múltiplas classes: *AdaBoost*, *K-Nearest Neighbors (KNN)*, *Multi-layer Perceptron (MLP)*, *Random Forest (RF)* e *XGBoost*. Estes modelos visam demonstrar como o processo de aprendizagem está se comportando ao considerar as características selecionadas para as classes de classificação “benigna” e “maliciosa”.

O segundo grupo de algoritmos selecionados consiste de modelos de uma classe, estes sendo *Isolation Forest* e *Support Vector Machines (SVM)*. Estes algoritmos focam em aprender uma classe, e comportamentos que não se enquadram nesta classe são considerados anomalias.

Ambos conjuntos de experimentos visam averiguar a eficácia no uso de informações estáticas de binários *WebAssembly* para a identificação de anomalias, demonstrando dois tipos de estratégias recorrentes durante o processo de identificação de anomalias.

5.2. Conjunto de dados

Para formar o *dataset* do experimento, foram consideradas duas fontes de dados, selecionadas levando em consideração as amostras presentes, tanto para definir o grupo de amostras benignas como amostras maliciosas [Lehmann and Pradel 2022, Stiévenart et al. 2022].

Ao todo foram selecionadas 277 amostras para o experimento. Estas amostras estão divididas entre 127 amostras benignas e 150 amostras maliciosas. O objetivo das amostras

é apresentar comportamentos recorrentes para aplicações benignas e vulnerabilidades ou erros de implementação para as amostras maliciosas.

Antes da execução dos experimentos e coleta das informações presentes nos binários, o processo de extração de informações com o formato DWARF foi utilizado, como apresentado na Seção 4.1. As características extraídas dos binários são salvas em um arquivo, que contém a contagem da aparição dos atributos e posteriores codificações de variáveis extraídas. Após este processo, as informações foram utilizadas para o processo de treinamento e teste dos modelos.

5.3. Discussão dos resultados

Os algoritmos foram treinados e testados em uma proporção de 1:1, ou seja, 50% dos dados foram usados para treinamento e 50% para teste. Todos os modelos utilizaram os parâmetros com as configurações padrões. A Tabela 2 apresenta os resultados alcançados pelos algoritmos avaliados, considerando as métricas usuais da área de aprendizado de máquina. Os resultados alcançados variaram de acordo com a estratégia de classificação utilizada pelos modelos de aprendizado de máquina. No entanto, os resultados são favoráveis para uma estratégia de detecção de anomalias utilizando quatro dos algoritmos multi-classe.

Tabela 2. Desempenho dos algoritmos de aprendizado de máquina para a detecção de anomalias.

Tipo	Classifier	Precision	Recall	F1Score	Accuracy	BAC	Brier Score
Multi classe	AdaBoost	96.10%	100%	98.01%	97.84%	97.69%	2.16%
	KNN	93.67%	100%	96.73%	96.40%	96.15%	3.6%
	MLP	71.15%	100%	83.15%	78.42%	76.92%	21.58%
	RandomForest	98.67%	100%	99.33%	99.28%	99.23%	0.72%
	XGBoost	96.10%	100%	98.01%	97.84%	97.69%	2.16%
Uma classe	IsolationForest	90.95%	89.21%	88.93%	89.21%	87.70%	-
	SVM	84.79%	79.14%	77.52%	79.14%	76.23%	-

O primeiro subgrupo de modelos utilizados consiste de algoritmos multi classe, onde um modelo aprende duas classes distintas de comportamentos para o posterior processo de classificação. A *precision* destaca que modelos como MLP e KNN tiveram o maior impacto por falsos positivos, com os modelos classificando amostras benignas como maliciosas. Falsos negativos não afetaram este conjunto de testes, como apresentado pela métrica *recall*.

A métrica *F1score* considera o impacto das métricas *precision* e *recall*, destacando o impacto dos falsos positivos. A métrica *accuracy* demonstra o impacto dos verdadeiros positivos e verdadeiros negativos, demonstrando que os modelos conseguiram aprender adequadamente os padrões dos binários através das características extraídas.

Observando os valores alcançados pelas métricas *Balanced Accuracy (BAC)* e *brier score*, a evidência de aprendizagem do modelo persiste, exceto para o algoritmo MLP. Devido ao valor de 21% de *brier score*, o modelo não está balanceado para realizar

corretamente a classificação entre as duas classes, ao contrário dos outros modelos, que possuem uma adequação na etapa de aprendizagem com valores inferiores a 4% para *brier score* e acima de 96% para BAC.

O uso de características dos binários por modelos que realizam a classificação entre duas classes demonstra potencial para estratégias de detecção de anomalias, já que a diferença entre ambos os grupos ficou clara para os modelos de detecção. Este fator destaca o potencial que informações extraídas de binários *WebAssembly* possuem para a identificação de anomalias em aplicações *WebAssembly*.

Já o segundo subgrupo consiste de dois algoritmos de uma classe que foram treinados só com uma classe, com expectativa que o modelo aprenda uma classe e qualquer padrão que seja diferente destas observações seja classificado como anomalia.

Ambos os modelos foram impactados por falsos positivos (*precision*) e falsos negativos (*recall*), o que reflete no valor alcançado pela métrica *F1score*. Apesar do *F1score* alcançado pelo algoritmo *Isolation Forest* ser superior aos valores alcançado pelo algoritmo MLP, o primeiro algoritmo ainda apresenta um valor de 87.7% para o BAC, demonstrando um impacto nos falsos positivos e falsos negativos. Mesmo assim, esse modelo ainda obteve um nível de aprendizado aceitável.

No geral, os algoritmos avaliados para a estratégia de detecção de anomalias alcançaram valores aceitáveis para a proposta. Os únicos algoritmos que não atenderam as expectativas foram o SVM e o MLP. O uso de informações extraídas de binários *WebAssembly* para a identificação de anomalias possui potencial, com modelos de aprendizado de máquina alcançando um nível de aprendizagem adequado para um processo de detecção.

A avaliação estática de binários é uma das técnicas de avaliação utilizadas por AntiVirus (AV) no processo de identificação de ameaças [Botacin et al. 2022]. sendo um processo essencial para garantir a segurança de sistema e a identificação de ameaças. Devido à natureza de aplicações *WebAssembly*, avaliações dinâmicas podem não ser possíveis, já que aplicações podem utilizar recursos externos para a execução dos binários [Heinrich et al. 2023].

Ao avaliar as três seções que compõem um binário *WebAssembly*, observamos que as seções de `padrão` e `customizada` contém informações essenciais para a aplicação. Estas informações foram extraídas e foi avaliado o potencial no uso destes campos para a identificação de padrões no binário através de uma avaliação estática.

Para um processo de detecção de anomalias, a proposta apresentada não é limitada por recursos externos que a aplicação possa requerer durante sua execução. Esta proposta pode ser utilizada para a identificação de problemas após o processo de desenvolvimento como para a identificação de conteúdo malicioso.

6. Conclusão

O *WebAssembly* é um formato binário que teve uma rápida adoção na Web. Por consequência, problemas relacionados com a segurança do *WebAssembly* foram identificados, sendo importante o desenvolvimento de estratégias para a detecção de intrusões no meio.

Este trabalho apresenta uma estratégia para a detecção de anomalias em aplicações *WebAssembly*. Através de uma estratégia estática foi realizada a extração de características

dos binários *WebAssembly* e avaliado o potencial no uso destas informações para um processo de detecção de anomalias.

Os resultados obtidos demonstram que informações extraídas de binários *WebAssembly* são úteis no processo de detecção de intrusão, sendo que 19 características permitiram a classificação dos binários com um *F1score* superior a 96% para quatro dos modelos treinados. Considerando trabalhos explorando dados estáticos na literatura, o respectivo trabalho demonstra o potencial no uso de informações encontradas no binário para o processo de detecção de anomalia.

Para trabalhos futuros, um conjunto de dados maior será gerado para explorar mais profundamente o uso de informações dos binários *WebAssembly* para a identificação de anomalias. Novas funcionalidades ainda estão sendo desenvolvidas para o formato *WebAssembly*.

Agradecimentos

Este trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), UDESC e FAPESC. Os autores também agradecem o Programa de Pós-Graduação em Informática da UFPR e a UTFPR *Campus* Dois Vizinhos.

Referências

Alcorn, W., Frichot, C., and Orru, M. (2014). *The Browser Hacker's Handbook*. John Wiley & Sons.

Balakrishnan, A. and Schulze, C. (2005). Code obfuscation literature survey. *CS701 Construction of Compilers*, 19.

Bandhakavi, S., King, S. T., Madhusudan, P., and Winslett, M. (2010). {VEX}: Vetting browser extensions for security vulnerabilities. In *19th USENIX Security Symposium (USENIX Security 10)*.

Bian, W., Meng, W., and Zhang, M. (2020). MineThrottle: Defending against Wasm in-browser cryptojacking. In *Proceedings of the 29th The Web Conference*, pages 3112–3118, Taipei, Taiwan. ACM.

Bosamiya, J., Lim, W. S., and Parno, B. (2022). Provably-Safe multilingual software sandboxing using WebAssembly. In *Proceedings of the 31st USENIX Security Symposium*, pages 1975–1992, Boston, MA, USA. USENIX Association.

Botacin, M., Domingues, F. D., Ceschin, F., Machnicki, R., Alves, M. A. Z., de Geus, P. L., and Grégio, A. (2022). Antiviruses under the microscope: A hands-on perspective. *Computers & Security*, 112:102500.

Brito, T., Lopes, P., Santos, N., and Santos, J. F. (2022). Wasmati: An efficient static vulnerability scanner for WebAssembly. *Computers & Security*, 118:102745.

Castanhel, G. R., Heinrich, T., Ceschin, F., and Maziero, C. A. (2020). Detecçao de anomalias: Estudo de técnicas de identificaçao de ataques em um ambiente de contêiner. In *Anais Estendidos do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 169–182. SBC.

Castanhel, G. R., Heinrich, T., Ceschin, F., and Maziero, C. A. (2021). Taking a peek: An evaluation of anomaly detection using system calls for containers. In *26th IEEE Symposium on Computers and Communications (ISCC 2021)*.

Ceschin, F., Gomes, H. M., Botacin, M., Bifet, A., Pfahringer, B., Oliveira, L. S., and Grégio, A. (2020). Machine learning (in) security: A stream of problems. *CoRR*, abs/2010.16045.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

Delendik, Y. (2020). DWARF for WebAssembly. `https://yurydelendik.github.io/webassembly-dwarf/`.

Falliere, N. (2018). Reverse engineering WebAssembly. `https://www.pnfsoftware.com/reversing-wasm.pdf`.

Galante, L., Botacin, M., Grégio, A., and de Geus, P. (2019). Forseti: Extração de características e classificação de binários elf. In *Anais Estendidos do XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 5–10. SBC.

Grosskurth, A. and Godfrey, M. W. (2006). Architecture and evolution of the modern web browser. *Preprint submitted to Elsevier Science*, 12(26):235–246.

Heinrich, T., Will, N. C., Obelheiro, R. R., and Maziero, C. A. (2023). Uso de chamadas WASI para a identificação de ameaças em aplicações webassembly. In *Anais Estendidos do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. SBC.

Hoffman, K. (2019). Programming webassembly with Rust: unified development for web, mobile, and embedded applications. *Programming WebAssembly with Rust*, pages 1–220.

Kim, M., Jang, H., and Shin, Y. (2022). Avengers, Assemble! survey of WebAssembly security solutions. In *Proceedings of the 15th International Conference on Cloud Computing*, pages 543–553, Barcelona, Spain. IEEE.

Kirchmayr, W., Moser, M., Nocke, L., Pichler, J., and Tober, R. (2016). Integration of static and dynamic code analysis for understanding legacy source code. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*, pages 543–552. IEEE.

Lehmann, D., Kinder, J., and Pradel, M. (2020). Everything old is new again: Binary security of WebAssembly. In *Proceedings of the 29th USENIX Security Symposium*, pages 217–234, Boston, MA, USA. USENIX Association.

Lehmann, D. and Pradel, M. (2022). Finding the dwarf: Recovering precise types from WebAssembly binaries. In *Proceedings of the 43rd International Conference on Programming Language Design and Implementation*, pages 410–425, San Diego, CA, USA. ACM.

Lemos, R., Heinrich, T., Maziero, C. A., and Will, N. C. (2022). Is it safe? identifying malicious apps through the use of metadata and inter-process communication. In *2022 IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE.

Lemos, R., Heinrich, T., Will, N. C., Obelheiro, R. R., and Maziero, C. A. (2023). Inspecting binder transactions to detect anomalies in android. In *Proceedings of the 17th Annual IEEE International Systems Conference*, Vancouver, BC, Canada. IEEE.

Liu, M., Xue, Z., Xu, X., Zhong, C., and Chen, J. (2018). Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys (CSUR)*, 51(5):98.

McFadden, B., Lukasiewicz, T., Dileo, J., and Engler, J. (2018). Security Chasms of Wasm. *NCC Group Whitepaper*.

Michael, A. E., Gollamudi, A., Bosamiya, J., Disselkoen, C., Denlinger, A., Watt, C., Parno, B., Patrignani, M., Vassena, M., and Stefan, D. (2022). Mswasm: Soundly enforcing memory-safe execution of unsafe code. *arXiv preprint arXiv:2208.13583*.

Naseem, F. N., Aris, A., Babun, L., Tekiner, E., and Uluagac, A. S. (2021). Minos: A lightweight real-time cryptojacking detection system. In *Network and Distributed System Security Symposium (NDSS)*.

Quan, L., Wu, L., and Wang, H. (2019). Evulhunter: Detecting fake transfer vulnerabilities for eosio's smart contracts at webassembly-level.(2019). *arXiv preprint arXiv:1906.10362*.

Romano, A., Lehmann, D., Pradel, M., and Wang, W. (2022). Wobfuscator: Obfuscating JavaScript malware via opportunistic translation to WebAssembly. In *Proceedings of the 43rd Symposium on Security and Privacy*, pages 1574–1589, San Francisco, CA, USA. IEEE.

Rossberg, A. (2018). Webassembly specification. `https://webassembly.github.io/spec/core/_download/WebAssembly.pdf`.

Stiévenart, Q., Binkley, D., and De Roover, C. (2023). Dynamic slicing of webassembly binaries. In *39th IEEE International Conference on Software Maintenance and Evolution*. IEEE.

Stiévenart, Q. and De Roover, C. (2020). Compositional information flow analysis for WebAssembly programs. In *Proceedings of the 20th International Working Conference on Source Code Analysis and Manipulation*, pages 13–24, Adelaide, Australia. IEEE.

Stiévenart, Q., De Roover, C., and Ghafari, M. (2022). Security risks of porting C programs to WebAssembly. In *Proceedings of the 37th Symposium on Applied Computing*, pages 1713–1722, Virtual Event. ACM.