CRTK

Release 0.0.23

Jun 09, 2020

Contents:

CHAPTER 1

Getting Start with CRTK

CRTK simplifies converting and parsing of Ethereum smart contracts, and provids many great features for reverse engineering researchers including:

- Load contract by type(runtime or creation)
- Get contract attributes such as bytecode, opcode, swarm source and constructor arguments
- Analysis for contract such as opcode occurrence and ERC standard checking

1.1 Quick start

Assuming you have Python already, install pysha3 for keccak:

```
$ pip install pysha3
```

Then install CRTK:

```
$ pip install crtk
```

Check installation in python interactive shell:

```
>>> import crtk
```

CHAPTER 2

CRTK.contract module

class CRTK.contract.**Contract**

Bases: object

get_address()

get_bytecode()

get_function_definition_list()

get_function_signature_list()

get_opcode()

get_opcode_occurrence(*collapse=0*)

is_ERC20()

is_ERC721()

is_ERC777()

is_real_contract()

CHAPTER 3

CRTK.creationcontract module

class CRTK.creationcontract.**CreationContract**(*bytecode*, *address=''*)

Bases: *CRTK.contract.Contract*

get_bzzr()

get_constructor_arguments()

get_deployment_bytecode()

get_deployment_opcode()

get_runtime_bytecode()

get_runtime_contract()

get_runtime_opcode()

is_runtime_contract()

CHAPTER 4

CRTK.runtimecontract module

class CRTK.runtimecontract.**RuntimeContract**(*bytecode*, *address=''*)

Bases: *CRTK.contract.Contract*

is_runtime_contract()

CHAPTER 5

CRTK.utilities module

CRTK.utilities.**check_ERC_standard**(*function_signatures*, *standard='ERC20'*)
Check if contract follows certain ERC standards.

input: list, string output: bool

standard - ERC20: Check if the contract follows ERC20. - ERC721: Check if the contract follows ERC721. - ERC777: Check if the contract follows ERC777.

CRTK.utilities.**clean_opcode**(*opcode_list*)
Clean opcode in a opcode list. Drop PUSH-like, DUP-like and SWAP-like opcodes, then replace all LOG-like opcodes with LOG.

input: list of strings output: list of strings

CRTK.utilities.**collapse_opcode**(*opcode_list*, *collapse*)
Collapse opcodes by certain level.

input: list of strings, int output: list of strings

collapse - 0: Count all opcodes including all the PUSH-like, DUP-like and SWAP-like ones. - 1: Collapse all PUSH-like opcodes to PUSH, DUP-like opcodes to DUP, SWAP-like opcodes to SWAP and LOG-like opcode to LOG. - 2: Drop PUSH-like, DUP-like and SWAP-like opcodes, then replace all LOG-like opcodes with LOG. - 3: Drop all PUSH-like, DUP-like, SWAP-like and LOG-like opcodes.

CRTK.utilities.**fix_hex_string**(*hex_string*)
Fix truncated hex strings. '4e' -> '0x0000004e'

input: string output: string

CRTK.utilities.**function_to_signature**(*function_name*)
Convert function definition to function signature.

input: string output: string

CRTK.utilities.**get_function_definitions_list**(*function_signatures*)
Get function definitions by function siguature-to-definition convertion.

input: list output:list

CRTK.utilities.**get_function_signatures_list**(*opcode_list*)
Get all the funtcion signatures within given opcode.

input: list of lists output: list

structure of an opcode: [address, bytecode, opcode, arguments if exist]

CRTK.utilities.**get_opcode_list**(*bytecode*)
Convert bytecode to opcode list.

input: string output: list of lists

structure of an opcode: [address, bytecode, opcode, arguments if exist]

CRTK.utilities.**opcode_occurrence**(*opcode_list*, *collapse=0*)
Count ccurences of each opcode by a certain opcode sequence.

input: list of strings, int output: dict (string -> int)

collapse - 0: Count all opcodes including all the PUSH-like, DUP-like and SWAP-like ones. - 1: Collapse all PUSH-like opcodes to PUSH, DUP-like opcodes to DUP, SWAP-like opcodes to SWAP and LOG-like opcode to LOG. - 2: Drop PUSH-like, DUP-like and SWAP-like opcodes, then replace all LOG-like opcodes with LOG. - 3: Drop all PUSH-like, DUP-like, SWAP-like and LOG-like opcodes.

CRTK.utilities.**signature_to_function**(*function_signature*)
Convert function signature to function definition by reverse query. See https://github.com/Yzstr/Function-Signatures.

input: string output: list

There might be several query results if hash collision encountered and also might be no result due to data vacancy.

CRTK.utilities.**split_bytecode**(*bytecode*)
Split contract creation code.

input: string output: tuple

structure of contract creation code +——————+ | Deployment Bytecode | +——————+ | Runtime Bytecode | +——————+ | BZZR: Swarm Source | +——————+ | Constructor Arguments | +——————+

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index