CPSC 457 - Tutorial 21
SystemTap 101

Daniel de Castro

Department of Computer Science
University of Calgary
Alberta, Canada

April 5, 2012
Introduction

- **SystemTap**: Scripts for analysing a “live” system without modifying the kernel.

  **Tracing** Provides information about execution while running
  Think *strace*

  **Debugging** Set breakpoints, look variables, memory, registers, stack trace, etc.
  Think *GDB*

  **Profiling** Generate data during execution for further analysis
  Think *Oprofile*
How can SystemTap access “privileged” information?

- Use of Kprobes, an interface for kernel modules to implant probes and register corresponding probe handlers.
- Kernel debug information provides addresses for functions and variables referenced in the script.
- `stap` parses the script and translate into C code, which is compiled as a module and loaded.
Installation

- Note: Please, do not use your modified kernel!
- Installation on Fedora 10
  
  ```
  sudo yum install systemtap systemtap-runtime
  sudo yum install kernel-devel-`uname -r`
  sudo debuginfo-install kernel-`uname -r`
  ```
- System tap modules
- Kernel development tools (needed to create LKM)
- Kernel debug information (functions, variables, line numbers)
Probes

General syntax of a script

```plaintext
function FUNC { ... }
probe PROBE { HANDLER }
```

- Probe: “Event listener”

Some examples of probes:

- `begin` The startup of the systemtap session.
- `end` The end of the systemtap session.
- `kernel.function("sys.open")` The entry to the function named `sys.open` in the kernel.
- `syscall.close.return` The return from the close system call.
- `module("ext3").statement(0xdeadbeef)` The addressed instruction in the ext3 filesystem driver.
- `timer.ms(200)` A timer that fires every 200 milliseconds.
- `perf.hw.cache_misses` A particular number of CPU cache misses have occurred.
- `process("a.out").statement("*@main.c:200")` Line 200 of the a.out program.

$ man stapprobes
Handles and Functions

General syntax of a script

```c
function FUNC { ... }
probe PROBE { HANDLER }
```

- Handlers are sequences of commands executed when a probe is “triggered”.
- Functions are also sequence of commands, but called by handlers.
- C-Like Syntax (AWK based)
  - Provide statements for loops (`for`, `while`, `foreach`), conditional branches (`if`), and much more.
- Functions can be implemented in C (not covered here)

Some examples of useful functions get:

- `tid()` The id of the current thread.
- `pid()` The process (task group) id of the current thread.
- `uid()` The id of the current user.
- `execname()` The name of the current process.
- `gettimeofday_s()` Number of seconds since epoch.
- `probefunc()` If known, the name of the function in which this probe was placed.
- `$$vars` If available, a pretty-printed listing of all local variables in scope.
- `print_backtrace()` If possible, print a kernel backtrace.

```bash
$ man stapfuncs
```
Probing a kernel function

fork.stp

```c
global proc_counter

probe begin {
    print ("Starting monitoring...");
    print ("Press ^C to terminate\n");
    printf ("%-25s %-10s %-s", "Process Name", "PID", "Flags")
}

probe kernel.function("do_fork") {
    proc_counter++
    printf("%-25s %-10d 0x%-x", execname(), pid(), $clone_flags)
}

probe end {
    printf ("\n%d processes forked", proc_counter)
}
```

- Global variable
- kernel.function

Executing

```bash
$ sudo stap fork.stp
```

You need root privileges to run. Why?
Probing system calls

syscount.stp

```plaintext
global syscall
probe syscall.* { syscall[name] += 1 }
probe timer.s(10) {
    foreach(n in syscall limit 5)
    printf("%s : %d\n", n, syscall[n])
    delete syscall
}
```

- Syscall probe
- Associative Arrays
- Foreach loop
Targeted monitoring (Reusable scripts)

- Assume you want to monitor a specific process or program.

```bash
# targeted.stp

probe syscall.* { 
    if ( pid() == target() )
        printf("Syscall: %s",name)
}
```

- `sudo stap targeted.stp -x 1234`
  - Traces only the process which PID is 1234

- `sudo stap targeted.stp -c ls`
  - Traces only the process `ls`

**Note:** The script ends when the process is terminated.
Practice

Let us practice a bit!

Thank you!

May the Force be with all of you!

Whatever “Force” means for each one of you!