АКОС

Семинар 1
Инструменты разработки

Этапы компиляции

1. Препроцессинг: **-E**
2. Компиляция: **-S**
3. Ассемблирование: **-c**
4. Линковка

N.B. Флаги сообщают компилятору на какой стадии остановиться

--save-temps - сохранить все временные файлы при компиляции

z.B. **$ g++ -S main.cpp**

GDB

- **g++ -g main.cpp -> gdb a.out / gdb --args a.out arg1 arg2**
- База:
  - **break(b)** - breakpoint
  - **step(s) / next(n)** - следующая строка с заходом / без захода в функцию
  - **stepi(si) / nexti(ni)** - следующая ассемблерная инструкция с заходом / без захода в функцию
  - **run(r)** - запустить исполнение программы с начала
  - **continue(c)** - продолжить исполнение (до конца или следующего breakpoint)
  - **quit** - выйти
- Layout:
  - **layout next(n)/prev(p)/src/asm/split/regs** - изменить layout
  - **focus next(n)/prev(p)/src/asm/split/regs** - изменить фокус

GDB

- **gdb -p** - подключиться уже запущенному процессу
- **detach** - отсоединиться от процесса
- **backtrace(bt)** - стек вызовов
- **print(p) var** - распечатать значение переменной var
- **finish** - дойти до конца исполнения функции
- **return** – выйти из функции
- **set disassembly-flavor intel (att)** (>> ~/.gdbinit)

GDB (Breakpoints)

- **break** (**b**) **func_name/line/...**– set a breakpoint
- **info break** – get the list of breakpoints
- **delete *break_num*** – delete a breakpoint
- **clear** – delete all breakpoints
- **enable** / **disable** *break_num* – obvious

GDB (Stack)

- **backtrace** (**bt**) – show backtrace
- **backtrace full** – show backtrace with local variables
- **list *func_name*/*line range*/...** – displays some code

GDB

- **set var *var_name*=*value*** – set a value to a variable
- **x/nfu addr** - examine memory
- **p /format *var_name*** – display a variable
- Format:
  - a – pointer
  - d – decimal
  - x – hexadecimal
  - f – floating point value
  - ...

Sources

GDB cheat sheet
One more

Sanitizers

- -fsanitize=…
  - address
  - undefined
  - leak
  - thread

How does it work?

- malloc/free wrappers
- memory access wrappers
- stack
- For more information, read GitHub

Before:

```
*address = ...;  // or: ... = *address;
```

After:

```
if (IsPoisoned(address)) {
  ReportError(address, kAccessSize, kIsWrite);
}
*address = ...;  // or: ... = *address;
```

Original code:

```
void foo() {
  char a[8];
  ...
  return;
}
```

Instrumented code:

```
void foo() {
  char redzone1[32];  // 32-byte aligned
  char a[8];          // 32-byte aligned
  char redzone2[24];
  char redzone3[32];  // 32-byte aligned
  int  *shadow_base = MemToShadow(redzone1);
  shadow_base[0] = 0xffffffff;  // poison redzone1
  shadow_base[1] = 0xffffff00;  // poison redzone2, unpoison 'a'
  shadow_base[2] = 0xffffffff;  // poison redzone3
  ...
  shadow_base[0] = shadow_base[1] = shadow_base[2] = 0; // unpoison all
  return;
}
```

strace

- ltrace - лопирует вызов библиотечных функций
- strace - лопирует вызов syscalls

z.B. strace ./a.out

```
write(1  "Hello, world\n"  13)                = 13
```

Флаги strace:
- -v (verbose) - подробный вывод
- -f - следить вывод в том числе от дочерних процессов
- -c - статистика по системным вызовам
- -e *syscall name* - отследить конкретный системный вызов
- -p - подключиться к процессу
- -t добавить отметки времени

Разделы man

1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within program libraries)
4 Special files (usually found in /dev)
5 File formats and conventions eg /etc/passwd
6 Games
7 Miscellaneous (including macro packages and conventions), e.g. **man**(7), **groff**(7)
8 System administration commands (usually only for root)
9 Kernel routines [Non standard]

(From $ man man)

Конец