

Deteción de Vulnerabilidades en Smart Contracts Usando Machine Learning a Nivel de Bytecode

Matias A. Carballo, Hernan D. Merlino,

Laboratorio de Sistemas de Información Avanzados,
Facultad de Ingeniería, Universidad de Buenos Aires,
Paseo Colon 850, CABA, Argentina,
{mcarballo, hmerlino}@fi.uba.ar

Abstract. La seguridad de los contratos inteligentes es un aspecto crucial para garantizar su despliegue exitoso. En este estudio, presentamos una investigación que se centra en el desarrollo de técnicas de extracción de propiedades y preprocesamiento de información de contratos inteligentes, con el objetivo de entrenar nuevas herramientas basadas en aprendizaje de máquina. Además, proponemos diversas arquitecturas y configuraciones de redes neuronales, junto con modelos de lenguajes específicos, para llevar a cabo la detección de vulnerabilidades en estos contratos.

Keywords: Smart Contracts; Blockchain; Ethereum; Security; Solidity; Deep learning.

1. Introducción

El uso de Smart Contracts^[1] en la plataforma Ethereum^[2] ha experimentado un notable aumento en los últimos años^[3]. Estos contratos inteligentes presentan características únicas, como su naturaleza inmutable al ser desplegados en la red, la visibilidad pública del código compilado en la cadena de bloques y la capacidad de interactuar automáticamente con otros contratos según los términos acordados. Sin embargo, estas características también implican riesgos significativos, lo que requiere una exhaustiva revisión y evaluación de los contratos antes de su despliegue en la red.

A pesar de los esfuerzos de evaluación previa, existe la posibilidad de que los contratos sean desplegados en la red con comportamientos inesperados bajo ciertas condiciones, lo cual representa un riesgo para los usuarios. Se ha documentado un caso en 2016 en el que un cibercriminal logró robar 50 millones de dólares mediante un ataque de reentrada, que fue posible debido al orden de las instrucciones en el contrato^[4].

En este trabajo de investigación, se analizan diversas estrategias para desarrollar herramientas de detección de vulnerabilidades en Smart Contracts, basadas en el análisis del código binario publicado en la red de Ethereum. Se examinan las diferentes partes que componen el código binario y se identifican las características relevantes que se pueden utilizar para detectar posibles vulnerabilidades. Además, se propone el uso de modelos de redes neuronales con diversas arquitecturas para evaluar la calidad de los resultados obtenidos mediante el análisis del código binario.

Mediante este enfoque, se busca mejorar la seguridad de los Smart Contracts en Ethereum y reducir el riesgo asociado a la explotación de vulnerabilidades. Los

resultados de esta investigación contribuirán al desarrollo de herramientas más efectivas para identificar y prevenir posibles ataques y comportamientos maliciosos en los contratos inteligentes.

2. Background

2.1 Contratos Inteligentes

Cuando se despliega un contrato en la red Ethereum, el código subyacente se representa en forma de bytecode, que es una forma compacta y binaria de representar las instrucciones y la lógica del contrato. El bytecode es una secuencia de instrucciones de bajo nivel que está diseñada para ser ejecutada por la Máquina Virtual Ethereum (EVM, por sus siglas en inglés), que es el entorno de ejecución en la red Ethereum.

El código fuente original del contrato, escrito en Solidity u otro lenguaje de programación compatible con Ethereum, se somete a un proceso de compilación para convertirlo en bytecode.

2.2 Extracción de Features

El bytecode puede ser dividido en cinco bloques de bytes que desempeñan funciones específicas. Estos bloques son: Creation Code, Function Selector, Wrapper, Body y Metadata Hash^[5].

En el contexto del análisis que se llevará a cabo, se detectará la porción de bytecode que corresponde al Body de cada contrato. Esta porción del bytecode contendrá la información relevante que se utilizará en la investigación. Los otros bloques de bytes, como Creation Code, Function Selector, Wrapper y Metadata Hash, no serán considerados en este estudio y no estarán disponibles para los modelos utilizados. Esto permitirá a los modelos centrarse en la información esencial.

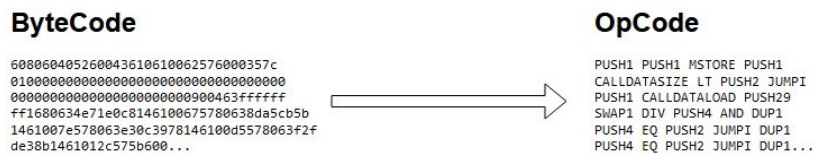


Fig. 1. La figura muestra la transformación de una cadena de bytes a una secuencia de operadores. Todos los contratos se truncaron a un largo máximo de 4048 operadores.

Otra consideración importante es que en la EVM, todas las instrucciones están compuestas por un operador, excepto las operaciones PUSH, que están seguidas de un valor inmediato. Sin embargo, para nuestra contextualización, descartaremos estos valores inmediatos, obteniendo una secuencia de operaciones (OpCodes)^[6] comprensibles por la EVM.

Esta representación de secuencia de operaciones es útil para comprender la lógica y el flujo de ejecución del contrato, ya que permite analizar cada operación individualmente y entender cómo interactúan con la pila de la EVM.

2.3 Tipos de Vulnerabilidades

Se utilizarán como fundamento los hallazgos obtenidos en el análisis de 47.518 contratos^[7] de Ethereum. Nuestro enfoque consiste en la aplicación de técnicas de aprendizaje automático a partir de bytecode presente en la red de Ethereum, con el objetivo de identificar patrones en los contratos vulnerables. En las fases de entrenamiento y predicción, se excluirán los archivos Solidity que contengan más de un contrato.

Abordaremos individualmente cada vulnerabilidad de forma separada. El objetivo de cada modelo será detectar si una vulnerabilidad específica está presente o no en un contrato dado. Por lo tanto, terminaremos con siete modelos diferentes de clasificación binaria, cada uno enfocado en una técnica de detección de vulnerabilidades particular.

Tabla 1. Tipos de vulnerabilidades y cantidad de contratos con las mismas detectadas.

Vulnerabilidad	Descripción	Cantidad de contratos
Access Control	Fallo al usar modificadores de funciones o el uso de tx.origin.	585
Arithmetic	Desbordamiento/subdesbordamiento de enteros.	2,137
Denial Service	Contrato abrumado por cálculos que consumen mucho tiempo.	644
Front Running	Dos transacciones dependientes que invocan el mismo contrato se incluyen en un bloque.	932
Reentrancy	Las llamadas de función reentrante hacen que un contrato se comporte de una manera inesperada.	1,003
Time Manipulation	La marca de tiempo del bloque es manipulada por el minero.	271
Unchecked Low Calls	call(), callcode(), delegatecall() o send() fallan y no se verifican.	479

Para lograr esto, balancearemos los conjuntos de datos de manera que haya una cantidad igual de contratos sin vulnerabilidades detectadas y contratos que presenten diferentes vulnerabilidades. Este enfoque nos permitirá entrenar y evaluar nuestros modelos de manera equilibrada, asegurando una representación adecuada de los casos con y sin vulnerabilidades.

3. Técnicas Propuestas

Dentro de esta sección se mostrará el detalle de cómo se aplican diferentes técnicas a los mismos set de datos confeccionados anteriormente de manera de obtener resultados comparables entre sí. Cada modelo utilizará las secuencias de *OPCODES* como entrada y la categoría binaria de presencia o ausencia de la vulnerabilidad como salida.

3.1 Regresión Logística e Ingeniería de Características

Este modelo se presenta como un punto de referencia inicial, con requisitos computacionales mínimos, un tiempo de entrenamiento rápido pero que aún sea lo suficientemente complejo como para obtener resultados significativos en la detección de vulnerabilidades.

Al aplicar TF-IDF a las operaciones, bigramas y trigramas de las secuencias de *OPCODES*, se asignará un peso a cada característica según su frecuencia en un contrato específico y su rareza en el conjunto de contratos. Esto permitirá capturar las operaciones y combinaciones de operaciones que son más relevantes y distintivas para cada contrato.

Se entrenará un modelo supervisado de regresión logística binaria para cada tipo de vulnerabilidad. La regresión logística binaria es un modelo estadístico que se utiliza para predecir una variable categórica binaria, en este caso, la presencia o ausencia de una vulnerabilidad específica.

La función logística en una dimensión tiene la forma:

$$P(Y = 1 | X) = (1 + e^{-z})^{-1} \quad (1)$$

Donde $P(Y=1|X)$ es la probabilidad de que la vulnerabilidad esté presente dadas las características X , y z es una combinación lineal de los features ponderados por los coeficientes del modelo.

Para ajustar el modelo al conjunto de datos, se utiliza la función de pérdida de "negative log-likelihood" (log-verosimilitud negativa), que es una medida de qué tan bien se ajusta el modelo a los datos observados. Esta función de pérdida se utiliza para maximizar la probabilidad conjunta de los datos de entrenamiento dado el modelo.

3.2 Redes Neuronales Recurrentes

Las redes neuronales recurrentes se destacan por su habilidad de retener información dentro de una serie ordenada de datos, un ejemplo de estas es la red LSTM^[8].

En particular se utilizó una arquitectura basada en LSTM^[9]. A esta arquitectura se la complementará con dos capas densas al final, donde cada neurona de una capa se conecta con todas las de la siguiente, para realizar una tarea de clasificación.

Para que el modelo pueda utilizar una secuencia de operaciones, es necesario aplicar un proceso de tokenización. La tokenización implica tomar cada operación individual y transformarla en un token, que es una representación numérica única.

Como técnica de Tokenización se utilizó SentencePiece^[10] que proporciona flexibilidad para ajustar el tamaño del vocabulario y la granularidad de las unidades de

subpalabras, así como también ofrece tokens especiales que ayudan a condensar información.

3.3 Modelo de Lenguaje y Transferencia de Conocimiento

Una metodología innovadora en el campo de las redes neuronales consiste en la transferencia de conocimientos entre ellas. Esta implica adaptar y aplicar el conocimiento aprendido de una tarea previa a una nueva tarea. Esto se logra mediante la transferencia de los pesos y parámetros del modelo previamente entrenado al nuevo modelo, lo que permite aprovechar la información relevante ya aprendida y acelerar el proceso de entrenamiento en el nuevo contexto. La transferencia de conocimiento puede mejorar el rendimiento y la eficiencia del modelo en la tarea objetivo.

Partiremos con el entrenamiento de un modelo de lenguaje utilizando una arquitectura encoder-decoder de manera tal que el encoder tenga la misma arquitectura basada en LSTM^[9] para poder luego hacer dicha transferencia de conocimiento.

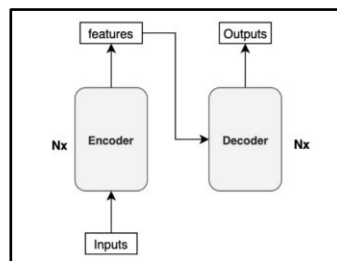


Fig. 2. Arquitectura encoder-decoder. Esta figura muestra los componentes esenciales de la arquitectura. Se deja en evidencia la relación de las distintas partes con la información de entrada y salida, así como también la ubicación de la representación vectorial.

Dentro de la arquitectura de encoder-decoder, el encoder es responsable de capturar la información intrínseca de las secuencias de entrada y generar una representación vectorial que capture su significado. El decoder utiliza dicha representación para contextualizar y generar elementos secuenciales de manera coherente.

Para entrenar un modelo de lenguaje con esta arquitectura, la información de entrada consiste en una lista de tokens. La información de salida es la misma lista de tokens, pero con un corrimiento, lo que implica que cada token en la salida es el siguiente token en la secuencia original.

Tabla 2. Ejemplo de set de datos para el entrenamiento del modelo de lenguaje.

Información de entrada	Información de salida
xxbos jumpdest push1 push1 swap1	jumpdest push1 push1 swap1 sload
exp dup2 sload dup2 push20	dup2 sload dup2 push20 mul

Para la confección de este dataset no es necesario el conocimiento previo de la presencia o no de vulnerabilidades, lo que nos permite utilizar cualquier contrato desplegado en la red de ethereum.

Entrenaremos el modelo de lenguaje utilizando la totalidad de los contratos disponibles, este modelo será el mismo para todas las vulnerabilidades.

Para el proceso de transferencia de conocimiento tomaremos el encoder previamente entrenado y lo complementaremos con dos capas de redes neuronales densas para especializarse en la tarea de clasificación. Una diferencia a la hora del entrenamiento es la técnica de congelamiento, esta consiste en una primera iteración del entrenamiento donde los pesos del encoder no se actualizarán según el gradiente, y se utilizara un parámetro de aprendizaje mayor. En la segunda iteración se reduce ese coeficiente de aprendizaje y se entrena toda la arquitectura en conjunto.

3.4 Transformers

Debido a lo extenso que puede ser un contrato, propondremos una arquitectura cuyo núcleo contiene una arquitectura de Transformers^[11]. Por otra parte, entrenaremos un modelo de lenguajes que se ajuste a este núcleo para poder aplicar transferencia de conocimiento.

3.5 RoBERTa

Para el núcleo de la arquitectura utilizaremos una implementación basada en RoBERTa^[12], la cual adaptaremos con la siguiente configuración: 512 tokens, 4 hidden layers, 6 attention heads y 216 embedding size.

3.6 Tokenización y Modelo de Lenguaje

Para la tokenización realizaremos un byte-level Byte-Pair-Encoding^[13], esta puede capturar subunidades significativas en la secuencia, lo que ayuda a reducir la dimensionalidad creando una representación más eficiente y mejorar la compresión de datos.

Entrenaremos RoBERTa^[12] como modelo de lenguaje, esto se realiza mediante la técnica de máscaras, una máscara es un token específico que reemplaza un 15% de los tokens presentes en cada secuencia. El objetivo del entrenamiento es lograr predecir que tokens debe reemplazar en las máscaras para obtener la secuencia original.

3.7 Arquitectura

Propondremos el uso de una arquitectura siamesa de 8 bloques de RoBERTa^[12], seguida de dos capas de redes neuronales densas. Esta elección se debe a que nuestro corpus de texto consta de 4096 tokens, mientras que el límite de tokens de entrada del modelo es de solo 512.

En esta arquitectura siamesa, dividimos la información de entrada en 8 bloques, lo que nos permite procesar cada bloque de forma independiente. Cada bloque se pasa a

través de un bloque correspondiente de RoBERTa^[12], que comparte los mismos pesos y arquitectura. Esto significa que los 8 bloques de RoBERTa^[12] son idénticos, lo que facilita la paralelización del procesamiento.

Después de procesar cada bloque individualmente, recopilamos la información mediante dos capas de redes neuronales densas. Estas capas se utilizan para fusionar la información extraída de los 8 bloques y generar una representación global del texto de entrada.

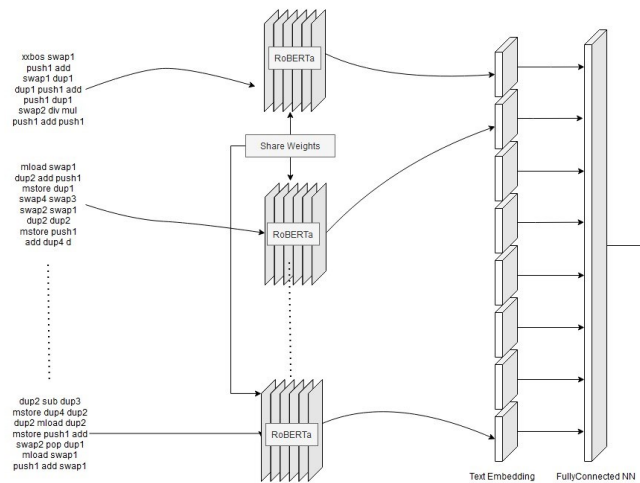


Fig. 3. La figura muestra una representación de la arquitectura siamesa.

4. Resultados

Para poder evaluar y comparar el rendimiento de los diferentes modelos en relación a los distintos tipos de vulnerabilidades, utilizamos la métrica del área bajo la curva ROC (Receiver Operating Characteristic).

La curva ROC es una representación gráfica que muestra la relación entre la tasa de verdaderos positivos (TPR) y la tasa de falsos positivos (FPR) a medida que se varía el umbral de clasificación del modelo. El área bajo esta curva (AUC-ROC) es una medida numérica que resume el rendimiento global del modelo. Un AUC-ROC mayor indica un mejor rendimiento, ya que significa que el modelo tiene una mejor capacidad para distinguir entre las clases positivas y negativas.

Tabla 3. Los resultados del Area bajo la curva ROC para cada modelo.

Vulnerabilidad	Reg. Log.	LSTM	LSTM + LM	Transformers	Transf. + LM
Acc. Ctrl.	0.87	0.57	0.87	0.61	0.76
Arithmetic	0.90	0.90	0.91	0.68	0.82
D. Service	0.87	0.62	0.91	0.60	0.82
F. Running	0.86	0.86	0.89	0.53	0.78

Reentrancy	0.83	0.51	0.91	0.62	0.81
Time Man.	0.81	0.79	0.85	0.53	0.68
Unchecked L.C.	0.82	0.53	0.83	0.55	0.71

Observamos que los modelos basados en redes neuronales recurrentes que se inicializan con un modelo de lenguaje pre-entrenado tienden a mostrar mejores resultados en comparación con el resto de los enfoques. Así mismo, cuando analizamos el modelo de RoBERTa^[12] inicializado con un modelo de lenguaje pre-entrenado, también arroja una mejora sustancial sobre el mismo modelo inicializado aleatoriamente.

Por otra parte, la regresión logística muestra un alto rendimiento, lo que sugiere grandes oportunidades de mejora para arquitectura de Transformers propuesta, ya sea en volumen de datos para el entrenamiento, o en diferentes configuraciones de la misma.

5. Conclusiones

Hemos logrado entrenar varios modelos que son capaces de identificar vulnerabilidades con diferentes niveles de precisión. En segundo lugar, hemos aplicado técnicas de preprocesamiento y segmentación de datos para evitar la inclusión de información redundante e innecesaria.

El enfoque utilizado en esta investigación presenta la ventaja de ser flexible y adaptable a medida que surjan nuevos tipos de vulnerabilidades. Esto se debe a las técnicas aplicadas y la capacidad de los modelos para aprender y generalizar a partir de los datos de entrenamiento.

Finalmente es importante destacar que las técnicas utilizadas en este enfoque se centran en el análisis de secuencias de operaciones para identificar patrones relevantes. Esto proporciona una ventaja significativa en términos de velocidad de respuesta en comparación con las técnicas tradicionales que implican pruebas exhaustivas o métodos computacionalmente costosos y lentos.

Agradecimientos

Este proyecto forma parte del PIDAE 2022 (1-285) Computación Descentralizada y Silos de Procesamiento.

References

1. Szabo, N.: Smart contracts: building blocks for digital markets, EXTROPY: The Journal of Transhumanist Thought, (16), vol. 18, no. 2, 1996.
2. Buterin, V.: et al., A next-generation smart contract and decentralized application platform, white paper, vol. 3, no. 37, 2014

3. Asmakov, A.: Ethereum Smart Contracts Deployment Jumped 293% in 2022: Alchemy Developer Report <https://decrypt.co/119371/ethereum-smart-contracts-deploymentjumped-293-2022-alchemy-developer-report>
4. Meyer, B.: Ethereum smart contract vulnerabilities can lead to millions in losses <https://cybernews.com/security/ethereum-smart-contract-vulnerabilities/>
5. Santander, A., Arias, L.: Deconstructing a Solidity Contract, <https://blog.openzeppelin.com/deconstructing-a-solidity-contract-part-i-introduction832efd2d7737/>
6. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger berlin version beacfbf – 2022-10-24
7. Durieux, T., Ferreria, J.F., Abreu, R., Cruz, P.: Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts, arXiv:1910.10601
8. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory, 1997, <https://www.bioinf.jku.at/publications/older/2604.pdf>
9. Merity, S., Keskar, N. S., Socher, R.: Regularizing and Optimizing LSTM Language Models, arXiv:1708.02182
10. Kudo, T., Richardson, J.: SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing, arXiv:1808.06226
11. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I.: Attention Is All You Need, arXiv:1706.03762
12. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: RoBERTa: A Robustly Optimized BERT Pretraining Approach, arXiv:1907.11692
13. Sennrich, R., Haddow, B., Birch, A.: Neural Machine Translation of Rare Words with Subword Units, arXiv:1508.07909