static void insecure (localhost *unix)

Eric Pancer

epancer@infosec.depaul.edu

Information Security Team

DePaul University

http://infosec.depaul.edu

Overview

- This presentation intends to help the audience better understand both security risks and techniques to combat local attacks against a UNIX host.
- We will focus on local security and not address any of the security risks involved in attaching to a network.
- <Blurb on why local security is important>

Scope

- The focus of this talk will be on “newer” UNIX variants that have become more common over the past 10 years. These include
  - Linux
  - BSD 4.4 variants
    - OpenBSD
    - FreeBSD
- While these aren’t the only UNIX variants alive and well, they feature some of the more progressive security improvements.

What We Won't Cover

While this talk will not focus on network security, it *assumes* that you have. . .

- Configured network applications to run under unique, unprivileged accounts.
- Disabled services not vital for production.
- Disabled unauthenticated services (r* services, anonymous ftp, fingerd).
- Implemented kernel level TCP/IP filtering and hardened your TCP/IP stack.
- Actively monitor traffic to and from the host.

Assumptions

This talk assumes that you have a working knowledge of UNIX and understand the basics of a kernel, file-system, privileges, etc..

Preface

The Good News

- Simple model: userland vs. privileged space.

The Good News

- Simple model: userland vs. privileged space.
- (Relatively) easy to audit via `syslogd(8)`.

The Good News

- Simple model: userland vs. privileged space.
- (Relatively) easy to audit via `syslogd(8)`.
- Aged,

The Good News

- Simple model: userland vs. privileged space.
- (Relatively) easy to audit via `syslogd(8)`.
- Aged,
  - Well understood,

The Good News

- Simple model: userland vs. privileged space.
- (Relatively) easy to audit via `syslogd(8)`.
- Aged,
  - Well understood,
  - Reviewed source code,

The Good News

- Simple model: userland vs. privileged space.
- (Relatively) easy to audit via `syslogd(8)`.
- Aged,
  - Well understood,
  - Reviewed source code,
  - Even “proprietary” versions aren’t very proprietary anymore.

The Good News

- Simple model: userland vs. privileged space.
- (Relatively) easy to audit via `syslogd(8)`.
- Aged,
  - Well understood,
  - Reviewed source code,
  - Even “proprietary” versions aren’t very proprietary anymore.
- Modular.

The Good News

- Simple model: userland vs. privileged space.
- (Relatively) easy to audit via `syslogd(8)`.
- Aged,
  - Well understood,
  - Reviewed source code,
  - Even “proprietary” versions aren’t very proprietary anymore.
- Modular.
- Based mostly on a simple language, `C`.

The Bad News

- Architecture is based on files. . .

The Bad News

- Architecture is based on files. . .
  - Permissions can be confusing ,

The Bad News

- Architecture is based on files. . .
  - Permissions can be confusing ,
  - Devices can easily be replaced or backdoored.

The Bad News

- Architecture is based on files. . .
  - Permissions can be confusing ,
  - Devices can easily be replaced or backdoored.
- Gaining access past userland usually leads to root.

The Bad News

- Architecture is based on files. . .
  - Permissions can be confusing ,
  - Devices can easily be replaced or backdoored.
- Gaining access past userland usually leads to root.
- File descriptors, status codes, etc., difficult to securely keep track of.

The Bad News

- Architecture is based on files. . .
  - Permissions can be confusing ,
  - Devices can easily be replaced or backdoored.
- Gaining access past userland usually leads to root.
- File descriptors, status codes, etc., difficult to securely keep track of.
- Long history of buffer overflows, and recently format string vulnerabilities. C isn't forgiving.

Goals

- Two major requirements in dealing host security are. . .

Goals

- Two major requirements in dealing host security are. . .
  - Users.

Goals

- Two major requirements in dealing host security are. . .
  - Users.
    - A good rule of thumb is: “once shell access is obtained, privileged access will soon follow.”

Goals

- Two major requirements in dealing host security are. . .
  - Users.
    - A good rule of thumb is: “once shell access is obtained, privileged access will soon follow.”
    - For this reason, shell access should be protected.

Goals

- Two major requirements in dealing host security are. . .
  - Users.
    - A good rule of thumb is: “once shell access is obtained, privileged access will soon follow.”
    - For this reason, shell access should be protected.
  - The Kernel.

Goals

- Two major requirements in dealing host security are. . .
  - Users.
    - A good rule of thumb is: “once shell access is obtained, privileged access will soon follow.”
    - For this reason, shell access should be protected.
  - The Kernel.
    - The kernel is busy and can be fooled into trusting malicious code.

Goals

- Two major requirements in dealing host security are. . .
  - Users.
    - A good rule of thumb is: “once shell access is obtained, privileged access will soon follow.”
    - For this reason, shell access should be protected.
  - The Kernel.
    - The kernel is busy and can be fooled into trusting malicious code.
    - It isn’t as smart when dealing large amounts of input; it will gladly overwrite memory segments.

Goals

- Two major requirements in dealing host security are. . .
  - Users.
    - A good rule of thumb is: “once shell access is obtained, privileged access will soon follow.”
    - For this reason, shell access should be protected.
  - The Kernel.
    - The kernel is busy and can be fooled into trusting malicious code.
    - It isn’t as smart when dealing large amounts of input; it will gladly overwrite memory segments.
    - The kernel panics and often cannot handle the type of an attacker will provide.

Goals

- Two major requirements in dealing host security are. . .
  - Users.
    - A good rule of thumb is: “once shell access is obtained, privileged access will soon follow.”
    - For this reason, shell access should be protected.
  - The Kernel.
    - The kernel is busy and can be fooled into trusting malicious code.
    - It isn’t as smart when dealing large amounts of input; it will gladly overwrite memory segments.
    - The kernel panics and often cannot handle the type of an attacker will provide.
    - Truly, the kernel does not understand that someone was foolish when coding.

Where Can This Be Applied

- Shell servers.
- FTP servers.
- Everywhere.

Section One

User Accounts and Environments

User Accounts

- Each user should be given a unique account. Use groups and train people how to use `chmod(1)`.
- Set a default umask appropriately. Be proactive and do not rely on users to set this up!
- Don’t be afraid to overly litter `/etc/group`. Remember, you can have (at least) 65535 groups.

Resource Exhaustion

- Building a restricted environment prevents resource exhaustion

```
#include <sys/types.h>
#include <unistd.h>

void main(int argc, char* argv[])
 {
   while(1) fork();
 }
```

- Leads to something you might not like:

```
11:34AM up 326 days, 19:21, 142 users,
load averages:  65.26, 55.20, 49.12
```

Restricted Environments

- Not completely safe, yet effective.

Restricted Environments

- Not completely safe, yet effective.
- Requires least amount of binaries be put in $PATH.

Restricted Environments

- Not completely safe, yet effective.
- Requires least amount of binaries be put in $PATH.
- Can be broken out of if you allow. . .

Restricted Environments

- Not completely safe, yet effective.
- Requires least amount of binaries be put in `$PATH`.
- Can be broken out of if you allow...
  - ...anything that calls `exec(3)`...

```
find / -exec /bin/sh -i \{\}\ ;
```

Restricted Environments

- Not completely safe, yet effective.
- Requires least amount of binaries be put in `$PATH`.
- Can be broken out of if you allow...
  - ...anything that calls `exec(3)`...
    ```
    find / -exec /bin/sh -i \{\}\ ;
    ```
  - ...anything that uses `sigsuspend(2)`...
    ```
    export EDITOR=/usr/bin/vi; pine -z ;^Z
    ```

Restricted Environments

- Not completely safe, yet effective.
- Requires least amount of binaries be put in `$PATH`.
- Can be broken out of if you allow...
  - ...anything that calls `exec(3)`...
    ```
    find / -exec /bin/sh -i \{\}\ ;
    ```
  - ...anything that uses `sigsuspend(2)`...
    ```
    export EDITOR=/usr/bin/vi; pine -z ;^Z
    ```
- Requires many environmental variables to be set (see next slide)

Restricted Environments - Variables

These variables should be set, at minimum, in a restricted environment.

- `DISPLAY ENV HOME TERM TMP TMPDIR USER`.

Restricted Environments - Variables

These variables should be set, at minimum, in a restricted environment.

- `DISPLAY ENV HOME TERM TMP TMPDIR USER`.
- `PATH`.

Restricted Environments - Variables

These variables should be set, at minimum, in a restricted environment.

- `DISPLAY ENV HOME TERM TMP TMPDIR USER`.
- `PATH`.
- `LD_LIBRARY_PATH LD_PRELOAD` (depends on shell).

Restricted Environments - Variables

These variables should be set, at minimum, in a restricted environment.

- `DISPLAY ENV HOME TERM TMP TMPDIR USER`.
- `PATH`.
- `LD_LIBRARY_PATH LD_PRELOAD` (depends on shell).
- `INPUTRC SHELLOPTS` (if using `bash`).

Restricted Environments - Variables

These variables should be set, at minimum, in a restricted environment.

- `DISPLAY ENV HOME TERM TMP TMPDIR USER`.
- `PATH`.
- `LD_LIBRARY_PATH LD_PRELOAD` (depends on shell).
- `INPUTRC SHELLOPTS` (if using `bash`).
- `SHELL VISUAL EDITOR`.

Restricted Environments - Builtins

Shells have “builtins” — many of which should be disabled. Under `bash`, set the following in `/etc/profile`

```
enable -n cd
enable -n declare
enable -n export
enable -n readonly
enable -n set
enable -n unset
enable -n ulimit
enable -n enable
```

Restricted Environments - Don’t Be Fooled

- Don’t be fooled: unless you remove all login profiles in `/etc/profile` before setting the restricting variables, the environment *will* be broken out of
- No guarantee is made that even building the environment will work. Restricted environments are difficult to maintain once you start adding more than a handful of applictations.

Section Two

Files and File Systems

Files and File Systems - SUID/SGID Files

- The more the merrier?
- SUID/SGID file permissions should be removed on anything that isn't critical to a user.
- In a default Redhat 8.0 install you may find. . .
  - Superfluous SUID bits == 20 (!)
  - Superfluous SGID bits == 9
- What's *your* definition of superfluous? :)

Files and File Systems - Problems

- Earlier versions of Solaris forgot to add a sticky bit to `/tmp`.

Files and File Systems - Problems

- Earlier versions of Solaris forgot to add a sticky bit to `/tmp`.
- Many platforms come with `/usr/*bin/*` as user writeable.

Files and File Systems - Problems

- Earlier versions of Solaris forgot to add a sticky bit to `/tmp`.
- Many platforms come with `/usr/*bin/*` as user writeable.
- `/tmp` is extremely useful for exploits that write shells as 4755.

File and File Systems - Beyond chmod

- `chmod(1)` is fine and dandy for file permissions, but cannot apply attributes. However,

File and File Systems - Beyond chmod

- `chmod(1)` is fine and dandy for file permissions, but cannot apply attributes. However,
  - Linux: `chattr -R +u /usr/*bin/* /*bin`

File and File Systems - Beyond chmod

- `chmod(1)` is fine and dandy for file permissions, but cannot apply attributes. However,
  - Linux: `chattr -R +u /usr/*bin/* /*bin`
  - *BSD: `chflags -R schg /usr/*bin/* /*bin /bsd*`

File and File Systems - Mount Options

- Where possible, try and…

File and File Systems - Mount Options

- Where possible, try and. . .
  - ...mount `/usr` “read-only,nodev”

File and File Systems - Mount Options

- Where possible, try and...
  - ...mount `/usr` “read-only,nodev”
  - ...mount `/tmp` and `/var` “noexec,nosuid,nodev”

File and File Systems - Mount Options

- Where possible, try and...
  - ...mount `/usr` “read-only,nodev”
  - ...mount `/tmp` and `/var` “noexec,nosuid,nodev”
  - ...mount `/home` “nosuid,nodev”

Double Standards

"What you've just told me. . . "

- Contradicts the "keep up to date on patches philosophy."
- Doesn't allow easy changes to the system.
- Will cause the much avoided "down-time."

Section Three

The Kernel

Kernel - Dynamic Configuration

- Loadable Kernel Modules (LKM) are popular among all major unices.
  - Popular for quite some time in Linux.
  - Has made it's way (as KLD) to *BSD.
  - Helps to dynamically configure an infinite state machine.
  - Makes end-user life "easier."
- How is the kernel to tell if `foobar.o` is a third-party device driver or part of a rootkit?

Kernel - Risks of LKM's

- An attacker can spoof replies from `fstat(2)` and gladly tell your file integrity checker that `/bsd` wasn't replaced before "yesterday's reboot."

Kernel - Risks of LKM's

- An attacker can spoof replies from `fstat(2)` and gladly tell your file integrity checker that `/bsd` wasn't replaced before "yesterday's reboot."
- The attacker may insert code to selectively copy some, or all, data accepted on an interface to a file, named pipe, etc.

Kernel - Risks of LKM's

- An attacker can spoof replies from `fstat(2)` and gladly tell your file integrity checker that `/bsd` wasn't replaced before "yesterday's reboot."
- The attacker may insert code to selectively copy some, or all, data accepted on an interface to a file, named pipe, etc.
- The attacker can modify the cryptographic framework if loaded as a module. This could be disastrous for <insert large number here> reasons!

Kernel - Risks of LKM's

- An attacker can spoof replies from `fstat(2)` and gladly tell your file integrity checker that `/bsd` wasn't replaced before "yesterday's reboot."
- The attacker may insert code to selectively copy some, or all, data accepted on an interface to a file, named pipe, etc.
- The attacker can modify the cryptographic framework if loaded as a module. This could be disastrous for <insert large number here> reasons!
- A rootkit may be installed on the system with known signatures for security programs, and attacker-defined ways of faking their requests.

Kernel - Risks of LKM's

- An attacker can spoof replies from `fstat(2)` and gladly tell your file integrity checker that `/bsd` wasn't replaced before "yesterday's reboot."
- The attacker may insert code to selectively copy some, or all, data accepted on an interface to a file, named pipe, etc.
- The attacker can modify the cryptographic framework if loaded as a module. This could be disastrous for <insert large number here> reasons!
- A rootkit may be installed on the system with known signatures for security programs, and attacker-defined ways of faking their requests.
- Oh, so many possibilities!

Kernel - Combating LKM Insecurities

- Permit only signed modules to be inserted into the kernel.

Kernel - Combating LKM Insecurities

- Permit only signed modules to be inserted into the kernel.
  - Requires cryptographic framework to part of the core kernel.

Kernel - Combating LKM Insecurities

- Permit only signed modules to be inserted into the kernel.
  - Requires cryptographic framework to part of the core kernel.
  - Where do the signatures come from? Hopefully not from the same host!

Kernel - Combating LKM Insecurities

- Permit only signed modules to be inserted into the kernel.
  - Requires cryptographic framework to part of the core kernel.
  - Where do the signatures come from? Hopefully not from the same host!
  - Will still be vulnerable if the attacker can backdoor the kernel loader.

Kernel - Combating LKM Insecurities

- Permit only signed modules to be inserted into the kernel.
  - Requires cryptographic framework to part of the core kernel.
  - Where do the signatures come from? Hopefully not from the same host!
  - Will still be vulnerable if the attacker can backdoor the kernel loader.
  - Makes for a much larger kernel. Why bother?

Kernel - Combating LKM Insecurities

- Permit only signed modules to be inserted into the kernel.
  - Requires cryptographic framework to part of the core kernel.
  - Where do the signatures come from? Hopefully not from the same host!
  - Will still be vulnerable if the attacker can backdoor the kernel loader.
  - Makes for a much larger kernel. Why bother?
- Linux 2.4 provides `CAP_SYS_MODULE`.

Kernel - Combating LKM Insecurities

- Permit only signed modules to be inserted into the kernel.
  - Requires cryptographic framework to part of the core kernel.
  - Where do the signatures come from? Hopefully not from the same host!
  - Will still be vulnerable if the attacker can backdoor the kernel loader.
  - Makes for a much larger kernel. Why bother?
- Linux 2.4 provides `CAP_SYS_MODULE`.
- StMichael: Monitors `init_module` and `delete_module` under Linux 2.2 and 2.4.

Kernel - Combating LKM Insecurities

- Permit only signed modules to be inserted into the kernel.
  - Requires cryptographic framework to part of the core kernel.
  - Where do the signatures come from? Hopefully not from the same host!
  - Will still be vulnerable if the attacker can backdoor the kernel loader.
  - Makes for a much larger kernel. Why bother?
- Linux 2.4 provides `CAP_SYS_MODULE`.
- StMichael: Monitors `init_module` and `delete_module` under Linux 2.2 and 2.4.
- Do away with LKM's (my vote).

More Security Defenses - Linux

- Openwall patches are useful for older 2.2 kernels.
  - Builds a non-executable user stack area.
  - Restricts games played in `/tmp`.
  - Handles file-descriptors 0, 1 and 2 better.
  - Destroys memory segments not in use.
- Gentoo Linux is now built with gcc 3.2+ProPolice.
- More development is taking place in Linux to increase the core security, but it is slooooooow.

More Security Defenses - OpenBSD

- As of 3.2-stable OpenBSD now has a non-executable stack under 5 architectures (i386, sun4m, sparc64, alpha, macppc).
- Can facilitate `systrace(1)` and execute applications according to defined policies.
- Binaries are built with gcc 3.2+ProPolice.

More Security Defenses - FreeBSD

- FreeBSD can be hardened by implementing the TrustedBSD patches to the core system.
  - Mandatory access control modules and access control lists for files/filesystems permit more granular security policies to be imposed system-wide.
  - Improved privilege structures will assist in reducing the amount of privileges required to run applications.
- While the TrustedBSD work doesn't apply directly to FreeBSD, much of the work is back-ported to FreeBSD.

Conclusion

- Contribute security related code to open-source movements.
- Take base system tools and improve them.
- Write your own advancements for UNIX,
  - Hint: It would be nice to have kernel functions that deny access to any files. . .
  - Hint: Re-engineering isn't necessarily a bad thing. . .

Thank You

- Thanks for your time.
  - For security related help contact <security@infosec.depaul.edu>
  - For incident related help, contact <abuse@depaul.edu>
- Questions of comments are welcome!
  - Contact me at <epancer@infosec.depaul.edu>
  - Visit us on the Web at <http://infosec.depaul.edu>