

[UNIX] Remote Shell Trojan: Threat, Origin and Solution

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2001-09/0047.html>

From: support@securiteam.com

Date: 09/10/01

From: support@securiteam.com

To: list@securiteam.com

Subject: [UNIX] Remote Shell Trojan: Threat, Origin and Solution

Message-Id: <20010910055339.59E55138C0@mail.der-keiler.de>

Date: Mon, 10 Sep 2001 07:53:39 +0200 (CEST)

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

When was the last time you checked your server's security?

How about a monthly report?

<http://www.AutomatedScanning.com> – Know that you're safe.

Remote Shell Trojan: Threat, Origin and Solution

SUMMARY

At the 5th of September Qualys released a Security Warning regarding a Linux based virus. This virus was called the "Remote Shell Trojan" (RST) and it attacks Linux ELF binaries. It has replicating abilities: when run it will infect all binaries in /bin and the current working directory. Besides that it also spawns a process listening on UDP port 5503. When a properly crafted packet is received by this process it will connect back with a system shell.

DETAILS

Very often viruses are not seen as a real security threat for UNIX. A virus can not infect binaries where the UserID it is running under has no write access to. Even under this situation, viruses can be a threat for UNIX based operating systems: Every time an infected binary is run, it will infect all binaries in the current working directory. It is not unthinkable that a user with increased privileges will later run a binary infected by the RST. In this way, the virus can transparently spread itself over the system. This is especially the case in production

Securiteam: [UNIX] Remote Shell Trojan: Threat, Origin and Solu

environments of in an environment where many users share files. This process will get into a rapid once the /bin binaries are infected. Every execution of normal system commands like 'ls' will infect all binaries in the current working directory. In spite of the theoretical immunity UNIX has, the situation described here is not unlikely to happen in many human situations. The backdoor process can give unprivileged people access to your system under the UserID the backdoor process is running. Attackers can attempt to get higher privileges on the system from there.

Origin:

RST was developed as a research project and intended only for internal use. The goal was to analyze how a non-privileged virus could affect a system running Linux in a normal work-environment. Things however did not go as they were intended to go. An infected binary accidentally leaked out from the research lab and came into the hands of so-called "scriptkiddies". They infected their own systems and other systems where they had access to. From this point, the virus seemed to spread in the wild.

The main concern now is that the spread of this virus is stopped and that all the infected hosts are cleaned as soon as possible. As of now the format of the specially crafted packet, send to the listening backdoor process is unknown to the public. However, this might eventually be reverse engineered in the future and RST can then be actively abused by other people.

Solution:

A set of utilities that can recursively detect and remove the virus from the system has been created. It also has the option to make binaries IMMUNE for future infection by the RST. It is especially recommend that multi-user systems make their system immune for the RST as the risks for these systems are much higher. Immunisation works by increasing the size of the text segment by 4096 bytes so that the "hole" between the text and data segments is gone. After this there is no space for the RST to add it self to the binary anymore.

The interface to these programs is simple and easy to use. The user can decide whether he wants to automatically detect and remove the RST on the system recursively or if he wants to apply the remover on a per binary base. In this mode, the administrator can also get an individual status report on whether this binary is infected, immune, or innocent.

Sample usage would be:

```
% perl Recurse.pl remove
```

For more information, regarding this read the included documentation.

Conclusion:

Again, it is strongly recommend that anybody running Linux run the detector to see if their system is infected. Even if they do not expect

Securiteam: [UNIX] Remote Shell Trojan: Threat, Origin and Solu

```
print "Checking $startdir for infected binaries..\n" if($verbose);

while($filename = readdir($DH)) {
    next if($filename eq "." or $filename eq "..");
    next if(-1 "$startdir/$filename");
    stat("$startdir/$filename");
    RecursiveDeinfect("$startdir/$filename") && next if(-d _);
    next unless(-f _ && -x _);
    $ret = qx($path_to_cleaner $options_to_cleaner '$startdir/$filename'
2>/dev/null);
    if($options_to_cleaner eq "1" && $ret =~ /much alive/m) {
        warn "$startdir/$filename: Trojan Detected.\n";
    } elsif($options_to_cleaner eq "2" && $ret =~ /trojan disabled/m) {
        warn "$startdir/$filename: Trojan Detected and Removed\n";
    }
}

closedir($DH);
}
```

kill.c

```
/* RST trojan manipulator program
 *
 * this is an RST-remover code, it has the following features:
 * 1. Detect RST trojans on the specified executable
 * 2. Disable RST on the specified binary and during that process make
 * the binary immune to further infection attempts
 * 3. Make an innocent, not-infected binary immune to infection attempts
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <elf.h>
#include <errno.h>
#include <sys/stat.h>

/* Disable_trojan()
 * disables a trojan in an infected binary, making the binary immune to
 * any further infection attempts */
int Disable_trojan(FILE *fp)
{
    unsigned int i, padding, poffset, psize, oldentry;
    Elf32_Phdr textseg, dataseg;
    Elf32_Ehdr ehdr;

    // move to the beginning of the file
    if (fseek(fp, 0, SEEK_SET) != 0) goto err;

    // read ELF binary header
    if (fread(&ehdr, sizeof(ehdr), 1, fp) != 1) goto err;
```

Securiteam: [UNIX] Remote Shell Trojan: Threat, Origin and Solu

```
if (ehdr.e_type != ET_EXEC) goto err;

// find text segment and data segment headers
if (fseek(fp, ehdr.e_phoff, SEEK_SET) != 0) goto err;
for (i=0;i<ehdr.e_phnum;i++)
{
if (fread(&textseg, sizeof(textseg), 1, fp) != 1) goto err;
if (textseg.p_offset == 0) break;
}
if (fread(&dataseg, sizeof(dataseg), 1, fp) != 1) goto err;
if (textseg.p_offset != 0)
{
/* wtf?! no text segment ??? */
goto err;
}

// do calculations
padding = dataseg.p_vaddr - (textseg.p_vaddr + textseg.p_filesz);
if (padding >= 4096)
return 0; /* Sheww, binary is not infected */
psize = (textseg.p_vaddr + textseg.p_filesz) - ehdr.e_entry;
poffset = (textseg.p_offset + textseg.p_filesz) - psize;
if (psize != 4096)
return 0; /* Binary already cleaned */

// read original entry point, that according to my reverse engineering
// is stored on the parasite at offset 1
if (fseek(fp, poffset+1, SEEK_SET)!=0) goto err;
if (fread(&oldentry, 4, 1, fp) != 1) goto err;

// restore the binary's entry point to point to the real program again,
// avoiding the execution of the parasite code.
// this permanently disables the parasite code and makes the binary
immune
// to further infection attempts.
ehdr.e_entry = oldentry;
if (fseek(fp, 0, SEEK_SET) != 0) goto err;
if (fwrite(&ehdr, sizeof(ehdr), 1, fp) != 1) goto err;

return 1; /* all done */

err: ;
return -1;
}

/* DisplayStatus()
 * display the infection status of the specified binary */
int DisplayStatus(FILE *fp)
{
unsigned int i, padding, poffset, psize, oldentry;
Elf32_Phdr textseg, dataseg;
```

```

Elf32_Ehdr ehdr;

// move to the beginning of the file
if (fseek(fp, 0, SEEK_SET) != 0) goto err1;

// read ELF binary header
if (fread(&ehdr, sizeof(ehdr), 1, fp) != 1) goto err1;
if (ehdr.e_type != ET_EXEC) goto err1;

// find text segment and data segment headers
if (fseek(fp, ehdr.e_phoff, SEEK_SET) != 0) goto err1;
for (i=0;i<ehdr.e_phnum;i++)
{
if (fread(&textseg, sizeof(textseg), 1, fp) != 1) goto err1;
if (textseg.p_offset == 0) break;
}
if (fread(&dataseg, sizeof(dataseg), 1, fp) != 1) goto err1;
if (textseg.p_offset != 0)
{
/* wtf?! no text segment ??? */
goto err1;
}

// do calculations
padding = dataseg.p_vaddr - (textseg.p_vaddr + textseg.p_filesz);
psize = (textseg.p_vaddr + textseg.p_filesz) - ehdr.e_entry;
poffset = (textseg.p_offset + textseg.p_filesz) - psize;

printf("Estimated parasite offset : %d\n",poffset);
printf("Binary infection status : ");
if (padding >= 4096)
    printf("Not infected. not immune.\n");
else if (psize != 4096)
    printf("Trojan disabled, immune.\n");
else
    printf("Trojan is very much alive!\n");
fflush(stdout);

return 0;
err1 : ;
return -1;
}

int movedata(FILE *fp, int src_offset, int dst_offset, const int size)
{
    char data[size];
    int sloc;

    // save current location
    sloc = ftell(fp);
    // move to source offset and read data

```

Securiteam: [UNIX] Remote Shell Trojan: Threat, Origin and Solu

```
if (fseek(fp, src_offset, SEEK_SET) != 0) return -1;
fread(&data,size,1,fp);
// write data to destination offset
if (fseek(fp, dst_offset, SEEK_SET) != 0) return -1;
fwrite(&data,size,1,fp);
// return to original position
fseek(fp,sloc,SEEK_SET);

return 0;
}

/* make_immune()
 * turns an innocent binary into an uninfected binary */
int make_immune(FILE *fp)
{
    unsigned int i, padding, poffset, psize, oldentry;
    Elf32_Phdr textseg, dataseg;
    struct stat stat;
    Elf32_Ehdr ehdr;

    // get file status
    if (fstat(fileno(fp), &stat)<0) return -1;

    // move to the beginning of the file
    if (fseek(fp, 0, SEEK_SET) != 0) return -1;

    // read ELF binary header
    if (fread(&ehdr, sizeof(ehdr), 1, fp) != 1) return -1;
    if (ehdr.e_type != ET_EXEC) return -1;

    // find text segment and data segment headers
    if (fseek(fp, ehdr.e_phoff, SEEK_SET) != 0) return -1;
    for (i=0;i<ehdr.e_phnum;i++)
    {
        if (fread(&textseg, sizeof(textseg), 1, fp) != 1) return -1;
        if (textseg.p_offset == 0) break;
    }
    if (fread(&dataseg, sizeof(dataseg), 1, fp) != 1) return -1;
    if (textseg.p_offset != 0)
    {
        /* wtf?! no text segment ??? */
        return -1;
    }

    // do calculations
    padding = dataseg.p_vaddr - (textseg.p_vaddr + textseg.p_filesz);
    psize = (textseg.p_vaddr + textseg.p_filesz) - ehdr.e_entry;
    poffset = textseg.p_offset + textseg.p_filesz;

    if (padding < 4096)
        return 0; /* Already infected or already immune */
}
```

```

// Update segment header table
if (fseek(fp, ehdr.e_phoff, SEEK_SET) != 0) return -1;
for (i=0;i<ehdr.e_phnum;i++)
{
Elf32_Phdr phdr;

if (fread(&phdr, sizeof(phdr), 1, fp) != 1) return -1;
if (phdr.p_offset >= poffset)
{
    phdr.p_offset += 4096;
    goto writedown;
}
if (phdr.p_offset == 0)
{
    phdr.p_filesz += 4096;
    phdr.p_memsz += 4096;
    goto writedown;
}

continue;
writedown: ;
if (fseek(fp, -1 * sizeof(phdr), SEEK_CUR) != 0) return -1;
if (fwrite(&phdr, sizeof(phdr), 1, fp) != 1) return -1;
}

// update section header table
if (fseek(fp, ehdr.e_shoff, SEEK_SET) != 0) return -1;
for (i=0;i<ehdr.e_shnum;i++)
{
Elf32_Shdr section;

if (fread(&section, sizeof(section), 1, fp) != 1) return -1;
if (section.sh_offset > poffset)
{
    section.sh_offset += 4096;
    if (fseek(fp, -1 * sizeof(section), SEEK_CUR) != 0) return -1;
    if (fwrite(&section,sizeof(section),1,fp) != 1) return -1;
}
}

// physically move data from (poffset->eof), 4096 bytes forward
if (movedata(fp, poffset, poffset+4096, stat.st_size - poffset) != 0)
    return -1; /* Fuck! */

// all done!

return 1;
}

```

```

int main(int argc, char *argv[])
{
    FILE *fp;
    int op;

    if (argc < 3)
    {
        printf("usage: %s <operation> <path>\n",argv[0]);
        printf("Available operations:\n");
        printf("1 display trojan status on the specified binary\n");
        printf("2 remove the trojan from a specified binary\n");
        printf("3 make an innocent binary immune\n");
        printf("4 remove trojan/if innocent binary then make immune\n");
        printf("5 check if trojan is currently running on the system\n");
        printf("\n");
        return -1;
    }

    op = atoi(argv[1]);

    /* Run status check if operation 5 was selected */
    if (op == 5)
    {
        struct flock lock;
        int fd;

        // try to open the trojan's lockfile
        if ((fd = open("/tmp/982235016-gtkrc-429249277", O_RDWR)) < 0)
        {
            printf("Trojan lockfile does not exist.\n");
            goto testok;
        }
        // try to lock the trojan's lockfile
        memset(&lock, 0, sizeof(lock));
        lock.l_type = F_WRLCK;
        if (fcntl(fd, F_SETLK, &lock) < 0)
        {
            perror("fcntl");
            printf("ALERT! A Trojan remote access process is currently
running!!\n");
            return 0;
        }
        printf("Trojan process is not currently running however trojan traces
were discovered.\n");
        return 0;
        testok: ;
        printf("Trojan is not currently running on this system.\n");
        return 0;
    }

    /* Otherwise, open target binary file */

```

Securiteam: [UNIX] Remote Shell Trojan: Threat, Origin and Solu

```
if (!(fp = fopen(argv[2], "r+b")))
{
fprintf(stderr, "Unable to open %s for reading/writing
(%s)\n",argv[2],strerror(errno));
return -1;
}

/* Implement the rest of the operations */
switch(op)
{
case 2: /* Remove trojan */
{
switch(Disable_trojan(fp))
{
case -1:
fprintf(stderr, "An error has occured (%s)\n",strerror(errno));
return -1;
break;
case 0:
printf("%s: trojan not present\n",argv[2]);
break;
case 1:
printf("%s: trojan disabled\n",argv[2]);
break;
}
}
break;
case 1: /* Display infection status */
DisplayStatus(fp);
break;
case 3: /* Make an innocent binary immune */
switch (make_immune(fp))
{
case -1:
fprintf(stderr, "An error has occured (%s)\n",strerror(errno));
return -1;
break;
case 0:
printf("%s: already immune.\n",argv[2]);
return 0;
break;
case 1:
printf("%s: binary is now immune.\n",argv[2]);
break;
}
}
break;
case 4: /* Remove trojan and make innocent binaries immune */
{
switch(Disable_trojan(fp))
{
case -1:
```

Securiteam: [UNIX] Remote Shell Trojan: Threat, Origin and Solu

```
fprintf(stderr, "An error has occured (%s)\n",strerror(errno));
return -1;
break;
case 0:
    if (make_immune(fp) == 1)
        printf("%s: trojan not present. innocent binary turned
immune.\n",argv[2]);
    else
        printf("%s: immune binary.\n",argv[2]);
    break;
case 1:
    printf("%s: trojan disabled\n",argv[2]);
    break;
    }
}
break;
default:
printf("Operation %d not implemented\n",op);
return -1;
break;
}
}
```

```
makefile
all:
#
clear
#####
# Compiling antivirus program ##
#####
gcc kill.c -o temp

#####
## Making the antivirus binary immune to RST infections ##
#####
cp temp kill ; ./temp 4 kill ; rm -f temp

#####
## All done, run 'perl Recurse.pl' for RST detection, ##
## and removal options. ##
## -- anonymous research team ##
#####
```

ADDITIONAL INFORMATION

The information has been provided by <<mailto:rst@coders.com>> kai takashi.

=====

Securiteam: [UNIX] Remote Shell Trojan: Threat, Origin and Solu

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====
=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.

-
- *Previous message:* support@securiteam.com: "[NEWS] AOLserver Authorization Buffer Overflow"
 - *Messages sorted by:* [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#) [\[attachment \]](#)