UMCS

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: Informatyka

Piotr Jasina
nr albumu: 279183

Identyfikacja inteligentnych kontraktów w sieci Ethereum

Ethereum smart contracts identification

Praca licencjacka
napisana w Zakładzie Cyberbezpieczeństwa
pod kierunkiem dr. Damiana Rusinka

Lublin rok 2019

Spis treści

Wstęp

...

Rozdział 1

Ethereum

1.1 Historia

Literatura: [2, ?]. TODO

1.2 Opis platformy

Literatura: [2, ?]. TODO

1.3 Ethereum Virtual Machine

Literatura: [2, ?]. TODO

1.4 Inteligentne kontrakty

Literatura: [2, ?]. TODO

Rozdział 2

Solidity

2.1 Sygnatura funkcji

Literatura: [2, ?]. TODO

2.2 Selektor funkcji

Literatura: [2, ?]. TODO

2.3 Generowanie akcesorów podczas kompilacji

Literatura: [2, ?]. TODO

Rozdział 3

Projekt Aplikacji

Celem mojej pracy licencjackiej było stworzenie aplikacji internetowej umożliwiającej identyfikację inteligentnych kontraktów wykorzystywanych w sieci Ethereum. Dzięki aplikacji użytkownik po wprowadzeniu na stronie kodu bajtowego kontraktu jest w stanie otrzymać najbardziej prawdopodobną implementacje kontraktu napisana w języku Solidity bazując na bazie danych aplikacji.

Poniżej zostało opisane działanie aplikacji wraz ze szczegółowym opisem funkcjonalności, architektury oraz wykorzystanych technologi.

3.1 Funkcjonalność

Podczas korzystania z aplikacji uzytkownik ma dostepne trzy funkcjonalnosci: identyfikację inteligentnych kontraktu, wprowadzanie plików źródłowych kontraktów do aplikacji oraz dokumentacje API aplikacji. Na stronie głównej poniżej menu znajduje się opis aplikacji wraz z aktualna liczba kodów źródłowych znajdujących się w bazie danych aplikacji oraz podstawowymi definicjami związanymi z aplikacją.

3.1.1 Identyfikacja inteligentnych kontraktów

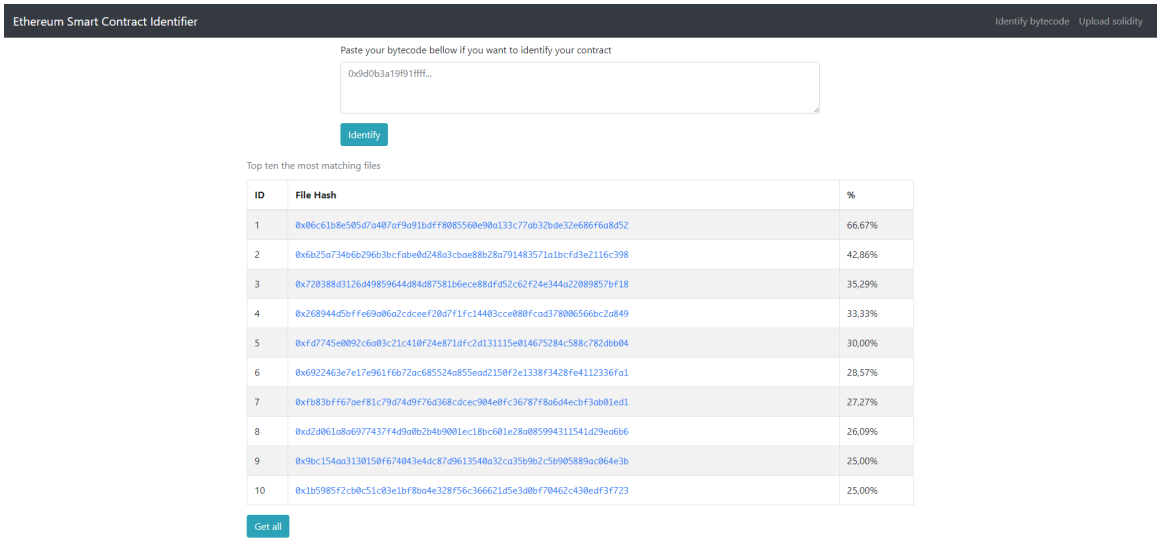
Pierwsza opcją dostępną w aplikacji jest identyfikacja inteligentnych kontraktów. Identyfikacje kontraktów rozpocząć będąc na stronie głównej lub za pomocą podstrony do której można się dostać klikając przycisk **Identify bytecode** znajdujący się w menu. W przypadku wprowadzenia kodu bajtowego na stronie głównej zostaniemy przekierowani na podstronę na której pojawią się wyniki identyfikacji kontraktu. Jeśli użytkownik wykorzysta opcje z menu to będzie musiał na podstronie wprowadzić kod bajtowy swojego kontraktu. W obydwu przypadkach, zarówno na stronie głównej, jak i podstronie dedykowanej tej funkcjonalności, kod bajtowy należy wprowadzać w szesnastkowym systemie liczbowym.

Podczas wprowadzania kodu istnieje możliwość wprowadzenia kodu z prefiksem **0x** lub bez tego prefixu. Jeśli użytkownik poda kod z prefiksem to aplikacja podczas przetwarzania tego kodu zignoruje ten prefiks. Takie rozwiązanie zostało zastosowane w celu zapewnienia użytkownikowi większej wygody oraz komfortu w korzystaniu z aplikacji. Użytkownik nie będzie musiał zastanawiać się czy kopiując kod bajtowy z dowolnego źródła, jest on z prefiksem czy nie, ponieważ obie opcje są wspierane.

Pomyślne wprowadzone dane wykorzystywane do identyfikacji zatwierdzamy przyciskiem "Submit", a następnie po stronie serwerowej aplikacji rozpoczynany jest proces analizy wprowadzonego kodu bajtowego oraz wyszukiwane są najbardziej prawdopodobne implementacje. W rezultacie jak możemy zobaczyć na rysunku 3.1 użytkownik otrzymuje listę wyszukanych dziesięciu najbardziej prawdopodobnych implementacji posortowanych malejąco według współczynnika dopasowania. Mimo że domyślnie jest wyświetlanych tylko dziesięć najbardziej prawdopodobnych implementacji istnieje też możliwość pobrania wszystkich wyników identyfikacji kontraktu używając przycisku **Get all**. Jak widać na rysunku 3.1, przycisk **Get all** znajduje się pod pierwszą dziesiątka wyników. Po naciśnięciu

przycisku strona zostanie zaladowana ponownie oraz zostanie wyświetlona pełna lista identyfikatorów plików wraz ze współczynnikiem dopasowania do kodu bajtowego.

Po naciśnięciu w jedną z wyświetlanych implementacji, użytkowniki w nowej karcie przeglądarki otworzy się specjalna podstrona umożliwiająca podgląd implementacji. Jak widać na rysunku 3.2 kod źródłowy kontraktu jest wyświetlany z podświetleniem składni przygotowanym dla języka Solidity, natomiast po lewej stronie można zobaczyć specjalnie przygotowaną numeracje wierszy która ma za zadanie ułatwić poruszanie się po kodzie źródłowym na stronie. Rozwiązanie z numerowaniem linii zostało zaimplementowane w taki sposób, aby podczas kopiowania kodu źródłowego ze strony nie zostawały kopiowane wraz z nim liczby identyfikujące konkretną linie w kodzie.

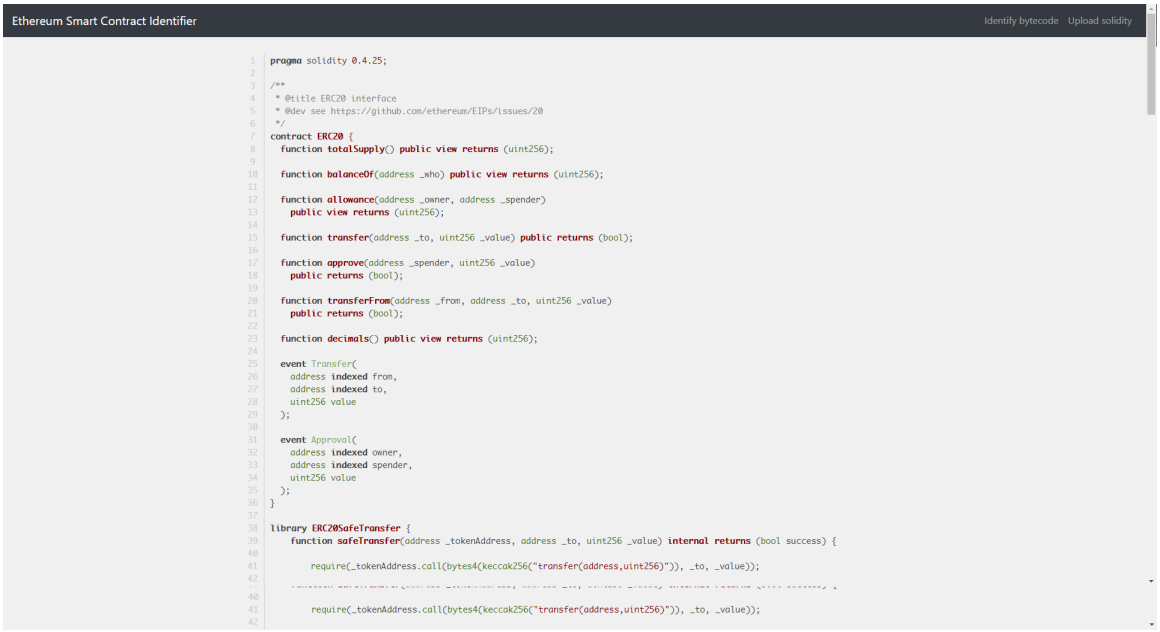


Rysunek 3.1: Wynik identyfikacji inteligentnego kontraktu

3.1.2 Dodanie kodu źródłowego kontraktu do aplikacji

Kolejna funkcjonalnością dostępną dla użytkownika jest możliwość dodania własnego kodu źródłowego kontraktu napisanego w języku Solidity. Opcja ta umożliwia użytkownikowi uzupełnienie aktualnej bazy danych o kolejne kody źródłowe inteligentnych kontraktów, w wyniku takiego działania wszyscy pozostali użytkownicy maja większa szanse na precyzyjną identyfikacje kontraktu. W celu wykorzystania tej funkcjonalności użytkownik musi użyć przycisku **Upload solidity**, po czym zostanie on przekierowany na podstronę na której będzie mógł wprowadzić kod źródłowy, który zostanie przesłany do serwera aplikacji. Zostały utworzone dwie możliwości wprowadzania kodów źródłowych.

Pierwszą sposobem jest przesłanie do aplikacji pliku zawierającego implementacje kontraktu napisana w języku Solidity. W tym przypadku użytkownik powinien kliknąć przycisk **Browse**, który umożliwi mu wybranie za pomocą prze-



Rysunek 3.2: Podgląd implementacji

glądarki internetowej konkretnego pliku znajdującego się na dysku lokalnym, a następnie zatwierdzić go przyciskiem **Upload** znajdującym się obok wcześniej wspomnianego przycisku.

Innym sposobem na przesłanie kodu źródłowego do aplikacji jest wklejenie kodu źródłowego bezpośrednio do formularza znajdującego się po prawej części strony internetowej. Ta opcja została utworzona w celu zapewnienia użytkownikowi większej elastyczności i komfortu w korzystaniu z aplikacji. Przykładowo podczas korzystania z aplikacji, użytkownik może bezpośrednio skopiować kod źródłowy, który jest w dowolnym innym źródle tekstowym np. innej stronie internetowej i wkleić go bezpośrednio do aplikacji bez konieczności tworzenia pliku tymczasowego.

3.1.3 Interfejs programistyczny aplikacji

Trzecia funkcjonalnością aplikacji jest interfejs programistyczny. Umożliwia on tworzenie własnego oprogramowania oraz skryptów, bazując na utworzonej prze zemnie aplikacji, przez innych programistów. Dzięki temu można wykorzystać mechanizmy zaimplementowane w aplikacji w celu rozszerzenia ich w innej aplikacji lub w celu zautomatyzowania niektórych procesów bez wykorzystania GUI (ang. graphical user interface) aplikacji.

W celu skorzystania z interfejsu programistycznego należy utworzyć specjalne żądanie HTTP. Za pomocą interfejsu aplikacji użytkownik ma możliwość dostarczenia na serwer nowego kodu źródłowego, pobrania kodu źródłowego na podstawie na podstawie hasza wcześniej przesłanego pliku źródłowego oraz identyfikacje kontraktu wykorzystujac.

Przykładowym zastosowaniem API (ang. Application programming interface) może być utworzenie skryptu umożliwiającego zautomatyzowanie wysyłania kodów źródłowych do aplikacji, bez konieczności korzystania z interfejsu graficz-

nego aplikacji. W tym przypadku użytkownik, który chce przesłać nowy plik do aplikacji musi najpierw przejść proces autoryzacji, natomiast pozostałe funkcje API nie wymagają uwierzytelniania użytkownika przez aplikacje.

Listing 3.1: Przykladowe żądanie HTTP

```
1 GET /api/solidityFiles/sourceCode?fileHash=0
    x06c61b8e505d7a407af9a91bdff8085560e90a133c77ab32bde32e686f6a8d52
     HTTP/1.1
2 Host: contractmy.herokuapp.com
3 Connection: keep-alive
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
    /537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
    image/webp,image/apng,*/*;q=0.8
8 Accept-Encoding: gzip, deflate, br
9 Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
```

Pobieranie informacji o kodzie zrodlowym z API

Jeśli użytkownik chce pobrać z aplikacji informacje o konkretnym kodzie źródłowym danego pliku, w tym celu powinien utworzyć żądanie HTTP. W przypadku tej aplikacji do wykonania tej czynności nie jest potrzebne uwierzytelnienie użytkownika. Na listingu 3.1 można zaobserwować przykładowe żądanie HTTP, które jest wykorzystane w celu pobrania informacji o konkretnym kodzie źródłowym.

Użytkownik podczas korzystania z API powinien zwrócić szczególna uwagę na wykorzystywaną metodę w nagłówku HTTP, ponieważ interfejs aplikacji wspiera tylko wybrane metody. W tym przypadku w pierwszej linii na listingu 3.1 widać, że używaną metodą jest **GET**, która jest jedną z kilku dostępnych metod w

standardzie *HTTP/1.1*, istnieją też między innymi metody: *POST*, *HEAD*, *PUT* czy *DELETE*. Metody HTTP zostały opisane szczegółowo w sekcji 9.2 dokumentu RFC2616 [1].

Po metodzie GET, jak widać na listingu 3.1, znajduje się adres pod którym zostanie wysyłane żądanie. Na końcu adresu po znaku zapytania podawany jest atrybut o nazwie *fileHash* ze wskazaną wartością hasza pliku. Przykładową odpowiedz w formacie JSON (w skróconej formie) możemy zobaczyc na listingu 3.2:

Identyfikacja inteligentnego kontraktu za pomocą API

Podczas identyfikacji inteligentnego kontraktu, za pomocą interfejsu programistycznego, żądanie wygląda nieco inaczej niż poprzednio. Podczas identyfikacji wykorzystywana jest metoda POST. W nagłówku HTTP należy wiec wprowadzić tą metodę, a następie podać ścieżkę */api/bytecode*, pod którą należy wysłać żądanie. W przypadku metody POST opisanej w sekcji 9.5 dokumentu RFC2616 [1] podajemy poniżej nagłówka zawartość, która będzie przesłana do serwera. W odróżnieniu od wykorzystanej poprzednio metody GET, tutaj zawartość zostaje podana poniżej nagłówka zamiast w adresie.

API pod adresem apibytecode wymaga od użytkownika dwóch atrybutów o nazwach: bytecode oraz allFiles. Poniżej nagłówka HTTP w żądaniu należy wprowadzić odpowiednio zakodowane atrybuty. W celu zakodowania listy atrybutów, każdy atrybut zapisuje się w postaci:

```
NAZWA_ATRYBUTU=WARTOSC_ATRYBUTU
```

łącząc je wszystkie ampersandem. Poprawnie zakodowana zawartość w żądaniu HTTP w tym przypadku powinna wyglądać następująco:

```
bytecode=KOD_BAJTOWY&allFiles=WARTOSC_LOGICZNA.
```

W tym przypadku wartość KOD_BAJTOWY użytkownik musi zamienić na swój kod bajtowy, który chce przeanalizować oraz WARTOSC_LOGICZNA należy zamienić na napis *true* lub *false*, w zależności czy chcemy w rezultacie otrzymać wszystkie pliki, które brały udział, czy chcemy otrzymać tylko pierwsze dziesięć najbardziej pasujących plików. Dodatkowo w nagłówku należy wprowadzić parametr **Content-Length**, dla którego należy podać długość wysyłanych. Przykładowo dla zawartości:

```
bytecode=234a12b45&allFiles=true
```

fragment nagłówka powinien wyglądać następująco:

```
Content-Length: 32
```

W rezultacie zapytania użytkownik otrzyma listę składająca się z haszu pliku oraz współczynnika dopasowania danego pliku w formacie JSON.

Przesyłanie nowego kodu źródłowego za pomocą API

Podczas przesyłania nowego kodu źródłowego przy wykorzystaniu API istnieje możliwość przesłania kodu źródłowego. W odróżnieniu od sposobu opisanego powyżej, teraz nie będą używane atrybuty z nazwami, tylko zostanie wysłane żądanie HTTP w którym pod nagłówkiem będzie wprowadzony tylko kod źródłowy.

W celu wysłania takiego żądania najpierw użytkownik musi zostać uwierzytelniony, w innym przypadku proba dodania nowego kodu źródłowego do aplikacji ukończy się niepowodzeniem. Zakładając, że użytkownik został uwierzytelniony to bazując na przykładowym żądaniu HTTP z listingu 3.1 należy wprowadzić poniżej nagłówka cały kod źródłowy kontraktu.

Należy pamiętać, że kod źródłowy musi być zaimplementowany w języku Solidity, w innym wypadku otrzymamy fałszywe informacje.

W żądaniu HTTP użytkownik musi ustawić metodę POST, wskazać ścieżkę na /api/solidityFiles oraz analogicznie do sekcji wyżej, ustawić wartość *Content-Length* w zależności od długości przesyłanych danych.

Po pomyślnym przesłaniu kontraktu, w odpowiedzi od serwera utrzymujemy status HTTP 200, informujący nas, że wszystko poszło pomyślnie. Następnie w informacji zwrotnej użytkownik otrzymuje przesłany przez niego kod źródłowy, hasz stworzony na podstawie kodu źródłowego oraz listę znalezionych w nim funkcji wraz z akcesoriami generowanymi podczas kompilacji. Na listingu 3.2 można zaobserwować przykładowe dane zwracane przez API po dodaniu nowego kodu źródłowego.

Listing 3.2: Przykładowa odpowiedz w formacie JSON(w skróconej formie)

```
{
    "sourceCodeHash": "0
       x80b739cbf3e89eeea1a96d9cfcf0567ddfef2af82eb14d2c5f97862f71e56265
       ",
    "sourceCode": "pragma solidity ^0.4.25;\n\ncontract Hello {\n\n
       \tstring public message;\n\t\n\tconstructor(string
       initialMessage) public {\n\t\tmessage = initialMessage;\n\t
       }\n\t\n\tfunction setMessage(string newMessage) public {\n\
       t\tmessage = newMessage;\n\t}\n\n}",
    "solidityFunctions": [
        {
            "selector": "e21f37ce",
            "signature": "message()"
        },
        {
            "selector": "368b8772",
            "signature": "setMessage(string)"
        }
    ]
}
```

3.2 Architektura

Literatura: [2, ?]. TODO

3.2.1 Wyszukiwanie sygnatur funkcji w kodzie źródłowym

Literatura: [2, ?]. TODO

3.2.2 Wyszukiwanie selektorów funkcji w kodzie bajtowym

Literatura: [2, ?]. TODO

3.2.3 Szukanie implementacji na podstawie kodu bajtowego

Literatura: [2, ?]. TODO

3.3 Wykorzystane technologie

Literatura: [2, ?]. TODO

Bibliografia

[1] Network Working Group `https://www.ietf.org/rfc/rfc2616.txt`.

[2] Bibliografia 2. *Nazwa*.

Spis tabel

Spis rysunków

Spis listingów