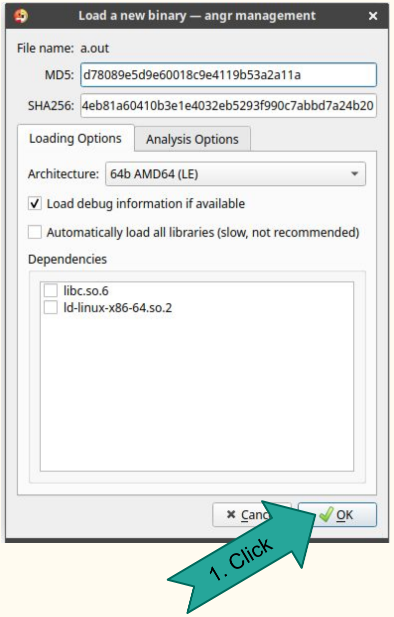
Symbolic Execution Debugger Tutorial

Common Strategy to Mitigate Path Explosion

Load a new binary and Start a Debugger

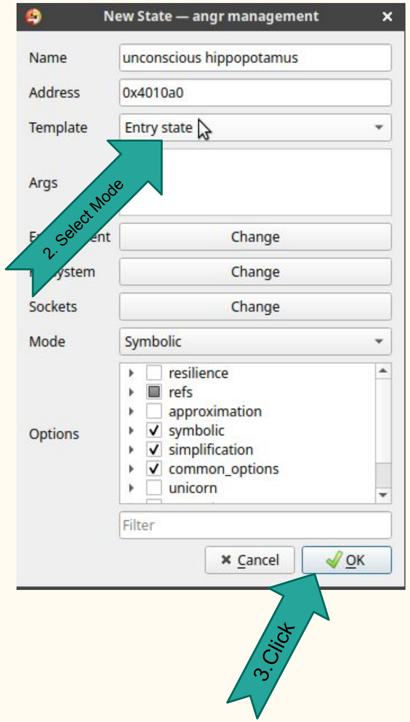
1. Compile the source code.
   - `gcc main.c -g -gdwarf-4 -o main`
   - Don't forget the `**-g gdwarf-4**` flag to add debug info.
2. Load it into SeDBG
   - Menu -> File -> Load a new binary
3. Start a Debugger:
   - Click 'New Debugger' in toolbar and 'New simulation'
   - Change the Template to ` Entry State` and click Ok


Load a new binary — angr management

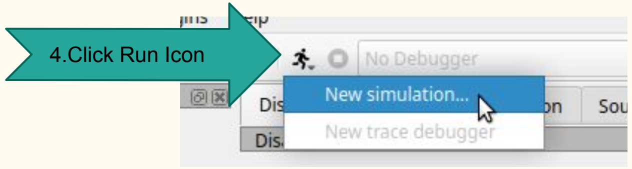
File name: a.out
MD5: d78089e5d9e60018c9e4119b53a2a11a
SHA256: 4eb81a60410b3e1e4032eb5293f990c7abbd7a24b20

Loading Options | Analysis Options

Architecture: 64b AMD64 (LE)
[✓] Load debug information if available
[ ] Automatically load all libraries (slow, not recommended)
Dependencies
[ ] libc.so.6
[ ] ld-linux-x86-64.so.2

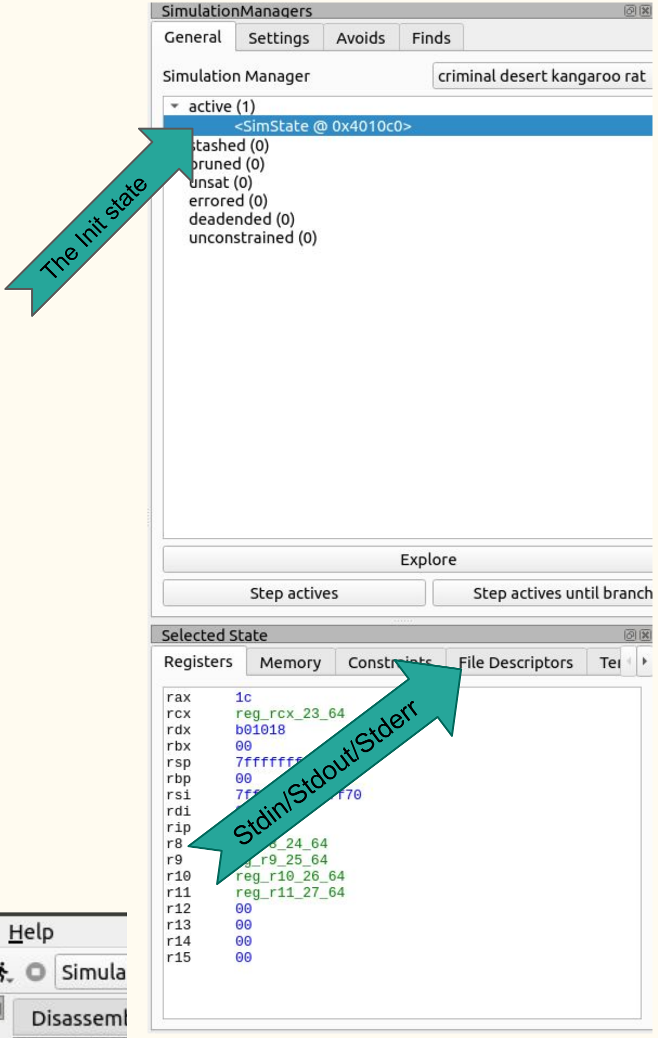

New State — angr management

Name: unconscious hippopotamus
Address: 0x4010a0
Template: Entry state
Args:
Environment: Change
Filesystem: Change
Sockets: Change
Mode: Symbolic
Options:
[ ] resilience
[▨] refs
[ ] approximation
[✓] symbolic
[✓] simplification
[✓] common_options
[ ] unicorn



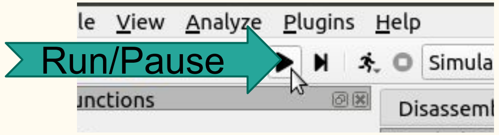
Debugger Process

- States are organized into “stashes”. Each time a step is taken forward, the execute engine retrieves one state and executes the subsequent instrument of this state.
- If it encounters a branch, it will produce one or more success states.
- The engine will then determine whether or not the new state reaches the end of the program or the AVOID line.
- If not, return the new state to the active stash state for further processing.
- When you click Continue, this procedure will be repeated until there are no more states in the active stash.



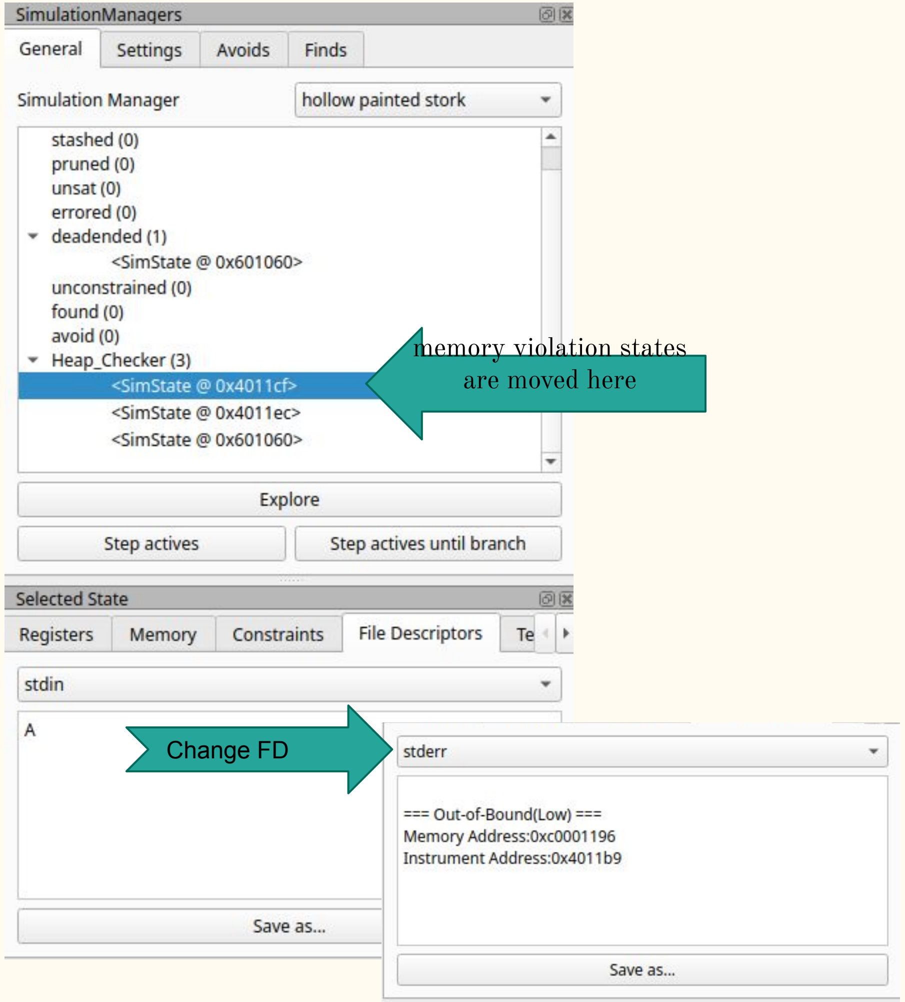
```
active (1)
    <SimState @ 0x4010c0>
stashed (0)
pruned (0)
unsat (0)
errored (0)
deadended (0)
unconstrained (0)
```

```
rax      1c
rcx      reg_rcx_23_64
rdx      b01018
rbx      00
rsp      7ffffff
rbp      00
rsi      7f       f70
rdi
rip
r8         8_24_64
r9        g_r9_25_64
r10      reg_r10_26_64
r11      reg_r11_27_64
r12      00
r13      00
r14      00
r15      00
```



Debugger Process

- Within SeDBG, the Heap _Checker plugin stands as a valuable aid, facilitating the identification of specific memory errors during symbolic execution.
- The tagged states will sent to Heap_Checker stash. select a state to perform a detailed examination
- Select the File Description tab to view the input that will cause heap out-of-bound and the memory access violation details in stderr.
- Click 'Save as...' to save input.txt as stdin.
- Not all memory errors lead to immediate crashes by default



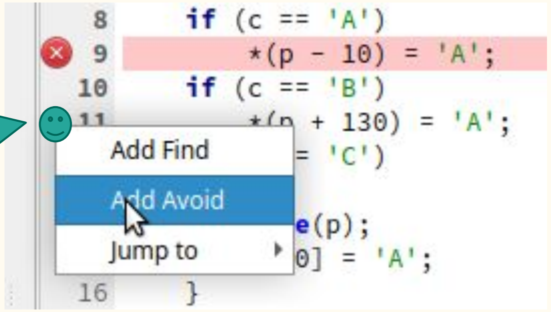
```
=== Out-of-Bound(Low) ===
Memory Address:0xc0001196
Instrument Address:0x4011b9
```

Tune the Path Exploration: Avoid Path

To avoid this less-informative path, we may set a line of source code to AVOID. When a state reaches an AVOID address, it is transferred from the "active" stash to the "avoid" stash.

- Examine the code under the Source Viewer Tab to identify potential unsuccessful paths.
- Left-click the line number's left blank and select "Add Avoid" to skip this line. A red icon will display at the location.
- Repeatedly clicking the icon will erase the AVOID point.
- Left click the avoid icon to delete this avoid address

Left click at the left of line number



```
    if (c == 'A')
        *(p - 10) = 'A';
    if (c == 'B')
        *(p + 130) = 'A';
            = 'C')

            e(p);
            0] = 'A';
    }
```

Type of Avoid Path

- Remove the failed operation path.
  - Print Invalid argument / command and Try again.
  - Return Error in a function
- Remove unnecessary command branch
  - Help command: (Just print some information and return to event dispatch again)
  - The command not used in Task
- Remove the duplicate switch-case branch
- Shrink the size of Array before compile

```
1.   link_node* find_user_node (const char *name)
2.   {
3.     for (link_node* node = username_list_head;
     node; node = node->next)
4.         if (strcmp (name, node->name) == 0)
5.           return node;
6.     return NULL;
7.   }
```

Return NULL in line 7 seems like a failed path and should be avoid, but sometime the program needs to ensure the same name node do not exist before adding.

Practice Task 1

- It’s a subset of the FTP protocol that only permits the following login methods:
  - >USER admin
  - 331 User admin OK.
  - >PASS ftp
  - 230 OK. Current user is admin.
  - You successfully logged.
- It uses a structure `session` to save the state of login process.
  - Session.state = 0 init state
  - Session.state = 1 username recorded.
  - Session.state >=2 logged in
- However, this program has a problem that allows users to log in without a password or even the PASS command.

Tip:

When strcpy is performed inside a structure, a buffer overflow occurs.

Compile the main.c by:

1. gcc main.c -g -gdwarf-4 -o main
2. Don’t forget the `-g gdwarf-4` flag to add debug info.

Practice Task 1 Solve Strategy

How to set avoid to restrict search space in order to locate the state that triggered the bug.

- Since we don’t use PASS command, set avoids at any command we don’t need.
  - command_PASS function
  - command_QUIT function
- We want to find the shortest path to the target state. Therefore, we don’t want to waste time on any failed attempts. Avoid failed attempts and ensure that every step in the path is valuable.
  - Line 81 printf("530 You aren't logged in.\n");
  - Line 66 printf("550 Invalid argument (no newline)\n");
  - Line 28 printf("530 Please tell me who you are\n");
  - Line 37 printf("530 Login authentication failed as User %s.\n", session->user);
- Set Find at
  - Line 85 printf("You successfully logged in.\n");

Practice Task 2

- This program reads a string from standard input and outputs the escaped version on standard output.
- However, input may cause the software to crash.

Tip: when a character is pushed into a buffer on the stack, a buffer overflow occurs.

Compile the main.c by:

1. gcc main.c -g -gdwarf-4 -o main
2. Don’t forget the `-g gdwarf-4` flag to add debug info.

Practice Task 2 Solve Strategy

The number of states in the escape function loop grows rapidly.

- 5 ways loop indicates 5 times the number of states following one loop.
- We need 64 loop to complete the string.
- 5^64 is an unsolvable large number.

We could reduce the amount of the buffer to accelerate program termination.

Four of the five branches transform one illegal character to two escaped characters. We need only one of them.

- Line 5 #define BUFF_LEN 64
  - Shrink the size to 8
  - Don’t forget to recompile the source code.
- Avoid Line 11 ~ 13:
  - case '\n': *dest++ = '\\'; *dest = 'n'; break;
  - Only one branch remained to keep program’s functional.
- The total number of state will be
  - 2^8 much better!

Practice Task 3

- Similar to the Practice Task 1 but
  - You are logged in at the first time
  - We need the QUIT command now.
  - Struct session are moved to heap area so we need free it after
- Do we manage the memory currently?

Tip: use-after-free

Compile the main.c by:

1. gcc main.c -g -gdwarf-4 -o main
2. Don’t forget the `-g gdwarf-4` flag to add debug info.

Practice Task 3 Solve Strategy

- Avoid failed attempts and ensure that each step along the path is significant.
    - Line 53 printf("550 Invalid argument (no newline)\n");
    - Line 64 printf("530 ???.\n");
    - Line 15 HELP command function