The Belt And Suspenders Approach

Mike Schiffman <mike@infonexus.com> (September 1998)

Introduction and Impetus

The OpenBSD project team. Purveyors of a FREE, multi-platform 4.4BSD-based UNIX-like operating system. Their efforts place emphasis on portability, standardization, correctness, security, and cryptography. And OpenBSD really concentrates on those last two. OpenBSD is simply the best choice for multi-user environments.

It is the flawed assumption that security mechanisms can be adequately provided in layers above the operating system. A perfect security application cannot make up for flawed or absent security features within the OS kernel. It is the classic example of building a castle on a swamp. You can build a strong fortress, but it makes no difference if it slowly sinks into the ground. In this article, we retrofit the OpenBSD kernel with some additional security.

This article is about cracking the whip. It's a prime example of security being (possibly) inconvenient. But by making things potentially a bit more difficult for normal users, we hope to severely hamper would-be attackers. Two effective ways of doing this are through limited program execution via path and credential checks and privacy restrictions.

This article is a follow-up to my P52-06 article on hardening the Linux kernel. Herein the reader will find several patches designed to harden a multi-user OpenBSD box. These patches can be broken down into two areas: privacy restriction and execution restriction (more on these below). The patches contained here should be used in conjunction with a savvy for intelligent administration; if you can't recompile a kernel, stop here.

Getting Sources

You will need an OpenBSD 2.4 box with full kernel sources for your architecture and sources for the following programs: w, who, ps, fstat, and ld.so. Below are sample instructions for getting the sources you'll need through anonymous CVS.

I. Pick a server and set the appropriate environment variables:
(assuming csh or tcsh)
1. setenv CVSROOT anoncvs@anoncvs3.usa.openbsd.org:/cvs
2. setenv CVS_RSH /usr/local/bin/ssh
3. cd /usr

II. Get the sources:
1. cvs get src/usr.bin/fstat
2. cvs get src/bin/ps
3. cvs get src/usr.bin/w
4. cvs get src/usr.bin/who
5. cvs get src/gnu/usr.bin/ld/
6. cvs get src/lib/libc/stdio/

III. If you need kernel sources:
(for i386-based machines, other architectures vary slightly)
1. cvs get ksrc-i386 ksrc-common

Privacy Patches

Tested on: 2.4-SNAP (Current as of 12.10.98)
Author: route

Why should we allow anyone to be able to view information on processes they do not own?

Normally, when a process wants system-wide process table information, it retrieves it from the kernel virtual memory interface by making calls to a kvm_*(3) derivative. All that is required is that the process have permissions to read from /dev/kmem (usually meaning the program file needs to be sgid kmem). I am of the school of thought that, unless you are really cool, you don't need to see everyone else's processes on a host. The privacy patches work towards this end.

Privacy Patches Modus Operandi

Simple credential check. Before the command is allowed to dump savory information, a UID check is made. If you're not root, you're not going to see other users' information. Due to the somewhat lazy way this is implemented, a savvy hacker could defeat this. I leave this as an exercise to the reader.

Privacy Patches Installation

I. Extract the code from this article:
1. extract P54-06
2. cd PP/

II. Apply the userland diffs:
1. cp Patch/PP-diff /usr/src
2. cd /usr/src
3. patch < PP-diff

III. Next, cd to the relevant directories and build the executables:
1. cd usr.bin/fstat; make; make install
2. cd usr.bin/who; make; make install
3. cd usr.bin/w; make; make install
4. cd bin/ps; make; make install

Trusted Path / ACL Execution Patches

Tested on: 2.4-SNAP (Current as of 12.10.98)
Author: route

Why should we allow arbitrary code execution rights?

Before any call to sys_execve() is allowed to proceed, we take the vnode of the parent directory that the targeted file lives in and grab the file attributes via the VOP_GETATTR() macro. We then check to see if the path is trusted (root owned directory that isn't group or world writable) and, barring that, we check to see if the user is trusted (on the kernel's trust list). If the last check fails, the file is denied execution privileges.

Oops! By setting certain environment variables, users can still preload libraries and modules filled with all sorts of arbitrary code. This is a no-no. To prevent this, we provide a mechanism to effectively ignore LD_PRELOAD and LD_LIBRARY_PATH environment variables.

TPE Implementation Overview

The tpe suite consists of 4 components: the in-kernel mechanisms, a system call, a userland agent and an ld.so component. The kernel resident components handle the path and credential verification as well as list maintenance. The system call is the vessel used to convey information from userland to the kernel and vice versa. The userland agent consists of the tpe administrative program used to manipulate the trust list (and enable/disable the ld.so environment checker). The ld.so piece is responsible for grooming the environment of any illegal variables.

TPE Trust List Kernel Interface and Abstract Data Types

The trust list inside the kernel is a static array of type `uid_t`. The decision was the made to use a static array to hold the trusted IDs for both convenience and runtime efficiency. By default, the list is elements long. If this for some reason is not sufficient, it can be increased by changing the CPP symbolic constant TPE_ACL_SIZE (however, you should first probably ask yourself why you need more than 80 trusted users).

The speed in which user ID verification is done is absolutely essential, as this check will be done for every call to exec that does not originate from a trusted path. This has the potential to be a huge bottle neck. This was taken into consideration and the bulk of processing overhead is offloaded to list initialization and modification.

The list is kept ordered after all insertions and deletions via insertion sort. Sorting is relatively costly (insertion sort has a running time of about O(n^2)) and is done when response time is not absolutely critical, during list additions and deletions.

Speed is essential when the lookups are done, and, since the list is ordered, a binary search can be done in a worst case of O(lg N). In fact, with the default list size of 80 elements, we can be guaranteed no more than 7 comparisons will be done. Compare that with a sequential search in an ordered list which has a worst case of O(N) (80 comparisons).

TPE ld.so protection

The dynamic linker is a great tool that allows us to write small programs that load external code at runtime. On a macro scale, ld.so allows processes to load arbitrary external code for execution. This can be used to bypass

our execution restrictions. A user could bypass path trust by simply loading code dynamically via library or object code redirection. This is against our best interests. To prevent this, we patch ld.so to strip the LD_PRELOAD and LD_LIBRARY_PATH environment variables.

There is a global int, tpe_ld_check, that is set, cleared and checked via the system call. When set, ld.so checks the environment of any non UID 0 process and calls unsetenv if LD_PRELOAD and/or LD_LIBRARY_PATH exist. The variables still exist in the user's environment, but they are ignored during the dynamic linking.

What TPE will do

Trusted path execution will prevent arbitrary users from executing arbitrary code. This means that malicious users cannot execute exploit programs to try and break root on your machine. This also means that they can't execute exploit programs and try to hack from your machine. It affords an administrator an extra level of confidence that her system is secure.

What TPE will not do

TPE relies on auditing a call to one of exec(2) family of functions. It ensures that the program file that contains the code to be executed resides in a trusted directory or is being executed by a trusted user. Programs living in a trusted directory that interpret symbolic code and link and assemble at runtime (and call exec from a trusted path) can bypass our TPE security mandate and must be audited differently. These are programs such as perl, any of the shell interpreters, sed, awk, etc... While a malicious user cannot just whip up a script in her home directory (it would be denied execution rights because it lives in an untrusted directory) she could specify the code on the command line or redirect it from a file.

There are different ways to tackle this problem, none of them very elegant. Changing the file permissions and ownership to allow only members of a certain group access to these files is a simple effort and an obvious choice, but will not work for the shell interpreters. Moving all of these programs to a special non-trusted directory would also work (normal users would not be able to execute them, but trusted users would), but again, this will not work for the shell programs.

To prevent the shell programs from being to execute arbitrary code it seems like the only real solution would be to patch them. This way you can prevent naughty activity and still get desired functionality.

Another area of trouble is command line buffer overflows. If a trusted program happens to contain a buffer overflow that is exploitable from the command line, an attacker can bypass the TPE and get arbitrary code executed. The overflow shellcode is passed in as standard command line argument and is not illegal as far as TPE sees. One possible fix is to audit or sanitize the command arguments before granting execution rights.

The other noteworthy issue regarding TPE is the fact that it generally does not protect the machine from remote attacks. Daemons running as root or as a trusted user id (usually the case -- otherwise how would it be started in the first place?) will be allowed execution rites. If this code contains

remotely exploitable buffer overflows, TPE cannot prevent arbitrary code execution.

tpe_adm

The userland agent is painfully simple to use. To show the kernel's trusted user list:

```
resentment:~# tpe_adm -s
trusted users: root diablerie
```

To add a user to the list:

```
resentment:~# tpe_adm -a devilish
UID 1000 added to trust list
resentment:~# tpe_adm -s
trusted users: root diablerie devilish
```

To remove a user from the list:

```
resentment:~# tpe_adm -d diablerie
UID 1000 removed from trust list
resentment:~# tpe_adm -s
trusted users: root diablerie
```

To enable/disable ld.so environment checking:

```
resentment:~# tpe_adm -le
ld.so environment protection enabled
resentment:~# tpe_adm -ls
ld.so environment protection is currently on
resentment:~# tpe_adm -ld
ld.so environment protection disabled
resentment:~# tpe_adm -ls
ld.so environment protection is currently off
```

TPE Installation

I. Extract the code from this article:
   1. extract P54-06
   2. cd TPE/

II. Apply the kernel diffs:
   1. cp Core/Patch/TPE-diff /usr/src/sys
   2. cd /usr/src/sys/
   3. patch < TPE-diff
   4. note any errors. hope they are benign.

III. Apply the ld.so diff:
   1. cp Core/Patch/ld.so-diff /usr/src/
   2. cd /usr/src/
   3. patch < ld.so-diff

IV. Copy over the tpe core files:
   1. cp Core/kern/kern_tpe.c /usr/src/sys/kern

2. cp Core/kern/kern_tpe_sys.c /usr/src/sys/kern
3. cp Core/sys/kern_tpe.h /usr/src/sys/sys

V. Rebuild your syscall table:
1. cd /usr/src/sys/kern
2. make

VI. Copy over the syscall include files:
1. cp /usr/src/sys/sys/syscall.h /usr/include/sys
2. cp /usr/src/sys/sys/syscallargs.h /usr/include/sys

VII. Reconfigure your kernel:
(This step assumes you have a previously configured kernel named YOUR_KERNEL. If you haven't, you need to config a kernel. Refer to OpenBSD documentation on how to do this.)
1. cd /usr/src/sys/arch/YOUR_ARCH/conf
2. config YOUR_KERNEL

VIII. Remake the dependencies and rebuild the kernel:
1. cd /usr/src/sys/arch/YOUR_ARCH/compile/YOUR_KERNEL
2. make depend ; make clean ; make
3. note any errors. hope you can fix them.
3. cp /bsd /bsd.old ; cp bsd /
4. reboot

IX. Build the new ld.so
1. cd /usr/src
2. cp lib/libc/stdio/vfprintf.c /usr/src/gnu/usr.bin/ld/rtld
3. cp lib/libc/stdio/local.h /usr/src/gnu/usr.bin/ld/rtld
4. cp lib/libc/stdio/fvwrite.h /usr/src/gnu/usr.bin/ld/rtld
5. cd /usr/src/gnu/usr.bin/ld/rtld
6. make ; make install

X. Build the TPE admin program:
1. cd Core/Admin/ ; make
2. make install

XI. Test it out:
1. As root, dump the current trust list:
(Only UID 0 should be on it.)
tpe_adm -s
trusted users: root

2. Try the following as an untrusted user (i.e. UID=1000):
cat > foo.c << EOF ; gcc foo.c
int main(int argc, char **argv){ printf("Hello world\n"); }
EOF
./a.out
EPERM should result.

3. Now add the user to the trust list:
tpe_adm -a UID

4. Dump the list again:
(You should see the user on the list.)
tpe_adm -s

5. Try to execute the command again as the user:
   ./a.out
   Hello world

6. Add only the necessary UIDs to the list.

7. NOTE TO QMAIL USERS:
   You may find that you will need to explicitly add the qmailq
   UID to the trust list. Do this in an rc startup script that
   runs before the qmail daemons start.

8. As root, ensure that ld.so environment protection is enabled:
   tpe_adm -le
   ld.so environment protection enabled

9. As an unprivileged user:
   setenv LD_PRELOAD test.o
   ls -l
   Your environment contains illegal variables which are being
   stripped out for the execution of this program
   a.out fo.c   foo.c

10. As root, ensure that ld.so environment protection is disabled:
    tpe_adm -ld
    ld.so environment protection disabled

11. As an unprivileged user:
    ls
    /usr/libexec/ld.so: preload: test.o: cannot map object

12. You're done. Pat yourself on the back and buy something from
    Precious Roy.

The Code

```
<++> TPE/Core/Admin/Makefile
#   $Id: P54-06,v 1.16 1998/12/10 00:01:28 route Exp $
#   Trusted path ACL implementation for OpenBSD 2.4
#   Copyright (c) 1998 route|daemon9 and Mike D. Schiffman
#   All rights reserved.
#
#   Originally published in Phrack Magazine (http://www.phrack.com).

tpe_adm:
      $(CC) tpe_adm.c -o tpe_adm

install: tpe_adm
      install -m 711 -o 0 tpe_adm /usr/local/sbin

clean:
      rm -rf core a.out tpe_adm

# EOF
<-->
<++> TPE/Core/Admin/tpe_adm.c
/*
```

```
 *  $Id: P54-06,v 1.16 1998/12/10 00:01:28 route Exp $
 *  Trusted path ACL userland administrative agent for OpenBSD 2.4
 *
 *  Copyright (c) 1998 route|daemon9 and Mike D. Schiffman
 *  All rights reserved.
 *  Originally published in Phrack Magazine (http://www.phrack.com).
 *
 *  Thanks to nirva for helping me choose an ADT.
 *  See <sys/kern_tpe.h> for more info.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <pwd.h>
#include "../sys/kern_tpe.h"

void usage();

int
main(int argc, char **argv)
{
    uid_t list[TPE_ACL_SIZE];
    int c, i, mode;
    uid_t candidate;
    struct passwd *pwd;

    if (geteuid() && getuid())
    {
        fprintf(stderr, "root access required\n");
        exit(1);
    }
```

```
if (argc == 1 || argc > 3)
{
    usage();
    exit(EXIT_SUCCESS);
}

while ((c = getopt(argc, argv, "a:d:l:s")) != EOF)
{
    switch (c)
    {
        case 'a':
            if (isalpha(optarg[0]))
            {
                pwd = getpwnam(optarg);
                if(!pwd)
                {
                    fprintf(stderr, "Unknown user: \"%s\"\n", optarg);
                    exit(EXIT_FAILURE);
                }
                candidate = pwd->pw_uid;
            }
            else if (!(candidate = (uid_t)atol(optarg)))
            {
                fprintf(stderr, "invalid UID: \"%s\"\n", optarg);
                exit(EXIT_FAILURE);
            }
            if (syscall(SYS_tpe_adm, TPE_ADD, candidate, NULL) == -1)
            {
                printf("Full trust list\n");
                exit(EXIT_FAILURE);
            }
            printf("UID %d added to trust list\n", candidate);
            break;
        case 'd':
            if (isalpha(optarg[0]))
            {
                pwd = getpwnam(optarg);
                if(!pwd)
                {
                    fprintf(stderr, "Unknown user: \"%s\"\n", optarg);
                    exit(EXIT_FAILURE);
                }
                candidate = pwd->pw_uid;
            }
            else if (!(candidate = (uid_t)atol(optarg)))
            {
                fprintf(stderr, "invalid UID: \"%s\"\n", optarg);
                exit(EXIT_FAILURE);
            }
            if (syscall(SYS_tpe_adm, TPE_REMOVE, candidate, NULL) == -1)
            {
                printf("UID %d not found on trust list\n", candidate);
                exit(EXIT_FAILURE);
            }
            printf("UID %d removed from trust list\n", candidate);
            break;
        case 'l':
```

```
                if (optarg[0] == 'e')
                {
                    if (syscall(SYS_tpe_adm, TPE_LDCHECK_E, -1, NULL) == -1)
                    {
                        printf("Unknown internal error\n"); /* should NOT fail
*/
                        exit(EXIT_FAILURE);
                    }
                    printf("ld.so environment protection enabled\n");
                }
                else if (optarg[0] == 'd')
                {
                    if (syscall(SYS_tpe_adm, TPE_LDCHECK_D, -1, NULL) == -1)
                    {
                        printf("Unknown internal error\n"); /* should NOT fail
*/
                        exit(EXIT_FAILURE);
                    }
                    printf("ld.so environment protection disabled\n");
                }
                else if (optarg[0] == 's')
                {
                    if (syscall(SYS_tpe_adm, TPE_LDCHECK_S, -1, list) == -1)
                    {
                        printf("Unknown internal error\n"); /* should NOT fail
*/
                        exit(EXIT_FAILURE);
                    }
                    printf("ld.so environment protection is currently %s\n",
                    list[0] ? "on" : "off");
                }
                else
                {
                    fprintf(stderr, "Huh?\n");
                    exit(EXIT_FAILURE);
                }
                break;
            case 's':
                /*
                 *  It is Very Important that `list` is an array of size
                 *  TPE_ACL_SIZE.  The kernel expects this.  Failure to do
                 *  so can result in a panic.  However, only root can issue
                 *  the tpe_adm system call.
                 */
                if (syscall(SYS_tpe_adm, TPE_SHOW, -1, list) == -1)
                {
                    /*
                     *  Should NOT fail.
                     */
                    printf("Hideous internal error\n");
                    exit(EXIT_FAILURE);
                }
                printf("trusted users: ");
                for (i = 0; list[i] != TPE_INITIALIZER; i++)
                {
                    pwd = getpwuid(list[i]);
                    if (pwd)
```

```
                    {
                        printf("%s ", pwd->pw_name);
                    }
                    else
                    {
                        printf("%d ", (int)list[i]);
                    }
                }
                printf("\n");
                break;
            default:
                usage();
                exit(EXIT_SUCCESS);
        }
    }
    return (0);
}


void
usage()
{
    fprintf(stderr, "usage: tpe_adm [-a UID]    Add a UID to the trust list\n"
                                    "[-d UID]    Delete a UID from the list\n"
                                    "[-l e|n]    Toggle LD_* usage\n"
                                    "[-l s]      Show status of ld.so
protection\n"
                                    "[-s]        Show the current list\n");
}
<-->
<++> TPE/Core/Patch/TPE-diff
--- ./kern/init_main.c  Tue Sep 15 23:21:08 1998
+++ ../Core/kern/init_main.c  Sun Oct 18 12:26:24 1998
@@ -80,6 +80,7 @@

 #include <sys/syscall.h>
 #include <sys/syscallargs.h>
+#include <sys/kern_tpe.h>

 #include <ufs/ufs/quota.h>

@@ -424,6 +425,16 @@
      srandom((u_long)(rtv.tv_sec ^ rtv.tv_usec));

      randompid = 1;
+
+        tpe_init();
+        printf("Trusted patch execution list initialized\n");
+        /*
+         *  root must be added hard at this point.  For safey's sake, the
+         *  userland agent can't do anything with UID 0 to prevent morons
+         *  from locking themselves out of their machines.
+         */
+        tpe_add(0);
+
      /* The scheduler is an infinite loop. */
      scheduler();
```

```
      /* NOTREACHED */
--- ./kern/syscalls.master    Thu Sep 17 13:54:04 1998
+++ ../Core/kern/syscalls.master    Sun Oct 18 12:35:59 1998
@@ -479,7 +479,8 @@
 242  UNIMPL
 243  UNIMPL
 244  UNIMPL
-245  UNIMPL
+245  STD             { int sys_tpe_adm(int mode, uid_t candidate, \
+                           uid_t *list); }
 246  UNIMPL
 247  UNIMPL
 248  UNIMPL
--- ./kern/kern_exec.c  Thu Sep 24 11:49:31 1998
+++ ../Core/kern/kern_exec.c  Sun Oct 18 12:32:03 1998
@@ -51,12 +51,16 @@
 #include <sys/mman.h>
 #include <sys/signalvar.h>
 #include <sys/stat.h>
+#include <ufs/ufs/quota.h>
+#include <ufs/ufs/inode.h>
 #ifdef SYSVSHM
 #include <sys/shm.h>
 #endif

 #include <sys/syscallargs.h>
-
+#include <sys/kern_tpe.h>
+#include <sys/systm.h>
+
 #include <vm/vm.h>
 #include <vm/vm_kern.h>

@@ -93,6 +97,7 @@
      struct exec_package *epp;
 {
      int error, i;
+        struct vattr at;
      struct vnode *vp;
      struct nameidata *ndp;
      size_t resid;
@@ -146,6 +151,30 @@
      if (error)
            goto bad2;
      epp->ep_hdrvalid = epp->ep_hdrlen - resid;
+
+        /*
+         *  Get the file attributes of the parent directory that the
+         *  executable lives in.
+         */
+        if ((error = VOP_GETATTR(ndp->ni_dvp, &at, NULL, NULL)) != 0)
+        {
+            goto bad2;
+        }
+
+        /*
+         *  Trusted path check.
```

```
+         */
+        if (!TRUSTED_PATH(at))
+        {
+                /*
+                 *  Trusted user check.
+                 */
+                if (!TRUSTED_USER(p->p_ucred->cr_uid))
+                {
+                        error = EACCES;
+                        goto bad2;
+                }
+        }

       /*
        * set up the vmcmds for creation of the process
--- ./conf/files  Sun Sep 27 19:43:22 1998
+++ ../Core/conf/files  Sun Oct 18 12:40:28 1998
@@ -209,6 +209,8 @@
 file kern/kern_sysctl.c
 file kern/kern_synch.c
 file kern/kern_time.c
+file kern/kern_tpe.c
+file kern/kern_tpe_sys.c
 file kern/kern_xxx.c
 file kern/subr_autoconf.c
 file kern/subr_disk.c
<-->
<++> TPE/Core/kern/kern_tpe.c
/*
 *  $Id: P54-06,v 1.16 1998/12/10 00:01:28 route Exp $
 *  Trusted path ACL implementation for OpenBSD 2.4
 *
 *  Copyright (c) 1998 route|daemon9 and Mike D. Schiffman
 *  All rights reserved.
 *  Originally published in Phrack Magazine (http://www.phrack.com).
 *
 *  Thanks to nirva for helping me choose an ADT.
 *  See <sys/kern_tpe.h> for more info.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
```

```
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 */

#include <sys/kern_tpe.h>

void
tpe_init()
{
    memset(tpe_acl, TPE_INITIALIZER, sizeof(uid_t) * TPE_ACL_SIZE);
    tpe_acl_candidates = 0;
    tpe_ld_check = 1;
#if (AUTO_ADD_ROOT)
    tpe_acl[0] = 0;
#endif
}


void
tpe_show()
{
    int i;

    printf("%d trusted users: ", tpe_acl_candidates);
    for (i = 0; i < tpe_acl_candidates; i++)
    {
        printf("%d ", tpe_acl[i]);
    }
    printf("\n");
}


int
tpe_add(uid_t candidate)
{
    if (tpe_acl_candidates == TPE_ACL_SIZE)
    {
        /*
         *  Full list.
         */
        return (NACK);
    }

    /*
     *  Don't add duplicates.
     */
    if ((tpe_search(candidate, 0, tpe_acl_candidates)) == NACK)
    {
      /*
       *    Add to the end of the list, then sort.
       */
      tpe_acl_candidates++;
      tpe_acl[tpe_acl_candidates] = candidate;
      tpe_sort(0, tpe_acl_candidates);

        printf("tpe: UID %d added to trust list\n", candidate);
```

```
    }
    else
    {
        printf("tpe: duplicate UID %d not added\n", candidate);
    }
    return (ACK);
}


int
tpe_remove(uid_t candidate)
{
    int n;

    if (tpe_acl_candidates == 0)
    {
        /*
         *  Empty list.
         */
        return (NACK);
    }
    if ((n = tpe_search(candidate, 0, tpe_acl_candidates)) != NACK)
    {
        /*
         *  Remove the candidate (mark the slot as unused), resort the list.
         */
        tpe_acl[n] = TPE_INITIALIZER;
        tpe_acl_candidates--;
        tpe_sort(0, tpe_acl_candidates);

        printf("tpe: UID %d removed from trust list\n", candidate);
        return (ACK);
    }
    /*
     *  Not found.
     */
    return (NACK);
}


int
tpe_verify(uid_t candidate)
{
    if ((tpe_search(candidate, 0, tpe_acl_candidates)) != NACK)
    {
        return (ACK);
    }
    else
    {
        return (NACK);
    }
}


void
tpe_sort(int low, int high)
{
```

```
    int i, j, n;

    /*
     *  Standard insertion sort.
     */
    for (i = low + 1; i <= high; i++)
    {
        COMPSWAP(tpe_acl[low], tpe_acl[i]);
    }

    for (i = low + 2; i <= high; i++)
    {
        j = i;
        n = tpe_acl[i];
        while (LESS(n, tpe_acl[j - 1]))
        {
            tpe_acl[j] = tpe_acl[j - 1];
            j--;
        }
        tpe_acl[j] = n;
    }
}


int
tpe_search(uid_t candidate, int low, int high)
{
    int n;

    /*
     *  Standard binary search.  XXX - should be iterative.
     */
    n = (low + high) / 2;

    if (low > high)
    {
        return (NACK);
    }
    if (candidate == tpe_acl[n])
    {
        return (n);
    }
    if (low == high)
    {
        return (NACK);
    }
    if (LESS(candidate, tpe_acl[n]))
    {
        return (tpe_search(candidate, low, n - 1));
    }
    else
    {
        return (tpe_search(candidate, n + 1, high));
    }
}

/* EOF */
```

```
<-->
<++> TPE/Core/kern/kern_tpe_sys.c
/*
 *  $Id: P54-06,v 1.16 1998/12/10 00:01:28 route Exp $
 *  Trusted path ACL syscall implementation for OpenBSD 2.4
 *
 *  Copyright (c) 1998 route|daemon9 and Mike D. Schiffman
 *  All rights reserved.
 *  Originally published in Phrack Magazine (http://www.phrack.com).
 *
 *  Thanks to nirva for helping me choose an ADT.
 *  See <sys/kern_tpe.h> for more info.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 */

#include <sys/kern_tpe.h>
#include <sys/systm.h>

#include <sys/mount.h>
#include <sys/syscallargs.h>


int
sys_tpe_adm(p, v, retval)
        struct proc *p;
        void *v;
        register_t *retval;
{
    struct sys_tpe_adm_args /* {
        syscallarg(int) mode;
        syscallarg(uid_t) candidate;
        syscallarg(uid_t *) list;
    } */ *uap = v;
    register struct pcred *pc = p->p_cred;
    register int i;
    register uid_t *lp;
```

```
/*
 *  The only thing a non root user can do is check the status of the
 *  ld.so environment protection.  This is necessary because ld.so
 *  runs without elevated privilidges and needs to check this.
 */
if (suser(pc->pc_ucred, &p->p_acflag) && SCARG(uap, mode) != TPE_LDCHECK_S)
{
     return (EPERM);
}

switch (SCARG(uap, mode))
{
    case TPE_ADD:
        if (tpe_add(SCARG(uap, candidate)) == ACK)
        {
            return (0);
        }
        else
        {
            return (ENOSPC);            /* Ugh.  Best we can do. */
        }
    case TPE_REMOVE:
        if (tpe_remove(SCARG(uap, candidate)) == ACK)
        {
            return (0);
        }
        else
        {
            return (ENOSPC);            /* Ugh. */
        }
    case TPE_SHOW:
        lp = SCARG(uap, list);
        if (lp == NULL)
        {
            return (ENOSPC);
        }
        else
        {
            for (i = 0; i < TPE_ACL_SIZE; i++)
            {
                lp[i] = tpe_acl[i];
            }
            return (0);
        }
    case TPE_LDCHECK_E:
        tpe_ld_check = 1;
        return (0);
    case TPE_LDCHECK_D:
        tpe_ld_check = 0;
        return (0);
    case TPE_LDCHECK_S:
        lp = SCARG(uap, list);
        if (lp == NULL)
        {
            return (ENOSPC);
        }
```

```
            else    /* XXX - sysctl would be cleaner. */
            {
                lp[0] = tpe_ld_check;
                return (0);
            }
        default:
            return (ENXIO);                 /* Ugh. */
    }
    return (ENXIO);
}
<-->
<++> TPE/Core/sys/kern_tpe.h
/*
 *  $Id: P54-06,v 1.16 1998/12/10 00:01:28 route Exp $
 *  Trusted path ACL implementation for OpenBSD 2.4
 *
 *  Copyright (c) 1998 route|daemon9 and Mike D. Schiffman
 *  All rights reserved.
 *  Originally published in Phrack Magazine (http://www.phrack.com).
 *
 *  Thanks to nirva for helping me choose an ADT.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *      Trusted path ACL implementation for OpenBSD 2.4
 *
 *      For the full write-up please see Phrack Magazine, issue 54, article 6
 *      http://www.phrack.com
 *
 *      Overview:
 *
 *  A trusted path/ACL execution implementation for OpenBSD.  We consider
 *  a path to be trusted if the parent directory is owned by root and is not
 *  group or world writable.  We consider a user to be trusted if she is on
 *  the kernels trust list.
 *
 *      Implementation details:
 *
```

```
 *  Inside the kern_exec function, we first check the path for trust, if that
 *  fails, we then check the user's credentials to see if she is  able to run
 *  binaries in an untrusted path.  Untrusted users are not allowed to execute
 *  programs from untrusted pathes.
 *
 *  The decision was the made to use a static array to hold the trusted IDs
 *  for both convienience and runtime efficiency.  We keep the list ordered
 *  after all insertions and deletions, and therefore, we can search the list
 *  (where speed is critical) in a worst case of O(lg N).  Compare that with a
 *  sequential search in an ordered list which has a worst case of O(N).
 *
 *  The speed in which user ID verification is done is absolutely essential,
 *  as this check will be done for every call to exec that does not originate
 *  from a trusted path.  This has the potential to be a huge bottle neck.
 *  This was taken into consideration and the bulk of processing overhead is
 *  offloaded to list initialization and modification.
 */

#ifndef __KERN_TPE_H
#define     __KERN_TPE_H

#ifdef _KERNEL
#include <sys/types.h>
#include <sys/cdefs.h>
#include <sys/systm.h>
#include <sys/param.h>
#include <sys/ucred.h>
#include <sys/proc.h>
#endif

/*
 *  syscall stuff
 */
#define TPE_ADD         0       /* add an entry */
#define TPE_REMOVE      1       /* delete an entry */
#define TPE_SHOW        2       /* show the list */
#define TPE_LDCHECK_E   3       /* enable ld.so environment checking */
#define TPE_LDCHECK_D   4       /* disable ld.so environment checking */
#define TPE_LDCHECK_S   5       /* show ld.so environment check status */

#define TPE_ACL_SIZE    80      /* Shouldn't need to be larger */
#define TPE_INITIALIZER -1      /* A UID that isn't used */

#define ACK             1       /* positive acknowledgement */
#define NACK            -1      /* negative acknowledgement */

#define LESS(X, Y)      (X < Y)
#define SWAP(X, Y)      (X ^= Y, Y ^= X, X ^= Y)
#define COMPSWAP(X, Y)  if (LESS(Y, X)) SWAP(X, Y)

/*
 *  Verify the path.  This macro is passed a filled in attr struct via
 *  VOP_GETATTR.
 */
#define TRUSTED_PATH(AT) \
(!(AT.va_mode & (S_IWGRP | S_IWOTH)) && (AT.va_uid == 0))
```

```
/*
 *  Verify the user.  This macro is passed the user's ID from the u_cred
 *  struct.
 */
#define TRUSTED_USER(UID) (tpe_verify(UID) == ACK)

uid_t tpe_acl[TPE_ACL_SIZE];    /* trusted user list */
int tpe_acl_candidates;         /* number of users on the list */
int tpe_ld_check;               /* check ld.so env */

/*
 *  Initialize the array with default values (TPE_INITIALIZER).
 */
void
tpe_init __P((
    void
    ));


/*
 *  Dump the list.
 */
void
tpe_show __P((
    void
    ));


/*
 *  Attempt to add a candidate to the list.  Only fails if the list is full.
 */
int
tpe_add __P((
    uid_t   /* candidate user for addition */
    ));


/*
 *  Attempt to remove a candidate from the list.  Only fails if the entry is
 *  not there.
 */
int
tpe_remove __P((
    uid_t       /* candidate user for deletion */
    ));


/*
 *  Verify a candidate user.
 */
int
tpe_verify __P((
    uid_t       /* candidate user for verification */
    ));


/*
```

```
 *  Insertion sort the list.
 */
void
tpe_sort __P((
    int,        /* list low element */
    int         /* list high high element */
    ));


/*
 *  Locate a uid in the list, standard recursive binary search, running in
 *  worst case of lg N.
 */
int
tpe_search __P((
    uid_t,      /* candidate user to search for */
    int,        /* list low element */
    int         /* list high high element */
    ));

#endif      /* __KERN_TPE_H */
/* EOF */
<-->
<++> PP/Patch/PP-diff
--- ./usr.bin/fstat/fstat.c.orig    Tue Oct 20 10:43:58 1998
+++ ./usr.bin/fstat/fstat.c   Tue Oct 20 10:47:22 1998
@@ -158,6 +158,7 @@
      char *memf, *nlistf;
      char buf[_POSIX2_LINE_MAX];
      int cnt;
+        pid_t __uid;

      arg = 0;
      what = KERN_PROC_ALL;
@@ -248,7 +249,12 @@
      else
            putchar('\n');

+        __uid = getuid();
      for (plast = &p[cnt]; p < plast; ++p) {
+               if (__uid)
+               {
+                   if (p->kp_eproc.e_pcred.p_ruid != __uid) continue;
+               }
            if (p->kp_proc.p_stat == SZOMB)
                 continue;
            dofiles(p);
--- ./bin/ps/ps.c.orig  Tue Oct 20 10:48:40 1998
+++ ./bin/ps/ps.c Tue Oct 20 10:51:26 1998
@@ -112,6 +112,7 @@
      dev_t ttydev;
      pid_t pid;
      uid_t uid;
+        uid_t __uid;
      int all, ch, flag, i, fmt, lineno, nentries;
      int prtheader, wflag, what, xflg;
      char *nlistf, *memf, *swapf, errbuf[_POSIX2_LINE_MAX];
```

```
@@ -281,6 +282,8 @@
      if (!all && ttydev == NODEV && pid == -1)  /* XXX - should be cleaner */
            uid = getuid();

+        __uid = getuid();
+
      /*
       * scan requested variables, noting what structures are needed,
       * and adjusting header widths as appropiate.
@@ -330,6 +333,20 @@
      for (i = lineno = 0; i < nentries; i++) {
            KINFO *ki = &kinfo[i];

+               /*
+                *  root gets to see the whole proccess list.
+                */
+               if (__uid)
+               {
+                   /*
+                    *  If the process in question is not our own, we do not
+                    *  get to see it.
+                    */
+                   if (kinfo[i].ki_p->kp_eproc.e_pcred.p_ruid != __uid)
+                   {
+                       continue;
+                   }
+               }
            if (xflg == 0 && (KI_EPROC(ki)->e_tdev == NODEV ||
                (KI_PROC(ki)->p_flag & P_CONTROLT ) == 0))
                 continue;
--- ./usr.bin/w/w.c.orig      Tue Oct 20 10:52:02 1998
+++ ./usr.bin/w/w.c     Tue Oct 20 10:54:46 1998
@@ -131,6 +131,7 @@
      int ch, i, nentries, nusers, wcmd;
      char *memf, *nlistf, *p, *x;
      char buf[MAXHOSTNAMELEN], errbuf[_POSIX2_LINE_MAX];
+        uid_t __uid;

      /* Are we w(1) or uptime(1)? */
      p = __progname;
@@ -332,6 +333,14 @@
                      ep->utmp.ut_host + UT_HOSTSIZE - x, x);
                 p = buf;
            }
+               __uid = getuid();
+               if (__uid)
+               (void)printf("%-*.*s %-2.2s %-*.*s ",
+                   UT_NAMESIZE, UT_NAMESIZE, ep->utmp.ut_name,
+                   strncmp(ep->utmp.ut_line, "tty", 3) ?
+                   ep->utmp.ut_line : ep->utmp.ut_line + 3,
+                   UT_HOSTSIZE, UT_HOSTSIZE, "<skulking about>");
+               else
            (void)printf("%-*.*s %-2.2s %-*.*s ",
                UT_NAMESIZE, UT_NAMESIZE, ep->utmp.ut_name,
                strncmp(ep->utmp.ut_line, "tty", 3) ?
@@ -339,7 +348,14 @@
                UT_HOSTSIZE, UT_HOSTSIZE, *p ? p : "-");
```

```
            pr_attime(&ep->utmp.ut_time, &now);
            pr_idle(ep->idle);
-           pr_args(ep->kp);
+                if (__uid)
+                {
+                    printf("<this n' that>");
+                }
+                else
+                {
+                    pr_args(ep->kp);
+                }
            printf("\n");
      }
      exit(0);
--- ./usr.bin/who/who.c.orig  Tue Aug 19 22:37:21 1997
+++ ./usr.bin/who/who.c Tue Oct 20 10:57:04 1998
@@ -227,6 +227,7 @@
      char state = '?';
      static time_t now = 0;
      time_t idle = 0;
+        uid_t __uid;

      if (show_term || show_idle) {
            if (now == 0)
@@ -265,8 +266,15 @@
                  (void)printf(" old  ");
      }

-     if (*up->ut_host)
-           printf("\t(%.*s)", UT_HOSTSIZE, up->ut_host);
+        __uid = getuid();
+        if (__uid)
+        {
+            printf("\t<skulking about>");
+        }
+        else if (*up->ut_host)
+        {
+            printf("\t(%.*s)", UT_HOSTSIZE, up->ut_host);
+        }
      (void)putchar('\n');
 }
<-->
<++> TPE/Core/Patch/ld.so-diff
--- gnu/usr.bin/ld/rtld/rtld.c.old  Thu Oct 22 20:44:52 1998
+++ gnu/usr.bin/ld/rtld/rtld.c      Sat Oct 24 16:44:00 1998
@@ -39,6 +39,8 @@
 #include <sys/resource.h>
 #include <sys/errno.h>
 #include <sys/mman.h>
+#include <sys/syscall.h>
+#include "/usr/src/sys/sys/kern_tpe.h"
 #ifndef MAP_COPY
 #define MAP_COPY MAP_PRIVATE
 #endif
@@ -150,7 +152,9 @@
 static uid_t           uid, euid;
 static gid_t           gid, egid;
```

```
 static int       careful;
+static int       tpe_ld_strip;
 static int       anon_fd = -1;
+static uid_t            list[TPE_ACL_SIZE];

 struct so_map          *link_map_head, *main_map;
 struct so_map          **link_map_tail = &link_map_head;
@@ -271,7 +275,20 @@

       careful = (uid != euid) || (gid != egid);

-      if (careful) {
+         if (syscall(SYS_tpe_adm, TPE_LDCHECK_S, -1, list) == -1)
+         {
+             fprintf(stderr, "Unknown internal error\n"); /* should NOT fail */
+             exit(EXIT_FAILURE);
+         }
+         if (list[0] && uid)
+         {
+             if (getenv("LD_PRELOAD") || getenv("LD_LIBRARY_PATH"))
+             {
+                 fprintf(stderr, "Your environment contains illegal variables
which are being stripped out for the execution of this program.\n");
+             }
+             tpe_ld_strip = 1;
+         }
+      if (careful || tpe_ld_strip) {
             unsetenv("LD_LIBRARY_PATH");
             unsetenv("LD_PRELOAD");
       }
<-->


EOF
```