Subject: Re: Container Test Campaign
Posted by Sam Vilain on Thu, 22 Jun 2006 23:39:24 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Marc E. Fiuczynski (mef@CS.Princeton.EDU):
>> Hi Clement,
>>
>> You mention that testing isolation properties is more of an extra than an
>> immediate criteria. Based on our experience, this actually is a fairly
>> important criteria. Without decent isolation (both from a namespace and
>
> As we develop our own patches for upstream inclusion, we will also be
> writing testcases to verify isolation, but obviously sitting down and
> writing such testcases for every c/r implementation is not something we
> can commit to :)
>
> OTOH, perhaps we can collaborate on a test wrapper. This would require
> details to be filled in by each implementation's owner, but would save
> us from each having to come up with the boundary conditions. For
> instance, my testcase for the utsname patches which are in -mm is
> attached. While the testcase is specific to that implementation, by
> abstracting the "start a new container" command into a variable which
> can be filled in for vserver, openvz, etc, we might be able to come up
> with a generic utsname resource testing shell which can be easily filled
> in to work for each implementation.
>
> Just a thought.
>
> -serge

You might like to consider making the output of these tests use the
Perl Test::TAP output; eg:

```
1..5 # declare how many tests you expect to run - 0..0 means unknown
ok 1 # pass test one
not ok 2 # this one failed - perhaps say why in comments
ok 3 # SKIP somereason - this test was skipped
ok 4
ok 5
```

This makes it easier to run the tests inside Harnesses.

Sam.

>
> ---------- ----------
>

```
> /*
>  * Copyright (C) 2005 IBM
>  * Author: Serge Hallyn <serue@us.ibm.com>
>  * Compile using "gcc -o utstest utstest.c"
>  * Run using "for i in `seq 1 5`; do  ./utstest $i; done"
>  *
>  * test1:
>     P1: A=gethostname
>     P2: B=gethostname
>     Ensure(A==B)
>
>  * test2:
>     P1: sethostname(newname); A=gethostname
>     P2: (wait); B=gethostname
>     Ensure (A==B)
>
>  * test3:
>     P1: A=gethostname; unshare(utsname); sethostname(newname);
C=gethostname
>     P2: B=gethostname; (wait); (wait); D=gethostname
>     Ensure (A==B && A==D && C!=D)
>
>  * test4:
>     P1: A=gethostname; unshare(utsname); (wait); C=gethostname
>     P2: B=gethostname; (wait); sethostname(newname); D=gethostname
>     Ensure (A==B && A==C && C!=D)
>
>  * test5:
>     P1: A=gethostname; unshare(utsname) without suff. perms; (wait);
C=gethostname
>     P2: B=gethostname; (wait); sethostname(newname); D=gethostname
>     Ensure (A==B==C==D) and state is ok.
>  *
>  */
>
> #include <sys/wait.h>
> #include <assert.h>
> #include <stdio.h>
> #include <stdlib.h>
> #include <unistd.h>
> #include <string.h>
> #include <errno.h>
>
> int drop_root()
> {
>     int ret;
>     ret = setresuid(1000, 1000, 1000);
>     if (ret) {
```

```
>         perror("setresuid");
>         exit(4);
>     }
>     return 1;
> }
>
> #include <linux/unistd.h>
>
> static inline _syscall1 (int,  unshare, int, flags)
>
> #ifndef CLONE_NEWUTS
> #define CLONE_NEWUTS            0x04000000      /* New utsname group? */
> #endif
>
> int p1fd[2], p2fd[2];
> pid_t cpid;
> int testnum;
>
> #define HLEN 100
> #define NAME1 "serge1"
> #define NAME2 "serge2"
>
> void picknewhostname(char *orig, char *new)
> {
>     memset(new, 0, HLEN);
>     if (strcmp(orig, NAME1) == 0)
>         strcpy(new, NAME2);
>     else
>         strcpy(new, NAME1);
> }
>
> void P1(void)
> {
>     char hostname[HLEN], newhostname[HLEN], rhostname[HLEN];
>     int err;
>     int len;
>
>     close(p1fd[1]);
>     close(p2fd[0]);
>
>     switch(testnum) {
>     case 1:
>         gethostname(hostname, HLEN);
>         len = read(p1fd[0], rhostname, HLEN);
>         if (len == strlen(hostname) &&
>             strncmp(hostname, rhostname, len) == 0) {
>             printf("test 1: success\n");
>             exit(0);
```

```
>       }
>       printf("test 1: fail\n");
>       printf("Proc 1: hostname %s\n", hostname);
>       printf("test 2: hostname %s\n", rhostname);
>       exit(1);
>   case 2:
>       gethostname(hostname, HLEN);
>       picknewhostname(hostname, newhostname);
>       err = sethostname(newhostname, strlen(newhostname));
>       write(p2fd[1], "1", 1);
>       if (err == -1) { perror("sethostname"); exit(1); }
>       len = read(p1fd[0], rhostname, HLEN);
>       if (len == strlen(newhostname) &&
>               strncmp(newhostname, rhostname, len) == 0) {
>           printf("test 2: success\n");
>           exit(0);
>       }
>       printf("test 2: fail\n");
>       printf("Proc 1: hostname %s\n", newhostname);
>       printf("test 2: hostname %s\n", rhostname);
>       exit(1);
>   case 3:
>       gethostname(hostname, HLEN);
>       picknewhostname(hostname, newhostname);
>       err = unshare(CLONE_NEWUTS);
>       printf("unshare returned %d (should be 0)\n", err);
>       err = sethostname(newhostname, strlen(newhostname));
>       write(p2fd[1], "1", 1);
>       if (err == -1) { perror("sethostname"); exit(1); }
>
>       len = read(p1fd[0], rhostname, HLEN);
>       if (len == strlen(newhostname) &&
>               strncmp(newhostname, rhostname, len) == 0) {
>           printf("test 3: fail\n");
>           printf("Proc 1: hostname %s\n", newhostname);
>           printf("test 2: hostname %s\n", rhostname);
>           printf("These should have been different\n");
>           exit(1);
>       }
>       if (len == strlen(hostname) &&
>               strncmp(hostname, rhostname, len) == 0) {
>           printf("test 3: success\n");
>           exit(0);
>       }
>       printf("test 3: fail\n");
>       printf("Proc 1: original hostname %s\n", hostname);
>       printf("Proc 2: hostname %s\n", rhostname);
>       printf("These should have been the same\n");
```

```
>       exit(1);
>
>   case 4:
>       gethostname(hostname, HLEN);
>       err = unshare(CLONE_NEWUTS);
>       printf("unshare returned %d (should be 0)\n", err);
>       write(p2fd[1], "1", 1); /* tell p2 to go ahead and sethostname */
>       len = read(p1fd[0], rhostname, HLEN);
>       gethostname(newhostname, HLEN);
>       if (strcmp(hostname, newhostname) != 0) {
>           printf("test 4: fail\n");
>           printf("Proc 1: hostname %s\n", hostname);
>           printf("Proc 1: new hostname %s\n", newhostname);
>           printf("These should have been the same\n");
>           exit(1);
>       }
>       if (strncmp(hostname, rhostname, len)==0) {
>           printf("test 4: fail\n");
>           printf("Proc 1: hostname %s\n", hostname);
>           printf("Proc 2: new hostname %s\n", rhostname);
>           printf("These should have been different\n");
>           exit(1);
>       }
>       printf("test 4: success\n");
>       exit(0);
>   case 5:
>       /* drop CAP_SYS_ADMIN, then do same as case 4 but check
>        * that hostname != newhostname && rhostname == newhostname */
>       if (!drop_root()) {
>           printf("failed to drop root.\n");
>           exit(3);
>       }
>       gethostname(hostname, HLEN);
>       err = unshare(CLONE_NEWUTS);
>       printf("unshare returned %d (should be -1)\n", err);
>       write(p2fd[1], "1", 1); /* tell p2 to go ahead and sethostname */
>       len = read(p1fd[0], rhostname, HLEN);
>       gethostname(newhostname, HLEN);
>       if (strncmp(newhostname, rhostname, len)!=0) {
>           printf("test 5: fail\n");
>           printf("Proc 1: newhostname %s\n", newhostname);
>           printf("Proc 2: new hostname %s\n", rhostname);
>           printf("These should have been the same\n");
>           exit(1);
>       }
>       if (strcmp(hostname, newhostname) == 0) {
>           printf("test 5: fail\n");
>           printf("Proc 1: hostname %s\n", hostname);
```

```
>            printf("Proc 1: new hostname %s\n", newhostname);
>            printf("These should have been different\n");
>            exit(1);
>        }
>        printf("test 5: success\n");
>        exit(0);
>    default:
>        break;
>    }
>    return;
> }
>
> void P2(void)
> {
>    char hostname[HLEN], newhostname[HLEN];
>    int len;
>    int err;
>
>    close(p1fd[0]);
>    close(p2fd[1]);
>
>    switch(testnum) {
>    case 1:
>        gethostname(hostname, HLEN);
>        write(p1fd[1], hostname, strlen(hostname));
>        break;
>    case 2:
>    case 3:
>        len = 0;
>        while (!len) {
>            len = read(p2fd[0], hostname, 1);
>        }
>        gethostname(hostname, HLEN);
>        write(p1fd[1], hostname, strlen(hostname));
>        break;
>    case 4:
>    case 5:
>        len = 0;
>        while (!len) {
>            len = read(p2fd[0], hostname, 1);
>        }
>        gethostname(hostname, HLEN);
>        picknewhostname(hostname, newhostname);
>        sethostname(newhostname, strlen(newhostname));
>        write(p1fd[1], newhostname, strlen(newhostname));
>        break;
>    default:
>        printf("undefined test: %d\n", testnum);
```

```
>         break;
>     }
>     return;
> }
>
> int main(int argc, char *argv[])
> {
>     if (argc != 2) {
>         printf("Usage: %s <testnum>\n", argv[0]);
>         printf(" where testnum is between 1 and 5 inclusive\n");
>         exit(2);
>     }
>     if (pipe(p1fd) == -1) { perror("pipe"); exit(EXIT_FAILURE); }
>     if (pipe(p2fd) == -1) { perror("pipe"); exit(EXIT_FAILURE); }
>
>     testnum = atoi(argv[1]);
>     if (testnum < 1 || testnum > 5) {
>         printf("testnum should be between 1 and 5 inclusive.\n");
>         exit(2);
>     }
>
>     cpid = fork();
>
>     if (cpid == -1) { perror("fork"); exit(EXIT_FAILURE); }
>
>     if (cpid == 0)
>         P1();
>     else
>         P2();
>
>     return 0;
> }
```

--
Sam Vilain, Catalyst IT (NZ) Ltd.
phone: +64 4 499 2267 cell: +64 21 55 40 50
DDI: +64 4 803 2342 PGP ID: 0x66B25843