A Security Benchmark Suite Exploring the Existing Vulnerabilities of a Computer System

Version: 0.1.0

Jiameng Ying, Boya Li, Sihao Shen, and Wei Song*
Institute of Information Engineeringat at the Chinese Academy of Sciences
89 Minzhuang Road, Haidian, Beijing 100093, P. R. China
*songwei@iie.ac.cn

April 15, 2019

Contributors

Boya Li (11.2017 – 01.2019, IIE-CAS) Participated in the initial vulnerability analyses. Provided the initial test cases for the buffer over/underflow write tests. Provided the initial ideas for testing use-after-free vulnerabilities.

Sihao Shen (02.2019 – 09.2019, IIE-CAS) Added documentation of BOF and CFI. Added the initial support for the RISC-V RV64G ISA.

Wei Song (11.2017 – 09.2019, IIE-CAS) The organizer of the project.

Jiameng Yin (11.2017 – 01.2019, IIE-CAS) Participated in the initial vulnerability analyses. Provided the initial test cases for numerous CFI and CPI tests.

Yuhui Zhang (11.2017 – 04.2018, IIE-CAS) Participated in the initial vulnerability analyses.

Contents

1 Introduction **5**

2 Overview of the Security Benchmark Suite **7**

3 Description of Test Cases **9**

3.1 Buffer Overflow (BOF) 9

3.1.1 `overflow-write-index-data` 10

3.1.2 `overflow-write-index-heap` 11

3.1.3 `overflow-write-index-stack` 12

3.1.4 `overflow-write-ptr-data` 13

3.1.5 `overflow-write-ptr-heap` 14

3.1.6 `overflow-write-ptr-stack` 15

3.1.7 `underflow-write-index-data` 16

3.1.8 `underflow-write-index-heap` 17

3.1.9 `underflow-write-index-stack` 18

3.1.10 `underflow-write-ptr-data` 19

3.1.11 `underflow-write-ptr-heap` 20

3.1.12 `underflow-write-ptr-stack` 21

3.2 Control Flow Integrity (CFI) 22

3.2.1 `call-instruction-in-data` 23

3.2.2 `call-instruction-in-heap` 24

3.2.3 `call-instruction-in-rodata` 25

3.2.4 `call-instruction-in-stack` 26

3.2.5 `call-mid-func` 27

3.2.6 `call-mid-instruction` 28

3.2.7 `call-wrong-func-offset-vtable` 29

3.2.8 `call-wrong-func-poly-vtable` 30

3.2.9 `call-wrong-func-vtable` 31

3.2.10 `call-wrong-func-within-static-analysis` . . . 32

3.2.11 `call-wrong-func` 33

3.2.12 `call-wrong-num-arg-func` 34

3.2.13 `call-wrong-num-arg-vtable-heap` 35

3.2.14 `call-wrong-num-arg-vtable` 36

3.2.15 `call-wrong-num-func-vtable-heap` 37

3.2.16 `call-wrong-num-func-vtable` 38

3.2.17 `call-wrong-type-arg-dp2fp-func-data` 39

3.2.18 `call-wrong-type-arg-dp2fp-func-heap` 40

3.2.19 `call-wrong-type-arg-dp2fp-func-rodata` 41

3.2.20 call-wrong-type-arg-dp2fp-func-stack 42
3.2.21 call-wrong-type-arg-fp2dp-func 43
3.2.22 call-wrong-type-arg-int2double-func 44
3.2.23 call-wrong-type-arg-vtable-heap 45
3.2.24 call-wrong-type-arg-vtable 46
3.2.25 call-wrong-type-arg-func 47
3.2.26 call-wrong-vtable-heap 48
3.2.27 jump-instruction-in-bss 49
3.2.28 jump-instruction-in-heap 50
3.2.29 jump-instruction-in-rodata 51
3.2.30 jump-instruction-in-stack 52
3.2.31 jump-mid-func 53
3.2.32 jump-mid-instruction 54
3.2.33 return-to-func 55
3.2.34 return-to-instruction-in-data 56
3.2.35 return-to-instruction-in-heap 57
3.2.36 return-to-instruction-in-rodata 58
3.2.37 return-to-instruction-in-stack 59
3.2.38 return-to-libc 60
3.2.39 return-to-mid-instruction 61
3.2.40 return-to-non-call-site 62
3.2.41 return-to-wrong-call-site 63
3.2.42 return-without-call 64

4 Remaining Issues **65**

Chapter 1

Introduction

Chapter 2

Overview of the Security Benchmark Suite

Chapter 3

Description of Test Cases

3.1 Buffer Overflow (BOF)

- Overflow

3.1.1 `overflow-write-index-data`
3.1.2 `overflow-write-index-heap`
3.1.3 `overflow-write-index-stack`
3.1.4 `overflow-write-ptr-data`
3.1.5 `overflow-write-ptr-heap`
3.1.6 `overflow-write-ptr-stack`

- Underflow

3.1.7 `underflow-write-index-data`
3.1.8 `underflow-write-index-heap`
3.1.9 `underflow-write-index-stack`
3.1.10 `underflow-write-ptr-data`
3.1.11 `underflow-write-ptr-heap`
3.1.12 `underflow-write-ptr-stack`

3.1.1 `overflow-write-index-data`

Description

Overflow by illegally using the buffer **index**. The index exceeds the expected buffer length, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **data** section.

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.2 `overflow-write-index-heap`

Description

Overflow by illegally using the buffer **index**. The index exceeds the expected buffer length, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **heap** (dynamically allocated data).

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.3 `overflow-write-index-stack`

Description

Overflow by illegally using the buffer **index**. The index exceeds the expected buffer length, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **stack**.

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.4 `overflow-write-ptr-data`

Description

Overflow by illegally using a buffer **pointer**. The pointer is modified to pointing a location outside the buffer, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **data** section.

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.5 `overflow-write-ptr-heap`

Description

Overflow by illegally using a buffer **pointer**. The pointer is modified to pointing a location outside the buffer, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **heap** (dynamically allocated data).

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.6 `overflow-write-ptr-stack`

Description

Overflow by illegally using a buffer **pointer**. The pointer is modified to pointing a location outside the buffer, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **stack**.

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.7 `underflow-write-index-data`

Description

Underflow by illegally using the buffer **index**. The index exceeds the expected buffer length, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **data** section.

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.8 `underflow-write-index-heap`

Description

Underflow by illegally using the buffer **index**. The index exceeds the expected buffer length, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **heap** (dynamically allocated data).

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.9 `underflow-write-index-stack`

Description

Underflow by illegally using the buffer **index**. The index exceeds the expected buffer length, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **stack**.

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.10 `underflow-write-ptr-data`

Description

Underflow by illegally using a buffer **pointer**. The pointer is modified to pointing a location outside the buffer, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **data** section.

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.11 `underflow-write-ptr-heap`

Description

Underflow by illegally using a buffer **pointer**. The pointer is modified to pointing a location outside the buffer, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **heap** (dynamically allocated data).

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.1.12 `underflow-write-ptr-stack`

Description

Underflow by illegally using a buffer **pointer**. The pointer is modified to pointing a location outside the buffer, leading to ilegal write accesses to the data outside the buffer. The overwrite occurs in the **stack**.

Vulnerability

Generic data overwrite and loss.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.2 Control Flow Integrity (CFI)

- Forward-edge CFI
  - Call

3.2.1 `call-instruction-in-data`
3.2.2 `call-instruction-in-heap`
3.2.3 `call-instruction-in-rodata`
3.2.4 `call-instruction-in-stack`
3.2.5 `call-mid-func`
3.2.6 `call-mid-instruction`
3.2.7 `call-wrong-func-offset-vtable`
3.2.8 `call-wrong-func-poly-vtable`
3.2.9 `call-wrong-func-vtable`
3.2.10 `call-wrong-func-within-static-analysis`
3.2.11 `call-wrong-func`
3.2.12 `call-wrong-num-arg-func`
3.2.13 `call-wrong-num-arg-vtable-heap`
3.2.14 `call-wrong-num-arg-vtable`
3.2.15 `call-wrong-num-func-vtable-heap`
3.2.16 `call-wrong-num-func-vtable`
3.2.17 `call-wrong-type-arg-dp2fp-func-data`
3.2.18 `call-call-wrong-type-arg-dp2fp-func-heap`
3.2.19 `call-wrong-type-arg-dp2fp-func-rodata`
3.2.20 `call-wrong-type-arg-dp2fp-func-stack`
3.2.21 `call-wrong-type-arg-fp2dp-func`
3.2.22 `call-wrong-type-arg-int2double-func`
3.2.23 `call-wrong-type-arg-func-vtable-heap`
3.2.24 `call-wrong-type-arg-vtable`
3.2.25 `call-wrong-type-arg-func`
3.2.26 `call-wrong-vtable-heap`

  - Jump

3.2.27 `jump-instruction-in-bss`
3.2.28 `jump-instruction-in-heap`
3.2.29 `jump-instruction-in-rodata`
3.2.30 `jump-instruction-in-stack`
3.2.31 `jump-mid-func`
3.2.32 `jump-mid-instruction`

- Backward-edge CFI
  - Return

3.2.33 `return-to-func`
3.2.34 `return-to-instruction-in-data`
3.2.35 `return-to-instruction-in-heap`
3.2.36 `return-to-instruction-in-rodata`
3.2.37 `return-to-instruction-in-stack`
3.2.38 `return-to-libc`
3.2.39 `return-to-mid-instruction`
3.2.40 `return-to-non-call-site`
3.2.41 `return-to-wrong-call-site`
3.2.42 `return-to-without-call`

3.2.1 `call-instruction-in-data`

Description

Illegally call an instruction constructed in **data**.

Vulnerability

Its executable on writable area (data)

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.2 `call-instruction-in-heap`

Description

Illegally call an instruction constructed in **heap**.

Vulnerability

Its executable on writable area (**heap**)

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.3 `call-instruction-in-rodata`

Description

Illegally call an instruction constructed in **rodata**.

Vulnerability

Its executable on writable area (**rodata**)

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.4 `call-instruction-in-stack`

Description

Illegally call an instruction constructed in **stack**.

Vulnerability

Its executable on writable area (**stack**)

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.5 `call-mid-func`

Description

Call a fake function entry point at the **middle** of a **function**.

Vulnerability

Illegal function entry point.

Remarks

This is a common case for calling (forward-edge) a gadget in a ROP attack.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.2.6 `call-mid-instruction`

Description

Call a fake instruction located at the **middle** of an other legal **instruction**.

Vulnerability

Illegal function entry point; illegal instruction formation.

Test result

return	*description*
0	might be **vulnerable**
other	might be safe

Known issues

Currently it is difficult to differentiate the case of a success call to the middle of an instruction and the case of fixed return value through compiler optimization.

3.2.7 `call-wrong-func-offset-vtable`

Description

Modify the **virtual table** pointer by making it pointing to its own but with an **offset**. **Call** a **function** with the **wrong** virtual table.

Vulnerability

Modifying virtual table pointer; modifying virtual table pointer by adding an offset.

Test result

return	*description*
0	**vulnerable**
1	might be **blocked by compiler optimization**
2–4	**partially vulnerable**
other	might be safe

Known issues

None.

3.2.8 `call-wrong-func-poly-vtable`

Description

Illegally call the wrong virtual function by modifying the poly Vtable pointer.

Vulnerability

Modify the Vtable pointer and call the wrong virtual function.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.2.9 `call-wrong-func-vtable`

Description

Illegally call the wrong virtual function by modifying the Vtable pointer.

Vulnerability

Modify the pointer and call the wrong function.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.10 `call-wrong-func-within-static-analysis`

Description

Illegally call the wrong function.

Vulnerability

Modify the pointer and call the wrong function.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.2.11 `call-wrong-func`

Description

Illegally call the wrong function.

Vulnerability

Illegally modify the pointer and call the wrong function.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.12 `call-wrong-num-arg-func`

Description

Illegally call a function with mismatched number of arguements.

Vulnerability

Break the function calling convention.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.2.13 `call-wrong-num-arg-vtable-heap`

Description

Replace the Vtable pointer with a fake Vtable constructed in heap and illegally call a virtual function with mismatched number of arguements.

Vulnerability

Break the function calling convention, the data integrity of the Vtable pointer and the Vtable itself.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: Currently only works with object allocated on heap.

3.2.14 `call-wrong-num-arg-vtable`

Description

Illegally call a virtual function with mismatched number of arguements by modifying the VTable pointer.

Vulnerability

Break the function calling convention and the data integrity of the Vtable pointer.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: Currently only works with object allocated on heap.

3.2.15 `call-wrong-num-func-vtable-heap`

Description

Replace the Vtable pointer with a fake Vtable constructed in heap with different number of virtual functions and illegally call a fake virtual function.

Vulnerability

Break the data integrity of the Vtable pointer and the Vtable itself.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: Currently only works with object allocated on heap.

3.2.16 `call-wrong-num-func-vtable`

Description

Illegally call a fake virtual function with the VTable being replaced with another one of different number of virtual functions.

Vulnerability

Break the data integrity of the Vtable pointer.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: Currently only works with object allocated on heap.

3.2.17 `call-wrong-type-arg-dp2fp-func-data`

Description

Illegally call a function with wrong types of arguments constructed in **data**.

Vulnerability

Break the function calling convention and non-execution on writable area (**data**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.18 `call-wrong-type-arg-dp2fp-func-heap`

Description

Illegally call a function with wrong types of arguments constructed in **heap**.

Vulnerability

Break the function calling convention and non-execution on writable area (**heap**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.19 `call-wrong-type-arg-dp2fp-func-rodata`

Description

Illegally call a function with wrong types of arguments constructed in **rodata**.

Vulnerability

Break the function calling convention and its excutable on writable area (**rodata**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.20 `call-wrong-type-arg-dp2fp-func-stack`

Description

Illegally call a function with wrong types of arguments constructed in **stack**.

Vulnerability

Break the function calling convention and non-execution on writable area (**stack**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.21 `call-wrong-type-arg-fp2dp-func`

Description

Illegally call a function with wrong types of argument (dp) expected to be fp.

Vulnerability

Break the function calling convention.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.22 `call-wrong-type-arg-int2double-func`

Description

Illegally call a function with wrong types of argument (int) expected to be double.

Vulnerability

Break the function calling convention.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.23 `call-wrong-type-arg-vtable-heap`

Description

Replace the Vtable pointer with a fake Vtable constructed in heap and illegally call a virtual function with wrong types of arguements.

Vulnerability

Break the function calling convention, the data integrity of the Vtable pointer and the Vtable itself.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: Currently only works with object allocated on heap.

3.2.24 `call-wrong-type-arg-vtable`

Description

Illegally call a function with wrong types of arguements by modifying the VTable pointer.

Vulnerability

Break the function calling convention and the data integrity of the Vtable pointer.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: Currently only works with object allocated on heap.

3.2.25 `call-wrong-type-arg-func`

Description

Illegally call a function with wrong types of arguements.

Vulnerability

Break the function calling convention.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.2.26 `call-wrong-vtable-heap`

Description

Replace the Vtable pointer with a fake Vtable constructed in heap.

Vulnerability

Break the data integrity of the Vtable pointer and the Vtable itself.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: Currently only works with object allocated on heap.

3.2.27 `jump-instruction-in-bss`

Description

Illegally jump from a function to an instruction constructed in **bss**.

Vulnerability

Break the execution compartment complied by most C/C++ programs and non-execution on writable area (**bss**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.28 `jump-instruction-in-heap`

Description

Illegally jump from a function to an instruction constructed in **heap**.

Vulnerability

Break the execution compartment complied by most C/C++ programs and non-execution on writable area (**heap**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.29 `jump-instruction-in-rodata`

Description

Illegally jump from a function to an instruction constructed in **rodata**.

Vulnerability

Break the execution compartment complied by most C/C++ programs and its excutable on writable area (**rodata**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.30 `jump-instruction-in-stack`

Description

Illegally jump from a function to an instruction constructed in **stack**.

Vulnerability

Break the execution compartment complied by most C/C++ programs and non-execution on writable area (**stack**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.31 `jump-mid-func`

Description

Jump to the **middle** of another **function**.

Vulnerability

Illegal jump target; breaking the function execution context.

Remarks

This is a common case for jumping to (forward-edge) a gadget in a JOP attack.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

None.

3.2.32 `jump-mid-instruction`

Description

Jump to a fake instruction located at the **middle** of an other legal **instruction**.

Vulnerability

Illegal instruction formation.

Test result

return	*description*
0	**vulnerable**
3	**partially vulnerable**
other	might be safe

Known issues

None.

3.2.33 `return-to-func`

Description

Illegally modify the return address stored on the stack and directly return to another function.

Vulnerability

Break the backward CFI and the integrity of the return address.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.34 `return-to-instruction-in-data`

Description

Illegally modify the return address stored on the stack and then return to an instruction constructed in **data**.

Vulnerability

Break the backward CFI and the integrity of the return address and non-execution on writable area (**data**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.35 `return-to-instruction-in-heap`

Description

Illegally modify the return address stored on the stack and then return to an instruction constructed in **heap**.

Vulnerability

Break the backward CFI and the integrity of the return address and non-execution on writable area (**heap**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.36 `return-to-instruction-in-rodata`

Description

Illegally modify the return address stored on the stack and then return to an instruction constructed in **rodata**.

Vulnerability

Break the backward CFI and the integrity of the return address and its executable on writable area (**rodata**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.37 `return-to-instruction-in-stack`

Description

Illegally modify the return address stored on the stack and then return to an instruction constructed in **stack**.

Vulnerability

Break the backward CFI and the integrity of the return address and non-execution on writable area (**stack**).

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.38 `return-to-libc`

Description

Illegally modify the return address stored on the stack and then return to a libc.

Vulnerability

Break the backward CFI and the integrity of the return address.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.39 `return-to-mid-instruction`

Description

Illegally modify the return address stored on the stack and then return to the middle of an instruction.

Vulnerability

Break the backward CFI and the integrity of the return address.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.40 `return-to-non-call-site`

Description

Illegally modify the return address stored on the stack and then return to an none call-site position.

Vulnerability

Break the backward CFI and the integrity of the return address.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: The `rbp` register might be (with `-g`) or not be (with `-O2`) pushed to the stack. The return address is modified by embedded assembly using `rsp` as the base register. See `STACK_STRUCT` in the make file.

3.2.41 `return-to-wrong-call-site`

Description

Illegally modify the return address stored on the stack and then return to a wrong call-site position.

Vulnerability

Break the backward CFI and the integrity of the return address.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

3.2.42 `return-without-call`

Description

Illegally add a fake function call onto the stack and return to it..

Vulnerability

Break the backward CFI and the integrity of the return address.

Test result

return	*description*
0	**vulnerable**
other	might be safe

Known issues

x86_64: The `rbp` register might be (with `-g`) or not be (with `-O2`) pushed to the stack. Currently the test works only with `-O2`.

Chapter 4

Remaining Issues

- `call-wrong-num-arg-func` 3.2.12: test for arguements passed on stack.
- `call-wrong-type-arg-func` 3.2.25: more importantly, test (data/code) pointer to integer.
- `call-wrong-num-arg-vtable` 3.2.14: known issues.
- `call-wrong-num-func-vtable` 3.2.16: known issues.
- `return-without-call` 3.2.42: known issues.
- call a unvisible function (call a local function from outside).
- differentiate between global data, heap and stack.