October 2023

METHOD AND SYSTEM TO AUTOMATICALLY SYNTHESIZE SMART CONTRACTS USING TRANSACTION TRACES

RANJIT KUMARESAN
VISA

“METHOD AND SYSTEM TO AUTOMATICALLY SYNTHESIZE SMART CONTRACTS USING TRANSACTION TRACES”

VISA

INVENTOR:

RANJIT KUMARESAN

TECHNICAL FIELD

[0001] The present subject matter is, in general, related to data analysis techniques, and more particularly, but not exclusively, to a method and a system to automatically synthesize smart contracts using transaction traces.

SUMMARY

[0002] Decentralized Finance (DeFi) is a financial system and ecosystem which operates on blockchain technology, with a primary goal of creating an open and decentralized alternative to Traditional Financial (TradFi) intermediaries such as banks, brokerages, auction, lottery and exchanges. Smart contracts have become a fundamental layer of the DeFi architecture. DeFi applications and platforms are built on blockchain networks, such as Ethereum. Etherscan is a widely used blockchain explorer and repository which provides access to various blockchain data, including transaction details and smart contract source code. Smart contracts define the rules and logic of DeFi protocols.

[0003] DeFi operates on public blockchain networks, providing a high degree of transparency. In contrast, Traditional Finance (TradFi) and Centralized Finance (CeFi) systems often have limitations regarding transparency. That is, traditional and centralized financial systems involve multiple intermediaries, such as banks. Composability is a key feature of DeFi which enables the building of complex financial applications by combining and reusing existing DeFi protocols and smart contracts. The DeFi security landscape includes Miner Extractable Value (MEV), wherein miners may strategically position and prioritize transactions to maximize their own profit, often at the expense of users and other market participants. However, MEV has negative externalities on blockchain networks, causing delays in consensus and introducing instability due to network congestion, market manipulation, and security risks. Further, DeFi attacks and MEV incidents require significant storage resources due to the extensive historical data involved. Hence, the concept of reverse engineering attacks in real-time is a necessity for identifying vulnerabilities in DeFi protocols and smart contracts, simulating attacks, and ensuring the integrity of decentralized financial services.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] The accompanying drawings, which are incorporated in and constitute a part of this disclosure, illustrate exemplary embodiments and, together with the description, explain the disclosed principles. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The same numbers are used throughout the figures to reference like features and components. Some embodiments of device or system and/or methods in accordance with embodiments of the present subject matter are now described, by way of example only, and with reference to the accompanying figures, in which:

[0005] FIG. 1 illustrates an architecture of an exemplary system which may be configured to automatically synthesize smart contracts using transaction traces, in accordance with some embodiments of the present disclosure.

[0006] FIG. 2 illustrates an exemplary flow diagram of a method of automatically synthesizing smart contracts using transaction traces, in accordance with some embodiments of the present disclosure.

[0007] FIG. 3 shows an exemplary raw transaction data, in accordance with some embodiments of the present disclosure.

[0008] FIG. 4 shows an exemplary debug trace output for Ethereum transactions, in accordance with some embodiments of the present disclosure.

[0009] FIG. 5 shows an exemplary call frame in Ethereum smart contract execution, in accordance with some embodiments of the present disclosure.

[0010] FIG. 6 shows an exemplary synthesized representation of the target smart contract, in accordance with some embodiments of the present disclosure.

[0011] FIG. 7 is a block diagram of an exemplary computer system for implementing embodiments consistent with the present disclosure.

[0012] The figures depict embodiments of the disclosure for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the disclosure described herein.

DESCRIPTION OF THE DISCLOSURE

[0013] It is to be understood that the present disclosure may assume various alternative variations and step sequences, except where expressly specified to the contrary. It is also to be understood that the specific devices and processes illustrated in the attached drawings and described in the following specification are simply exemplary and non-limiting embodiments or aspects. Hence, specific dimensions and other physical characteristics related to the embodiments or aspects disclosed herein are not to be considered as limiting.

[0014] In the present document, the word "exemplary" is used herein to mean "serving as an example, instance, or illustration." Any embodiment or implementation of the present subject matter described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other embodiments.

[0015] While the disclosure is susceptible to various modifications and alternative forms, specific embodiment thereof has been shown by way of example in the drawings and will be described in detail below. It should be understood, however that it is not intended to limit the disclosure to the particular forms disclosed, but on the contrary, the disclosure is to cover all modifications, equivalents, and alternative falling within the spirit and the scope of the disclosure.

[0016] The terms “comprises”, “comprising”, or any other variations thereof, are intended to cover a non-exclusive inclusion, such that a setup, device, or method that comprises a list of components or steps does not include only those components or steps but may include other components or steps not expressly listed or inherent to such setup or device or method. In other words, one or more elements in a device or system or apparatus proceeded by “comprises… a” does not, without more constraints, preclude the existence of other elements or additional elements in the device or system or apparatus.

[0017] The terms "an embodiment", "embodiment", "embodiments", "the embodiment", "the embodiments", "one or more embodiments", "some embodiments", and "one embodiment" mean "one or more (but not all) embodiments of the invention(s)" unless expressly specified otherwise.

[0018] The terms "including", "comprising", “having” and variations thereof mean "including but not limited to" unless expressly specified otherwise.

[0019] As used herein, the terms “communication” and “communicate” may refer to the reception, receipt, transmission, transfer, provision, and/or the like of information (e.g., data, signals, messages, instructions, commands, and/or the like). For one unit (e.g., a device, a system, a component of a device or system, combinations thereof, and/or the like) to be in communication with another unit means that the one unit can receive information directly or indirectly from and/or transmit information to the other unit. This may refer to a direct or indirect connection (e.g., a direct communication connection, an indirect communication connection, and/or the like) that is wired and/or wireless in nature. Additionally, two units may be in communication with each other even though the information transmitted may be modified, processed, relayed, and/or routed between the first and second unit. For example, a first unit may be in communication with a second unit even though the first unit passively receives information and does not actively transmit information to the second unit. As another example, a first unit may be in communication with a second unit if at least one intermediary unit (e.g., a third unit located between the first unit and the second unit) processes information received from the first unit and communicates the processed information to the second unit. In some non-limiting embodiments, a message may refer to a network packet (e.g., a data packet and/or the like) that includes data. It will be appreciated that numerous other arrangements are possible.

[0020] As used herein, the term “computing device” may refer to one or more electronic devices that are configured to communicate with directly or indirectly or over one or more networks. A computing device may be a mobile or portable computing device, a desktop computer, a server, and/or the like. Furthermore, the term “computer” may refer to any computing device that includes the necessary components to receive, process, and output data, and normally includes a display, a processor, a memory, an input device, and a network interface. A “computing system” may include one or more computing devices or computers.

[0021] As used herein, the term “Decentralized Finance” or “DeFi” may refer to a blockchain-based form of finance that does not rely on central financial intermediaries such as brokerages, exchanges, or banks to offer traditional financial instruments.

[0022] As used herein, the term “smart contract” may refer to a computer program or a transaction protocol that automatically executes, controls, or documents events and actions according to terms of a contract or an agreement.

[0023] FIG. 1 illustrates an exemplary environment 100 of a transaction decoder system 103, which is configured to automatically synthesize smart contracts using transaction trace data. The transaction trace data refers to detailed history of operations, which occur during the execution of a transaction on a blockchain, particularly in the context of Ethereum or similar smart contract platforms. In an embodiment, environment 100 comprises, without limitation, an input data unit 101, the transaction decoder system 103, and an output data unit 105. The input data unit 101 and the transaction decoder system 103 may be connected via a predefined communication network (not shown in **FIG. 1**). Such a communication network may include, without limitation, a direct interconnection, Local Area Network (LAN), Wide Area Network (WAN), wireless network (e.g., using Wireless Application Protocol), the Internet, and the like.

[0024] In an implementation, the transaction decoder system 103 may include one or more processors 107, an I/O interface 109, and a memory 111. In some embodiments, the memory 111 may be communicatively coupled to the one or more processors 107. The memory 111 may be configured to store instructions, executable by the one or more processors 107, which on execution, may cause the transaction decoder system 103 to automatically synthesize smart contracts using transaction trace data. In an embodiment, the memory 111 may include one or more modules 113 and data 115. The one or more modules 113 may be configured to perform the steps of the present disclosure using the data 115. In an embodiment, each of the one or more modules 113 may be a hardware unit, which may be present outside the memory 111 and externally coupled with the transaction decoder system 103. The transaction decoder system 103 may be implemented in a variety of computing systems, such as a laptop computer, a desktop computer, a Personal Computer (PC), a notebook, a smartphone, a tablet, e-book readers, a server, a network server, a cloud-based server and the like. In an embodiment, the transaction decoder system 103 may be a dedicated server or may be a cloud-based server.

[0025] Consider a scenario when a user/customer wishes to identify vulnerabilities in the transaction trace data. The transaction trace data records the sequence of operations, including calls to smart contracts, changes in account balances, and state modifications. That is, for each transaction, the trace data identifies which smart contracts are involved and how they interact. Input data unit 101 may be utilized to collect raw transaction trace data related to the step-by-step execution of a transaction involving smart contracts and accounts on a blockchain. The raw transaction trace data includes comprehensive information about the transaction, bytecode execution, contract interactions and changes to the contract’s state. The input data unit 101

transmits the transaction trace raw data obtained to the transaction decoder system 103 for initial data preprocessing. The transaction decoder system 103 analyzes the transaction trace raw data to extract relevant information (such as call frames) for contract synthesis. Thereafter, the contract address in each call frame is identified and it is verified that the contract address being called is known or unknown to Etherscan. If the contract address is unknown to Etherscan, data related to top-level call frames are collected within child frames. Subsequently, the collected top-level call frames are added to the contract as function definitions. Further, output data unit 105 receives a synthesized smart contract, which includes known contracts and functions as well as newly created function definitions for previously unknown contracts. As a result, automatic contract synthesis simplifies contract analysis and enhances security issues within the contract code.

[0026] FIG. 2 illustrates an exemplary flow diagram of a method of automatically synthesizing smart contracts using transaction traces, in accordance with some embodiments of the present disclosure.

[0027] In an embodiment, at block 201, the method comprises collecting a transaction trace raw data, by an input data unit 101, from a blockchain network. The transaction trace raw data is generated in real-time as a transaction is executed on the blockchain as shown in **FIG. 3**. Blockchains store and maintain data of all transactions, wherein the data includes details such as sender and recipient addresses, the amount of cryptocurrency transferred, and any input data provided with the transaction. Blockchain stores only the bytecode of smart contracts. Bytecode represents the low-level, machine-readable code of smart contracts and contains the instructions that the Ethereum Virtual Machine (EVM) executes when interacting with a smart contract. Thereafter, in block 203, the method comprises transmitting the transaction trace raw data obtained from the input data unit 101 to a transaction decoder system 103 for initial data preprocessing.

[0028] In an embodiment, at block 205, the method comprises analyzing the transaction trace raw data to identify and extract the call frames for contract synthesis. The transaction decoder system 103 analyzes the sequence of calls and interaction between contracts with the Remote Procedure Call (RPC) call's debug trace transaction. RPC call debug transaction trace provides an in-depth view of how a transaction interacts with smart contracts. RPC call debug transaction trace data includes detailed information related to the execution of the transaction, the sequence of calls, and input data. Thereafter, individual call frames containing contract

addresses (such as “to” address) are extracted from the transaction trace raw data, wherein each call frame indicates an interaction with the smart contract during the execution of the transaction. Each call frame includes information such as the “to” address and/or contract address, input data, and executed context, as shown in **FIG. 4**. For example, extracted callFrame of <target>.0x1234 (as shown in **FIG. 4** and as indicated by the blue arrow in **FIG. 5**) includes:

(1.1) <compound>.add_collateral

(1.2) <uniswap>.flashSwap

(1.3) <aave>.flashLoan

(1.4*)* <unknown>.executeNext

(1.5) <unknown2>.finalize

Similarly, other top-level call frame functions in <target> smart contract includes:

(1) <target>.0x1234

(1.2.1) <target>.callFunction

(1.3.1) <target>.onFlashLoan

[0029] In an embodiment, at block 207, the method comprises performing a Depth-First Search (DFS) traversal on the extracted call frames. The DFS traversal analyzes the order and hierarchy of call frames and interactions between smart contracts. Subsequently, in block 209, the method comprises verifying the contract addresses containing the “to” address in each call frame using Etherscan data. In other words, at each step of the DFS traversal, the contract address (“to” address) associated with a call frame is verified to determine if the contract address is known or unknown to Etherscan or blockchain data service.

[0030] In an embodiment, at block 211, the method comprises collecting top-level calls within the child call frames when the “to” address is unknown in the call frame. If the contract address (“to” address) of the call frame is unknown, it means that the contract may not be publicly documented or verified. The top-level calls represent external function calls made to unknown contracts. Finally, as indicated in block 213, the collected top-level calls are added to the contract as function definitions. That is, data related to top-level call frames, including function names, input parameters, and other relevant details are collected. For example, the top-level call frames include:

(-) <compound>.add_collateral

(-) <uniswap>.flashSwap

(-) <aave>.flashLoan

(-) <unknown>.executeNext

(-) <unknown2>.finalize

[0031] As a result, the automatic contract synthesis from a transaction trace raw data generates a synthesized smart contract or a modified existing contract as shown in **FIG. 6**. The synthesized smart contract includes known contracts and newly created function definitions.

Advantages of the present invention:

[0032] In an embodiment, the method disclosed in the present disclosure has the ability to understand and document unknown contract addresses, which may not be publicly recognized by Etherscan.

[0033] In an embodiment, the method disclosed in the present disclosure helps to identify vulnerabilities in smart contracts, even if the contracts involved are not publicly available.

[0034] In an embodiment, the method disclosed in the present disclosure helps in analyzing and understanding the behavior of complex financial contracts, for example, decentralized exchanges.

[0035] In an embodiment, the method disclosed in the present disclosure helps in detecting and addressing errors in contract execution by analyzing transaction traces and contract interactions, leading to improved contract reliability.

[0036] In an embodiment, the method disclosed in the present disclosure simplifies contract analysis, enhances security, and integration of blockchain-based applications.

General computer system:

[0037] FIG. 7 illustrates a block diagram of an exemplary computer system for implementing embodiments consistent with the present disclosure.

[0038] In an embodiment, **FIG. 7** illustrates a block diagram of an exemplary computer system 700 which may be used to implement the transaction decoder system 103 for creating inferred attributes using spend bands and/or basket size. In an embodiment, the computer system 700 may include a central processing unit (“CPU” or “processor”) 702. The processor 702 may include at least one data processor for executing processes in Virtual Storage Area Network.

The processor 702 may include at least one data processor for executing program components for executing user or system-generated business processes. The processor 702 may include specialized processing units such as integrated system (bus) controllers, memory management control units, floating point units, graphics processing units, digital signal processing units, etc.

[0039] The processor 702 may be disposed in communication with one or more Input/Output (I/O) devices (712 and 713) via I/O interface 701. The I/O interface 701 employ communication protocols/methods such as, without limitation, audio, analog, digital, monoaural, Radio Corporation of America (RCA) connector, stereo, IEEE-1394 high-speed serial bus, serial bus, Universal Serial Bus (USB), infrared, Personal System/2 (PS/2) port, Bayonet Neill-Concelman (BNC) connector, coaxial, component, composite, Digital Visual Interface (DVI), High-Definition Multimedia Interface (HDMI), Radio Frequency (RF) antennas, S-Video, Video Graphics Array (VGA), IEEE 802.11b/g/n/x, Bluetooth, cellular, for example, Code-Division Multiple Access (CDMA), High-Speed Packet Access (HSPA+), Global System for Mobile communications (GSM), Long-Term Evolution (LTE), Worldwide Interoperability for Microwave access (WiMax), or the like, etc.

[0040] Using the I/O interface 701, the computer system 700 may communicate with one or more I/O devices such as input devices 712 and output devices 713. For example, the input devices 712 may be an antenna, keyboard, mouse, joystick, (infrared) remote control, camera, card reader, fax machine, dongle, biometric reader, microphone, touch screen, touchpad, trackball, stylus, scanner, storage device, transceiver, video device/source, etc. The output devices 713 may be a printer, fax machine, video display (e.g., Cathode Ray Tube (CRT), Liquid Crystal Display (LCD), Light-Emitting Diode (LED), plasma, Plasma Display Panel (PDP), Organic Light-Emitting Diode display (OLED) or the like), audio speaker, etc.

[0041] In some embodiments, the processor 702 may be disposed in communication with a communication network 709 via a network interface 703. The network interface 703 may communicate with the communication network 709. The network interface 703 may employ connection protocols including, without limitation, direct connect, Ethernet (for example, twisted pair 10/100/1000 Base T), Transmission Control Protocol/Internet Protocol (TCP/IP), token ring, IEEE 802.11a/b/g/n/x, etc. The communication network 709 may include, without limitation, a direct interconnection, Local Area Network (LAN), Wide Area Network (WAN), wireless network (for example, using Wireless Application Protocol), the Internet, etc. Using the network interface 703 and the communication network 709, the computer system 700 may

communicate with a database 714, which may be the enrolled templates database 713. The network interface 703 may employ connection protocols include, but not limited to, direct connect, ethernet (e.g., twisted pair 10/100/1000 Base T), Transmission Control Protocol/Internet Protocol (TCP/IP), token ring, IEEE 802.11a/b/g/n/x, etc.

[0042] The communication network 709 includes, but is not limited to, a direct interconnection, a Peer-to-Peer (P2P) network, Local Area Network (LAN), Wide Area Network (WAN), wireless network (for example, using Wireless Application Protocol), the Internet, Wi-Fi, and such. The communication network 709 may either be a dedicated network or a shared network, which represents an association of the different types of networks that use a variety of protocols, for example, Hypertext Transfer Protocol (HTTP), Transmission Control Protocol/Internet Protocol (TCP/IP), Wireless Application Protocol (WAP), etc., to communicate with each other. Further, the communication network 709 may include a variety of network devices, including routers, bridges, servers, computing devices, storage devices, etc.

[0043] In some embodiments, the processor 702 may be disposed of in communication with a memory 705 (for example, RAM, ROM, etc. not shown in FIG. 7) via a storage interface 704. The storage interface 704 may connect to memory 705 including, without limitation, memory drives, removable disc drives, etc., employing connection protocols such as, Serial Advanced Technology Attachment (SATA), Integrated Drive Electronics (IDE), IEEE-1394, Universal Serial Bus (USB), fiber channel, Small Computer Systems Interface (SCSI), etc. The memory drives may further include a drum, magnetic disc drive, magneto-optical drive, optical drive, Redundant Array of Independent Discs (RAID), solid-state memory devices, solid-state drives, etc.

[0044] In some embodiments, the memory 705 may store a collection of program or database components, including, without limitation, user interface 706, an operating system 707, a web browser 708 etc. In some embodiments, computer system 700 may store user/application data, such as, the data, variables, records, etc., as described in this disclosure. Such databases may be implemented as fault-tolerant, relational, scalable, secure databases such as Oracle or Sybase.

[0045] In some embodiments, the operating system 707 may facilitate resource management and operation of the computer system 700. Examples of operating systems include, without

limitation, Apple Macintosh OS X™, UNIX™, Unix-like system distributions (e.g., Berkeley Software Distribution (BSD), FreeBSD, Net BSD™, Open BSD™, etc.), Linux distributions (e.g., Red Hat, Ubuntu, K-Ubuntu, etc.), International Business Machines (IBM™) OS/2™, Microsoft Windows (XP™, Vista/7/8, etc.), Apple iOS, Google Android, BlackBerry operating system (OS), or the like. The User Interface 706 may facilitate display, execution, interaction, manipulation, or operation of program components through textual or graphical facilities. For example, user interfaces may provide computer interaction interface elements on a display system operatively connected to the computer system 700, such as cursors, icons, checkboxes, menus, scrollers, windows, widgets, etc. Graphical User Interfaces (GUIs) may be employed, including, without limitation, Apple® Macintosh® operating systems' Aqua®, IBM® OS/2®, Microsoft® Windows® (e.g., Aero, Metro, etc.), web interface libraries (e.g., ActiveX®, Java®, JavaScript®, AJAX, HTML, Adobe® Flash®, etc.), or the like.

[0046] In some embodiments, the computer system 700 may implement web browser 708 stored program components. Web browser 708 may be a hypertext viewing application, such as Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, Apple Safari, etc. Secure web browsing may be provided using secure hypertext transport protocol (HTTPS), Secure Sockets Layer (SSL), Transport Layer Security (TLS), etc. Web browsers 708 may utilize facilities such as AJAX, DHTML, Adobe Flash, JavaScript, Application Programming Interfaces (APIs), etc.

[0047] In some embodiments, the computer system 700 may implement a mail server stored program component. The mail server may be an Internet mail server such as Microsoft Exchange, or the like. The mail server may utilize facilities such as ASP, ActiveX, ANSI C++/C#, Microsoft .NET, Common Gateway Interface (CGI) scripts, Java, JavaScript, PERL, PHP, Python, WebObjects, etc. The mail server may utilize communication protocols such as Internet Message Access Protocol (IMAP), Messaging Application Programming Interface (MAPI), Microsoft Exchange, Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), or the like.

[0048] In some embodiments, the computer system 700 may implement a mail client stored program component. The mail client may be a mail viewing application, such as APPLE® MAIL, MICROSOFT® ENTOURAGE®, MICROSOFT® OUTLOOK®, MOZILLA® THUNDERBIRD®, etc.

[0049] Furthermore, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer-readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer-readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term "computer-readable medium" should be understood to include tangible items and exclude carrier waves and transient signals, i.e., be non-transitory. Examples include Random Access Memory (RAM), Read-Only Memory (ROM), volatile memory, non-volatile memory, hard drives, Compact Disc (CD) ROMs, DVDs, flash drives, disks, and any other known physical storage media.

[0050] The described operations may be implemented as a method, system or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The described operations may be implemented as code maintained in a "non-transitory computer-readable medium", where a processor may read and execute the code from the computer-readable medium. The processor is at least one of a microprocessor and a processor capable of processing and executing the queries. A non-transitory computer-readable medium may include media such as magnetic storage medium (e.g., hard disk drives, floppy disks, tape, etc.), optical storage (CD-ROMs, DVDs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, Flash Memory, firmware, programmable logic, etc.), etc. Further, non-transitory computer-readable media may include all computer-readable media except for transitory. The code implementing the described operations may further be implemented in hardware logic (e.g., an integrated circuit chip, Programmable Gate Array (PGA), Application Specific Integrated Circuit (ASIC), etc.).

[0051] The illustrated steps are set out to explain the exemplary embodiments shown, and it should be anticipated that ongoing technological development will change the manner in which particular functions are performed. These examples are presented herein for purposes of illustration, and not limitation. Further, the boundaries of the functional building blocks have been arbitrarily defined herein for the convenience of the description. Alternative boundaries can be defined so long as the specified functions and relationships thereof are appropriately performed. Alternatives (including equivalents, extensions, variations, deviations, etc., of those

described herein) will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein. Such alternatives fall within the scope and spirit of the disclosed embodiments. Also, the words "comprising," "having," "containing," and "including," and other similar forms are intended to be equivalent in meaning and be open ended in that an item or items following any one of these words is not meant to be an exhaustive listing of such item or items or meant to be limited to only the listed item or items. It must also be noted that as used herein, the singular forms “a,” “an,” and “the” include plural references unless the context clearly dictates otherwise.

[0052] Furthermore, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer-readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer-readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term “computer-readable medium” should be understood to include tangible items and exclude carrier waves and transient signals, i.e., are non-transitory. Examples include Random Access Memory (RAM), Read-Only Memory (ROM), volatile memory, non-volatile memory, hard drives, CD ROMs, DVDs, flash drives, disks, and any other known physical storage media.

[0053] Finally, the language used in the specification has been principally selected for readability and instructional purposes, and it may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the disclosure of the embodiments of the disclosure is intended to be illustrative, but not limiting, of the scope of the disclosure.

[0054] With respect to the use of substantially any plural and/or singular terms herein, those having skill in the art can translate from the plural to the singular and/or from the singular to the plural as is appropriate to the context and/or application. The various singular/plural permutations may be expressly set forth herein for sake of clarity.

“METHOD AND SYSTEM TO AUTOMATICALLY SYNTHESIZE SMART CONTRACTS USING TRANSACTION TRACES”

ABSTRACT

The present disclosure relates to a method and system to automatically synthesize smart contracts using transaction traces. The present disclosure suggests collecting transaction trace raw data from a blockchain network and transmitting the transaction trace raw data obtained to a transaction decoder system for initial data preprocessing. Thereafter, call frames are extracted from the transaction trace raw data and a Depth-First Search (DFS) traversal is performed on the extracted call frames. Subsequently, at each step of the DFS traversal, the contract address associated (that is, “to” address) with the extracted call frame is verified to determine if it is known or unknown to Etherscan data. Further, the present disclosure suggests collecting top-level calls within the child call frames when the contract address is unknown in the call frame and adding them to the contract as function definitions.

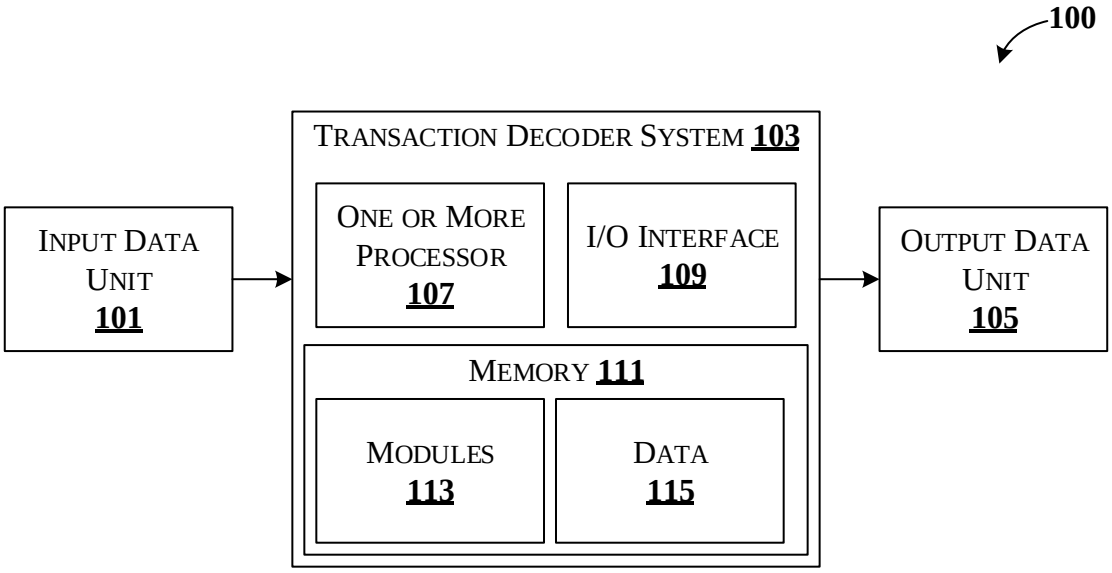
100
TRANSACTION DECODER SYSTEM 103
INPUT DATA UNIT 101
ONE OR MORE PROCESSOR 107
I/O INTERFACE 109
OUTPUT DATA UNIT 105
MEMORY 111
MODULES 113
DATA 115

FIG. 1

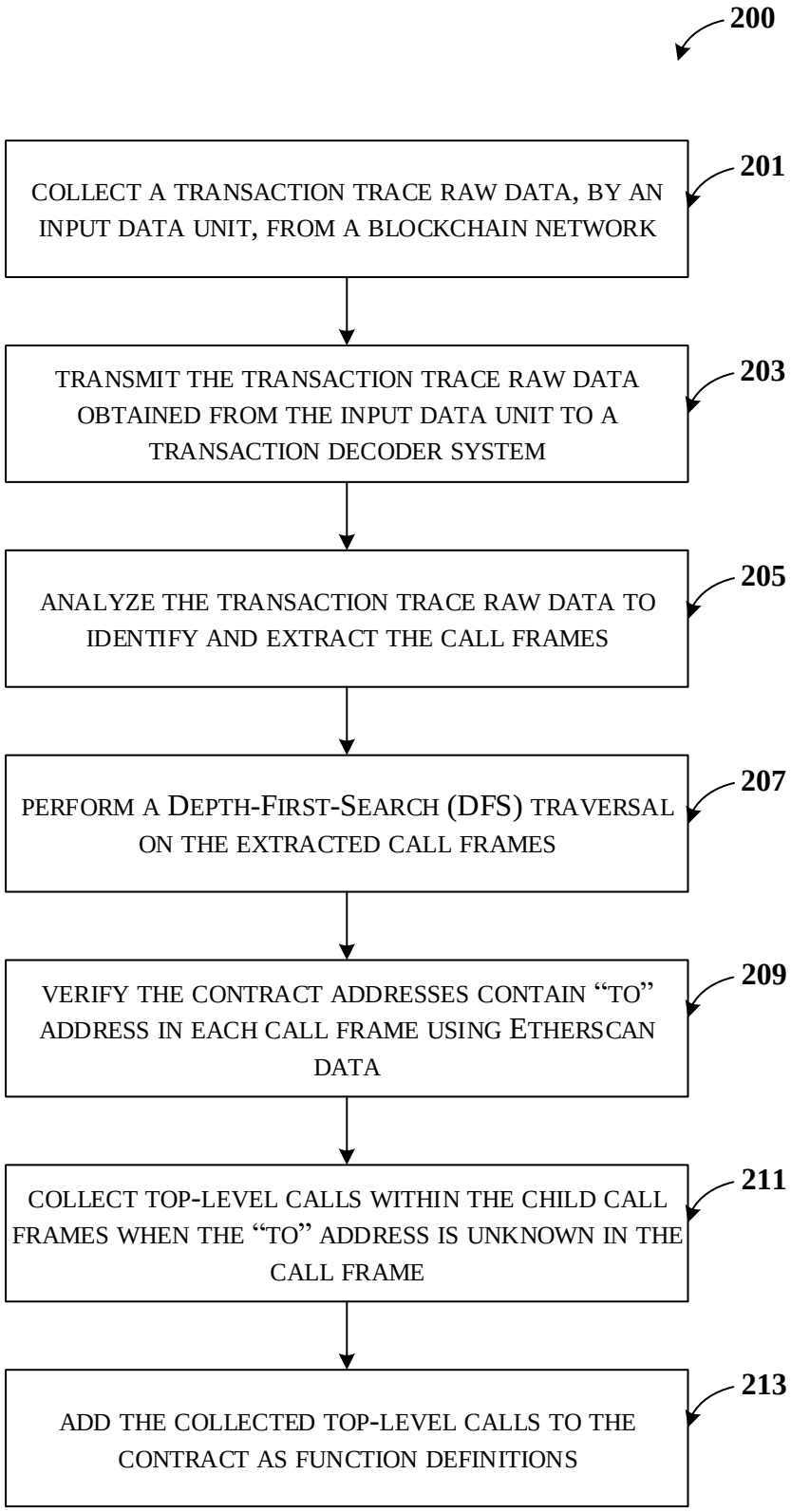
200
COLLECT A TRANSACTION TRACE RAW DATA, BY AN INPUT DATA UNIT, FROM A BLOCKCHAIN NETWORK
201
TRANSMIT THE TRANSACTION TRACE RAW DATA OBTAINED FROM THE INPUT DATA UNIT TO A TRANSACTION DECODER SYSTEM
203
ANALYZE THE TRANSACTION TRACE RAW DATA TO IDENTIFY AND EXTRACT THE CALL FRAMES
205
PERFORM A DEPTH-FIRST-SEARCH (DFS) TRAVERSAL ON THE EXTRACTED CALL FRAMES
207
VERIFY THE CONTRACT ADDRESSES CONTAIN "TO" ADDRESS IN EACH CALL FRAME USING ETHERSCAN DATA
209
COLLECT TOP-LEVEL CALLS WITHIN THE CHILD CALL FRAMES WHEN THE "TO" ADDRESS IS UNKNOWN IN THE CALL FRAME
211
ADD THE COLLECTED TOP-LEVEL CALLS TO THE CONTRACT AS FUNCTION DEFINITIONS
213

FIG. 2

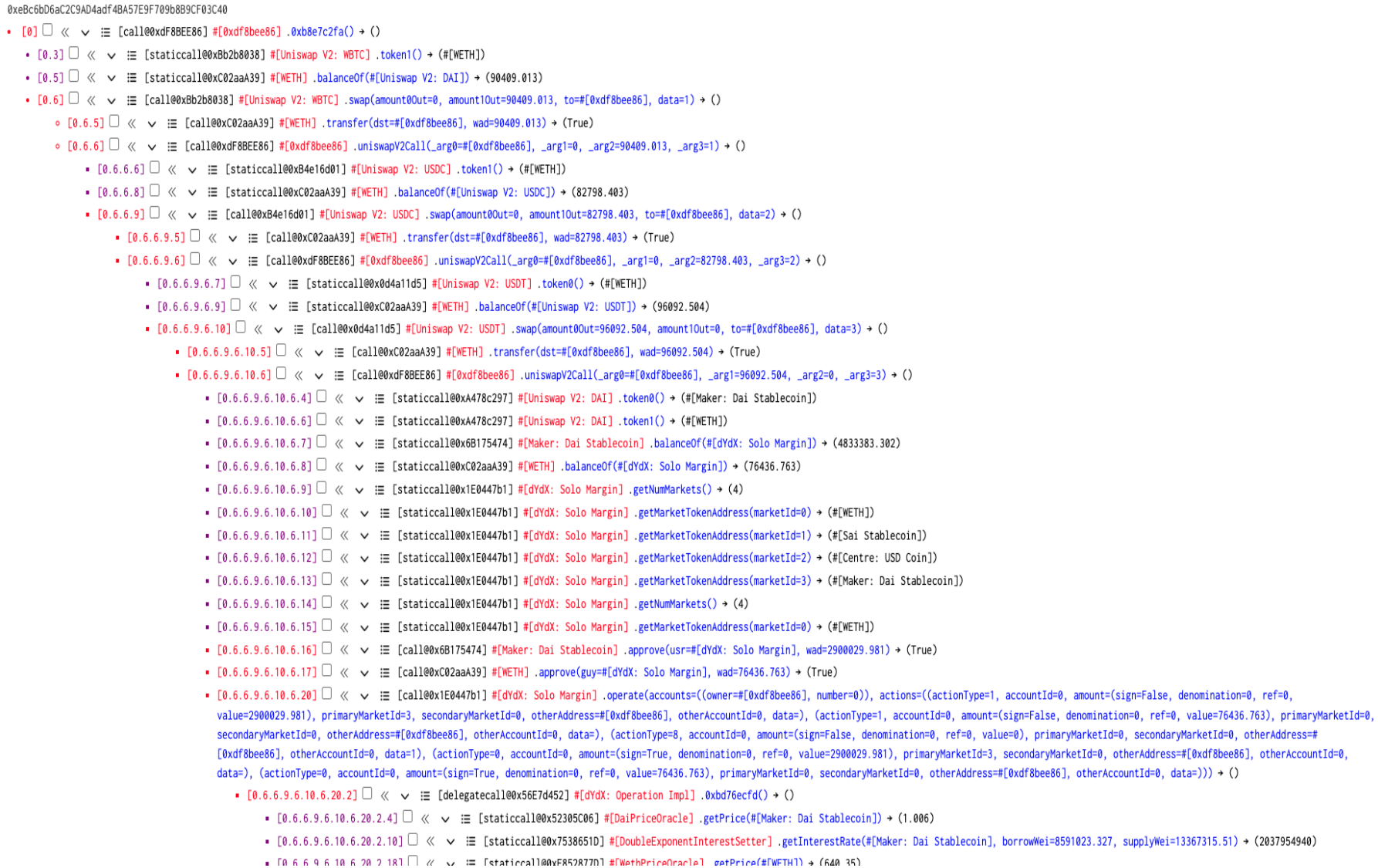
0xeBc6bD6aC2C9AD4adf4BA57E9F709b8B9CF03C40
[0] [call@0xdF8BEE86] #[0xdf8bee86] .0xb8e7c2fa() → ()
[0.3] [staticcall@0xBb2b8038] #[Uniswap V2: WBTC] .token1() → (#[WETH])
[0.5] [staticcall@0xC02aaA39] #[WETH] .balanceOf(#[Uniswap V2: DAI]) → (90409.013)
[0.6] [call@0xBb2b8038] #[Uniswap V2: WBTC] .swap(amount0Out=0, amount1Out=90409.013, to=#[0xdf8bee86], data=1) → ()
[0.6.5] [call@0xC02aaA39] #[WETH] .transfer(dst=#[0xdf8bee86], wad=90409.013) → (True)
[0.6.6] [call@0xdF8BEE86] #[0xdf8bee86] .uniswapV2Call(_arg0=#[0xdf8bee86], _arg1=0, _arg2=90409.013, _arg3=1) → ()
[0.6.6.6] [staticcall@0xB4e16d01] #[Uniswap V2: USDC] .token1() → (#[WETH])
[0.6.6.8] [staticcall@0xC02aaA39] #[WETH] .balanceOf(#[Uniswap V2: USDC]) → (82798.403)
[0.6.6.9] [call@0xB4e16d01] #[Uniswap V2: USDC] .swap(amount0Out=0, amount1Out=82798.403, to=#[0xdf8bee86], data=2) → ()
[0.6.6.9.5] [call@0xC02aaA39] #[WETH] .transfer(dst=#[0xdf8bee86], wad=82798.403) → (True)
[0.6.6.9.6] [call@0xdF8BEE86] #[0xdf8bee86] .uniswapV2Call(_arg0=#[0xdf8bee86], _arg1=0, _arg2=82798.403, _arg3=2) → ()
[0.6.6.9.6.7] [staticcall@0x0d4a11d5] #[Uniswap V2: USDT] .token0() → (#[WETH])
[0.6.6.9.6.9] [staticcall@0xC02aaA39] #[WETH] .balanceOf(#[Uniswap V2: USDT]) → (96092.504)
[0.6.6.9.6.10] [call@0x0d4a11d5] #[Uniswap V2: USDT] .swap(amount0Out=96092.504, amount1Out=0, to=#[0xdf8bee86], data=3) → ()
[0.6.6.9.6.10.5] [call@0xC02aaA39] #[WETH] .transfer(dst=#[0xdf8bee86], wad=96092.504) → (True)
[0.6.6.9.6.10.6] [call@0xdF8BEE86] #[0xdf8bee86] .uniswapV2Call(_arg0=#[0xdf8bee86], _arg1=96092.504, _arg2=0, _arg3=3) → ()
[0.6.6.9.6.10.6.4] [staticcall@0xA478c297] #[Uniswap V2: DAI] .token0() → (#[Maker: Dai Stablecoin])
[0.6.6.9.6.10.6.6] [staticcall@0xA478c297] #[Uniswap V2: DAI] .token1() → (#[WETH])
[0.6.6.9.6.10.6.7] [staticcall@0x6B175474] #[Maker: Dai Stablecoin] .balanceOf(#[dYdX: Solo Margin]) → (4833383.302)
[0.6.6.9.6.10.6.8] [staticcall@0xC02aaA39] #[WETH] .balanceOf(#[dYdX: Solo Margin]) → (76436.763)
[0.6.6.9.6.10.6.9] [staticcall@0x1E0447b1] #[dYdX: Solo Margin] .getNumMarkets() → (4)
[0.6.6.9.6.10.6.10] [staticcall@0x1E0447b1] #[dYdX: Solo Margin] .getMarketTokenAddress(marketId=0) → (#[WETH])
[0.6.6.9.6.10.6.11] [staticcall@0x1E0447b1] #[dYdX: Solo Margin] .getMarketTokenAddress(marketId=1) → (#[Sai Stablecoin])
[0.6.6.9.6.10.6.12] [staticcall@0x1E0447b1] #[dYdX: Solo Margin] .getMarketTokenAddress(marketId=2) → (#[Centre: USD Coin])
[0.6.6.9.6.10.6.13] [staticcall@0x1E0447b1] #[dYdX: Solo Margin] .getMarketTokenAddress(marketId=3) → (#[Maker: Dai Stablecoin])
[0.6.6.9.6.10.6.14] [staticcall@0x1E0447b1] #[dYdX: Solo Margin] .getNumMarkets() → (4)
[0.6.6.9.6.10.6.15] [staticcall@0x1E0447b1] #[dYdX: Solo Margin] .getMarketTokenAddress(marketId=0) → (#[WETH])
[0.6.6.9.6.10.6.16] [call@0x6B175474] #[Maker: Dai Stablecoin] .approve(usr=#[dYdX: Solo Margin], wad=2900029.981) → (True)
[0.6.6.9.6.10.6.17] [call@0xC02aaA39] #[WETH] .approve(guy=#[dYdX: Solo Margin], wad=76436.763) → (True)
[0.6.6.9.6.10.6.20] [call@0x1E0447b1] #[dYdX: Solo Margin] .operate(accounts=((owner=#[0xdf8bee86], number=0)), actions=((actionType=1, accountId=0, amount=(sign=False, denomination=0, ref=0, value=2900029.981), primaryMarketId=3, secondaryMarketId=0, otherAddress=#[0xdf8bee86], otherAccountId=0, data=), (actionType=1, accountId=0, amount=(sign=False, denomination=0, ref=0, value=76436.763), primaryMarketId=0, secondaryMarketId=0, otherAddress=#[0xdf8bee86], otherAccountId=0, data=), (actionType=8, accountId=0, amount=(sign=False, denomination=0, ref=0, value=0), primaryMarketId=0, secondaryMarketId=0, otherAddress=#[0xdf8bee86], otherAccountId=0, data=1), (actionType=0, accountId=0, amount=(sign=True, denomination=0, ref=0, value=2900029.981), primaryMarketId=3, secondaryMarketId=0, otherAddress=#[0xdf8bee86], otherAccountId=0, data=), (actionType=0, accountId=0, amount=(sign=True, denomination=0, ref=0, value=76436.763), primaryMarketId=0, secondaryMarketId=0, otherAddress=#[0xdf8bee86], otherAccountId=0, data=))) → ()
[0.6.6.9.6.10.6.20.2] [delegatecall@0x56E7d452] #[dYdX: Operation Impl] .0xbd76ecfd() → ()
[0.6.6.9.6.10.6.20.2.4] [staticcall@0x52305C06] #[DaiPriceOracle] .getPrice(#[Maker: Dai Stablecoin]) → (1.006)
[0.6.6.9.6.10.6.20.2.10] [staticcall@0x7538651D] #[DoubleExponentInterestSetter] .getInterestRate(#[Maker: Dai Stablecoin], borrowWei=8591023.327, supplyWei=13367315.51) → (2037954940)

FIG. 3

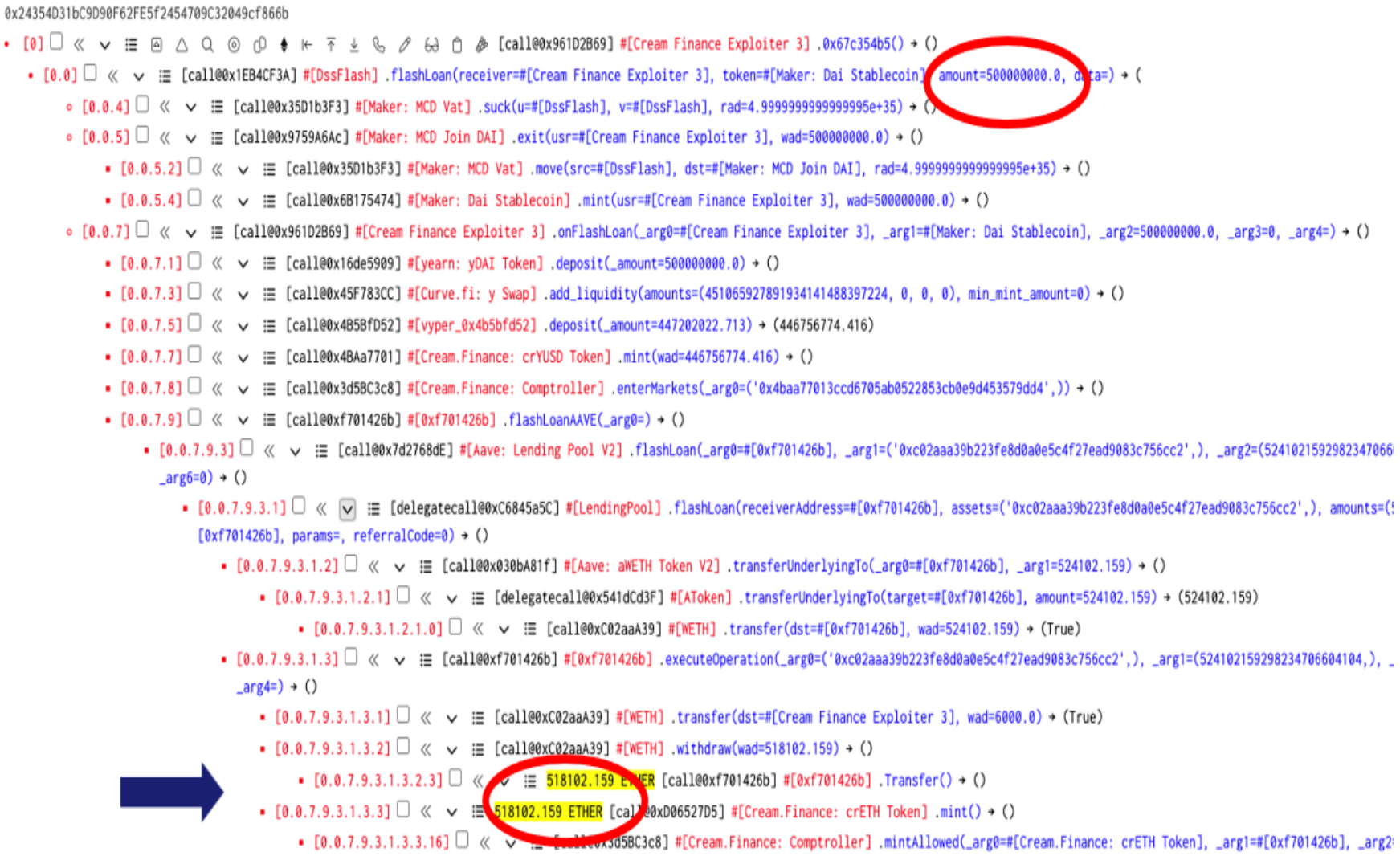
0x24354D31bC9D90F62FE5f2454709C32049cf866b
[0] [call@0x961D2B69] #[Cream Finance Exploiter 3] .0x67c354b5() → ()
[0.0] [call@0x1EB4CF3A] #[DssFlash] .flashLoan(receiver=#[Cream Finance Exploiter 3], token=#[Maker: Dai Stablecoin], amount=500000000.0, data=) → (
[0.0.4] [call@0x35D1b3F3] #[Maker: MCD Vat] .suck(u=#[DssFlash], v=#[DssFlash], rad=4.9999999999999995e+35) → ()
[0.0.5] [call@0x9759A6Ac] #[Maker: MCD Join DAI] .exit(usr=#[Cream Finance Exploiter 3], wad=500000000.0) → ()
[0.0.5.2] [call@0x35D1b3F3] #[Maker: MCD Vat] .move(src=#[DssFlash], dst=#[Maker: MCD Join DAI], rad=4.9999999999999995e+35) → ()
[0.0.5.4] [call@0x6B175474] #[Maker: Dai Stablecoin] .mint(usr=#[Cream Finance Exploiter 3], wad=500000000.0) → ()
[0.0.7] [call@0x961D2B69] #[Cream Finance Exploiter 3] .onFlashLoan(_arg0=#[Cream Finance Exploiter 3], _arg1=#[Maker: Dai Stablecoin], _arg2=500000000.0, _arg3=0, _arg4=) → ()
[0.0.7.1] [call@0x16de5909] #[yearn: yDAI Token] .deposit(_amount=500000000.0) → ()
[0.0.7.3] [call@0x45F783CC] #[Curve.fi: y Swap] .add_liquidity(amounts=(451065927891934141488397224, 0, 0, 0), min_mint_amount=0) → ()
[0.0.7.5] [call@0x4B5BfD52] #[vyper_0x4b5bfd52] .deposit(_amount=447202022.713) → (446756774.416)
[0.0.7.7] [call@0x4BAa7701] #[Cream.Finance: crYUSD Token] .mint(wad=446756774.416) → ()
[0.0.7.8] [call@0x3d5BC3c8] #[Cream.Finance: Comptroller] .enterMarkets(_arg0=('0x4baa77013ccd6705ab0522853cb0e9d453579dd4',)) → ()
[0.0.7.9] [call@0xf701426b] #[0xf701426b] .flashLoanAAVE(_arg0=) → ()
[0.0.7.9.3] [call@0x7d2768dE] #[Aave: Lending Pool V2] .flashLoan(_arg0=#[0xf701426b], _arg1=('0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2',), _arg2=(524102159298234706604104,), _arg3=(0,), _arg4=(0,), _arg5=(0,), _arg6=0) → ()
[0.0.7.9.3.1] [delegatecall@0xC6845a5C] #[LendingPool] .flashLoan(receiverAddress=#[0xf701426b], assets=('0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2',), amounts=(524102159298234706604104,), modes=(0,), onBehalfOf=#[0xf701426b], params=, referralCode=0) → ()
[0.0.7.9.3.1.2] [call@0x030bA81f] #[Aave: aWETH Token V2] .transferUnderlyingTo(_arg0=#[0xf701426b], _arg1=524102.159) → ()
[0.0.7.9.3.1.2.1] [delegatecall@0x541dCd3F] #[AToken] .transferUnderlyingTo(target=#[0xf701426b], amount=524102.159) → (524102.159)
[0.0.7.9.3.1.2.1.0] [call@0xC02aaA39] #[WETH] .transfer(dst=#[0xf701426b], wad=524102.159) → (True)
[0.0.7.9.3.1.3] [call@0xf701426b] #[0xf701426b] .executeOperation(_arg0=('0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2',), _arg1=(524102159298234706604104,), _arg2=(0,), _arg3=#[0xf701426b], _arg4=) → ()
[0.0.7.9.3.1.3.1] [call@0xC02aaA39] #[WETH] .transfer(dst=#[Cream Finance Exploiter 3], wad=6000.0) → (True)
[0.0.7.9.3.1.3.2] [call@0xC02aaA39] #[WETH] .withdraw(wad=518102.159) → ()
[0.0.7.9.3.1.3.2.3] 518102.159 ETHER [call@0xf701426b] #[0xf701426b] .Transfer() → ()
[0.0.7.9.3.1.3.3] 518102.159 ETHER [call@0xD06527D5] #[Cream.Finance: crETH Token] .mint() → ()
[0.0.7.9.3.1.3.3.16] #[Cream.Finance: Comptroller] .mintAllowed(_arg0=#[Cream.Finance: crETH Token], _arg1=#[0xf701426b], _arg2

FIG. 4

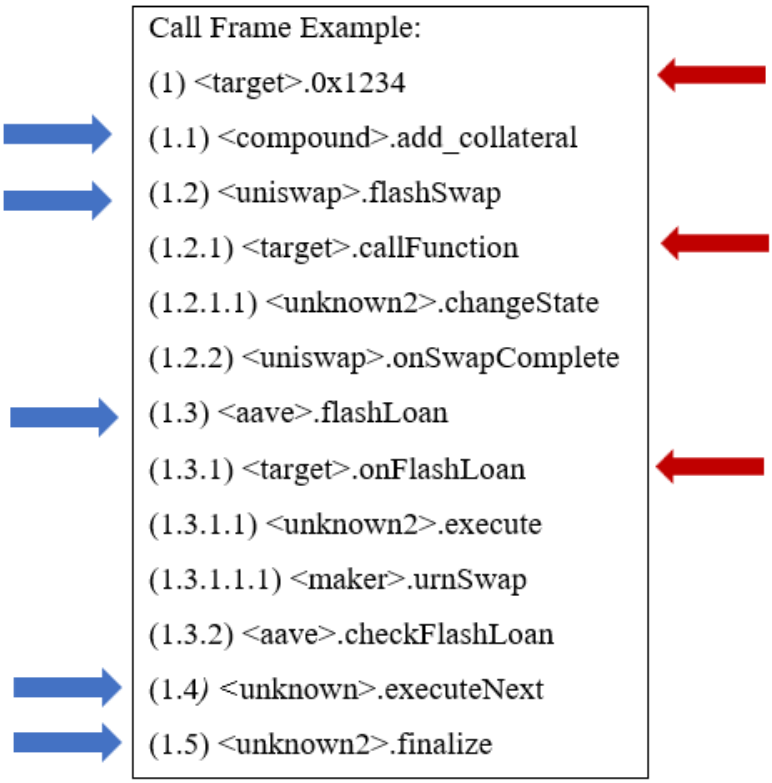

FIG. 5

Synthesis of <target> contract
(-) 0x1234
    (-) <compound>.add_collateral
    (-) <uniswap>.flashSwap
    (-) <aave>.flashLoan
    (-) <unknown>.executeNext
    (-) <unknown2>.finalize
(-) callFunction
    (-) <unknown2>.changeState
(-) onFlashLoan
    (-) <unknown2>.execute

FIG. 6

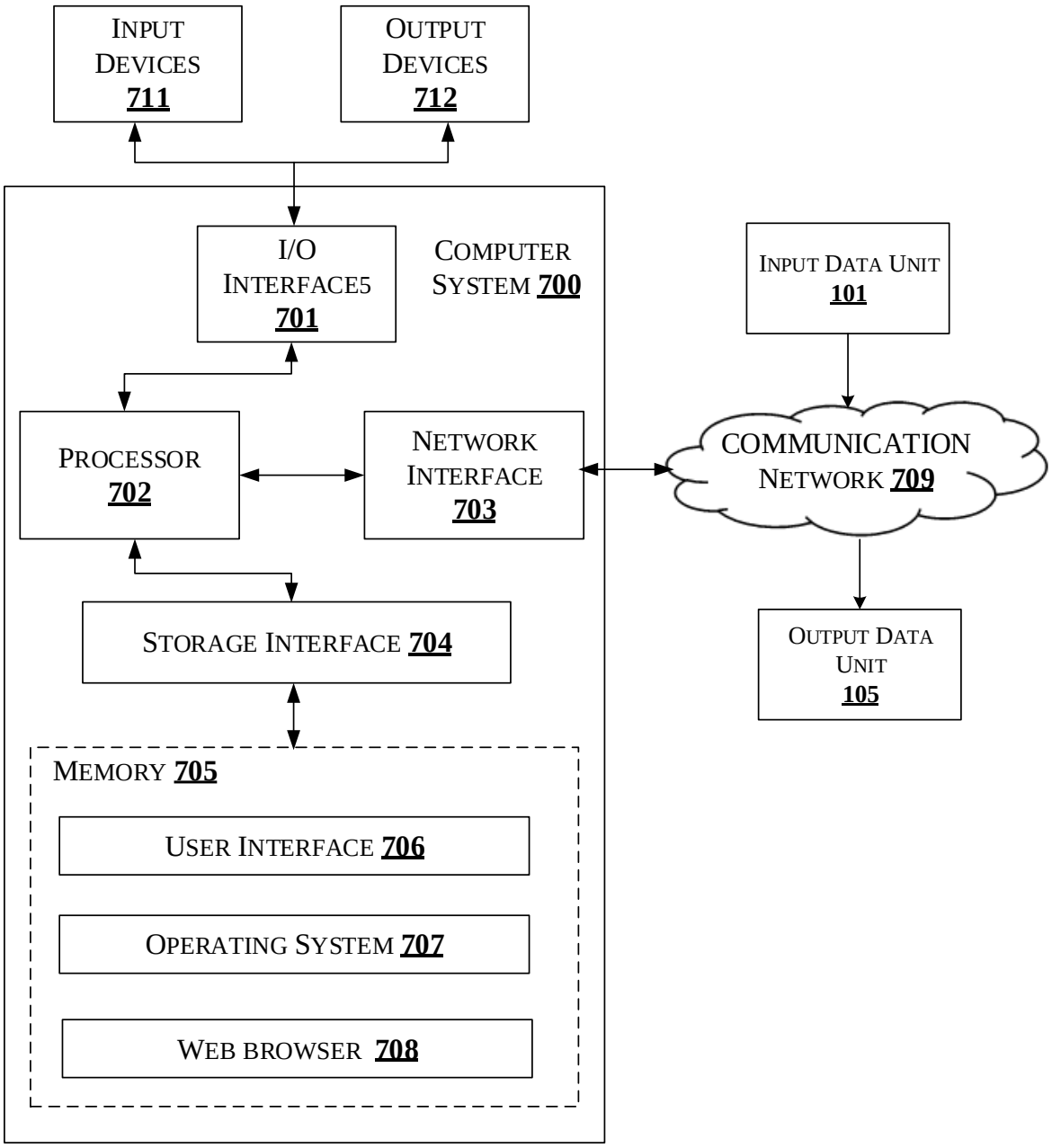
INPUT DEVICES 711
OUTPUT DEVICES 712
I/O INTERFACE5 701
COMPUTER SYSTEM 700
INPUT DATA UNIT 101
PROCESSOR 702
NETWORK INTERFACE 703
COMMUNICATION NETWORK 709
STORAGE INTERFACE 704
OUTPUT DATA UNIT 105
MEMORY 705
USER INTERFACE 706
OPERATING SYSTEM 707
WEB BROWSER 708

FIG. 7