

---

# Secure Programming II

## (SecProg 2)

Engin Kirda

[kirda@eurecom.fr](mailto:kirda@eurecom.fr)

---

# Administrative Issues

---

- We have close to 20 registrations
  - Challenge 1 will go online today at 15.00, after lecture
  - You need account information to be able to do the challenges. Send me e-mail if you have not done so
  - An e-mail will be sent to course e-mail list whenever a new challenge goes online (with a link)
  - The grading system works just like in SecProg I

# Interesting News From the Field

---

- Mobile TAN apparently not as safe as one would think
  - mTAN is becoming popular (e.g., in Austria)
  - It is a good idea. A second trusted channel for transactions
  - In Australia, bad guys managed to steal information from a user (e.g., mobile phone number, etc.)
  - In a second step, they ported to number to a different provider and stole money from the bank
  - Weakest link in the chain:
    - Identity checking during porting of the number

---

# Unix Security

---

# Unix

---

- Multi-user operating system
- Operating system functionality
  - process management
  - (virtual) memory management
  - file system management
  - I/O management
- Structure
  - operating system kernel
  - user-space programs (daemons, applications, shell)

# Unix

---

- Kernel
  - provides a hardware abstraction layer for user-space programs
  - complete access to all (physical) resources
  - trusted computing base
  - provides services via system calls
- System call
  - performs a transition from user mode to privileged (kernel) mode
  - this crosses the border between two security domains
  - usually implemented with hardware (processor) support
    - processor interrupt
    - x86 call gates

# Unix

---

- Kernel vulnerability
  - usually leads to complete system compromise
  - attacks performed via system calls
  - e.g., a famous one appeared in February, vmsplice
- Solaris / NetBSD call gate creation input validation problem
  - malicious input when creating a LDT (x86 local descriptor table)
  - used in 2001 by Last Stage of Delirium to win Argus Pitbull Competition
- Kernel Integer Overflows
  - FreeBSD `procfs` code (September 2003)
  - Linux `brk()` used to compromise `debian.org` (December 2003)
  - Linux `setsockopt()` (May 2004)
- Linux Memory Management
  - `mremap()` and `munmap()` (March 2004)

# Unix

---

- More Linux vulnerabilities
  - Linux message interface (August 2005, CAN-2005-2490)
  - race condition - `proc` and `prctl` (July 2006, CVE-2006-3626)
  - local privilege escalation - (September 2007, CVE 2007-4573)
- Device driver code is particularly vulnerable
  - (most) drivers run in kernel mode, either kernel modules or compiled-in
  - often not well audited
  - very large code based compared to core services
- Examples
  - `aironet`, `asus_acpi`, `decnet`, `mpu401`, `msnd`, and `pss` (2004)  
found by `sparse` (tool developed by Linus Torvalds)
  - remote root (MadWifi - 2006, Broadcom - 2006)

# Unix

---

- Virtualization
  - has become increasingly popular
  - adds an additional layer under the operating system
  - allows to run multiple guest operating systems in user mode
  - popular solutions are VMware and Xen
  - recently, VM-based rootkits (SubVirt or Blue Pill)
- x86 virtualization problems
  - platform contains “privileged” instructions that do not trap
  - simply different behavior between user and kernel mode
  - interpret each instruction (very slow)
  - restrict / adapt operating system (Xen uses this approach)
  - dynamic recompilation (VMware)

# Unix

---

- Code running in user mode is **always** linked to a certain identity
  - security checks and access control decisions are based on user identity
- Unix is user-centric
  - no roles
- User
  - identified by user name (UID), group name (GID)
  - authenticated by password (stored encrypted)
- User `root`
  - superuser, system administrator
  - special privileges (access resources, modify OS)
  - cannot decrypt user passwords

# Process Management

---

- Process
  - implements user-activity
  - entity that executes a given piece of code
  - has its own execution stack, memory pages, and file descriptors table
  - separated from other processes using the virtual memory abstraction
- Thread
  - separate stack and program counter
  - share memory pages and file descriptor table

# Process Management

---

- Process Attributes
  - process ID (PID)
    - uniquely identified process
  - user ID (UID)
    - ID of owner of process
  - effective user ID (EUID)
    - ID used for permission checks (e.g., to access resources)
  - saved user ID (SUID)
    - to temporarily drop and restore privileges
  - lots of management information
    - scheduling
    - memory management, resource management

# User Authentication

---

- How does a process get a user ID?
  - Authentication (`login`)
- Passwords
  - user passwords are used as keys for `crypt ( )` function
  - runs DES algorithm 25 times on a block of zeros
  - 12-bit “salt”
    - 4096 variations
    - chosen from date, not secret
    - prevent same passwords to map onto same string
    - make dictionary attacks more difficult
- Password cracking
  - dictionary attacks
  - `Crack`, `JohnTheRipper`

# User Authentication

---

- Shadow passwords
  - password file is needed by many applications to map user ID to user names
  - encrypted passwords are not
- `/etc/shadow`
  - holds encrypted passwords
  - account information
    - last change date
    - expiration (warning, disabled)
    - minimum change frequency
  - readable only by superuser and privileged programs
  - MD5 hashed passwords (default) to slow down guessing

# Group Model

---

- Users belong to one or more groups
  - primary group (stored in `/etc/passwd`)
  - additional groups (stored in `/etc/group`)
  - possibility to set group password
  - and become group member with `newgrp`
- `/etc/group`

```
groupname : password : group id : additional users
root:x:0:root
bin:x:1:root,bin,daemon
users:x:100:chris
```
- Special group `wheel`
  - protect `root` account by limiting user accounts that can perform `su`

# File System

---

- File tree
  - primary repository of information
  - hierarchical set of directories
  - directories contain file system objects (FSO)
  - root is denoted “/”
- File system object
  - files, directories, symbolic links, sockets, device files
  - referenced by *inode* (index node)

# File System

---

- Access Control
  - permission bits
  - `chmod`, `chown`, `chgrp`, `umask`
  - file listing:

–            **rwX**        **rwX**        **rwX**  
(file type) (user)    (group) (other)

Type	r	w	x	s	t
<b>File</b>	read access	write access	execute	suid / sgid inherit id	sticky bit
<b>Directory</b>	list files	insert and remove files	stat / execute files, chdir	new files have dir-gid	files only delete- able by owner

# SUID Programs

---

- Each process has *real* and *effective* user / group ID
  - usually identical
  - real IDs
    - determined by current user
    - `login`, `su`
  - effective IDs
    - determine the “rights” of a process
    - system calls (e.g., `setuid()`)
  - `suid` / `sgid` bits
    - to start process with effective ID different from real ID
    - attractive target for attacker
- You cannot use SUID shell scripts anymore

# Shell

---

- Shell
  - one of the core Unix application
  - both a command language and programming language
  - provides an interface to the Unix operating system
  - rich features such as control-flow primitives, parameter passing, variables, and string substitution
  - communication between shell and spawned programs via redirection and pipes
  - different flavors
    - bash and sh, tcsh and csh, ksh

# Shell Attacks

---

- Environment Variables
  - \$HOME and \$PATH can modify behavior of programs that operate with relative path names
  - \$IFS – internal field separator
    - used to parse tokens
    - usually set to [ \t\n] but can be changed to “/”
    - “/bin/ls” is parsed as “bin ls” calling bin locally
    - IFS now only used to split expanded variables
  - `preserve` **attack** (`/usr/lib/preserve` is SUID)
    - called “/bin/mail” when `vi` crashes to preserve file
    - change IFS, create `bin` as link to `/bin/sh`, kill `vi`

# Shell Attacks

---

- Control and escape characters
  - can be injected into command string
  - modify or extend shell behavior
  - user input used for shell commands has to be rigorously sanitized
  - easy to make mistakes
  - classic examples are `;` and `&`
- Applications that are invoked via shell can be targets as well
  - increased vulnerability surface
- Restricted shell
  - invoked with `-r`
  - more controlled environment

# Shell Attacks

---

- `system(char *cmd)`
  - function called by programs to execute other commands
  - invokes shell
  - executes string argument by calling `/bin/sh -c string`
  - makes binary program vulnerable to shell attacks
  - especially when user input is utilized
- `popen(char *cmd, char *type)`
  - forks a process, opens a pipe and invokes shell for `cmd`

# File Descriptor Attacks

---

- SUID program opens file
- forks external process
  - sometimes under user control
- on-execute flag
  - if `close-on-exec` flag is not set, then new process inherits file descriptor
  - malicious attacker might exploit such weakness
- Linux Perl 5.6.0
  - `getpwuid()` leaves `/etc/shadow` opened (June 2002)
  - problem for Apache with `mod_perl`

# Resource Limits

---

- File system limits
  - *quotas (note this for Challenge 1 ;))*
  - restrict number of storage blocks and number of inodes
  - hard limit
    - can never be exceeded (operation fails)
  - soft limit
    - can be exceeded temporarily
  - can be defined per mount-point
  - defend against resource exhaustion (denial of service)
- Process resource limits
  - number of child processes, open file descriptors

# Signals

---

- Signal
  - simple form of interrupt
  - asynchronous notification
  - can happen anywhere for process in user space
  - used to deliver segmentation faults, reload commands, ...
  - `kill` command
- Signal handling
  - process can install signal handlers
  - when no handler is present, default behavior is used
    - ignore or kill process
  - possible to catch all signals except SIGKILL (-9)

# Signals

---

- Security issues
  - code has to be re-entrant
    - atomic modifications
    - no global data structures
  - race conditions
  - unsafe library calls, system calls
  - examples
    - wu-ftpd 2001, sendmail 2001 + 2006, stunnel 2003, ssh 2006
- Secure signals
  - write handler as simple as possible
  - block signals in handler

# Shared Libraries

---

- Library
  - collection of object files
  - included into (linked) program as needed
  - code reuse
- Shared library
  - multiple processes share a **single** library copy
  - save disk space (program size is reduced)
  - save memory space (only a single copy in memory)
  - used by virtually all Unix applications (at least libc.so)
  - check binaries with `ldd`

# Shared Libraries

---

- Static shared library
  - address binding at link-time
  - not very flexible when library changes
  - code is fast
- Dynamic shared library
  - address binding at load-time
  - uses procedure linkage table (PLT) and global offset table (GOT)
  - code is slower (indirection)
  - loading is slow (binding has to be done at run-time)
  - classic `.so` or `.dll` libraries
- PLT and GOT entries are very popular attack targets
  - more when discussing buffer overflows

# Shared Libraries

---

- Management
  - stored in special directories (listed in `/etc/ld.so.conf`)
  - manage cache with `ldconfig`
- Preload
  - override (substitute) with other version
  - use `/etc/ld.so.preload`
  - can also use environment variables for override
  - possible security hazard
  - now disabled for SUID programs (old Solaris vulnerability)

# Unix / Linux Best Practices

---

- If you are maintaining a UNIX-based system...
  - Turn off unused services
    - Services that are not enabled cannot be attacked
    - Services may be vulnerable (e.g., the printer example)
    - You might want to check *inetd* (*/etc/inetd.conf*), */etc/init.d*

```
#pop stream tcp nowait root /etc/uva/tcp_wrapper/tcpd /usr/local/etc/popper popper  
#imap stream tcp nowait root /etc/uva/tcp_wrapper/tcpd /usr/local/etc/imapd4 imapd
```

- If you use xinetd, check */etc/xinetd*

```
service finger{  
    socket_type = stream  
    wait = no  
    user = nobody  
    server = /usr/sbin/in.fingerd  
    disable = yes  
}
```

# Unix / Linux Best Practices

---

- Install IP filter or firewall rules...
  - Even back in 2002, some UNIX systems were open (!)
  - *ipchains* is available with Linux 2.2, as of 2.4, *iptables*
  - AIX and IRIX have similar filtering capabilities
  - In Solaris, IP filtering is not part of the OS until version 8
    - You can buy it ;-)
- Install *tcpd*
  - *tcpd* is a wrapper daemon for tcp-based services
  - With a configuration file, one has finer grained control over accesses

# Unix / Linux Best Practices

---

- Install *sshd*
  - ssh is stable and secure.
    - ssh has a good reputation
    - Nevertheless, there have been problems in the past (so patch your system)
  - Ideally, passwords should not be typed (on keyboard) and remote root access should be disabled
  - ssh in combination with IP-based restrictions and public-key configurations is a good idea
- Try not to use “web application frameworks” and the like
  - Some of them are riddled with holes (e.g., Wordpress)
  - E.g., Dan Kaminsky, Kevin Mitnick, sites, etc.

# Unix / Linux Best Practices

---

- When you leave your desk...
  - You can use a screensaver with password protection
  - Unix systems often allow you to “lock” the screen
  - On MacOS, it might be a good idea to activate (multi-user login)
  - These things might sound trivial, but industrial espionage is an issue and unlocked computers are sometimes used to gain access
    - E.g., disguised cleaning lady
    - Companies often have checks and guidelines for desk management

# One Advantage of Linux Today

---

- No matter how popular it has become, it still has a small number of users (compared to Windows)
  - If you use Linux today, you have a very very small risk of getting infected by a drive-by download
  - Malware for Linux exists, but most attacks are server-side and do not target end-users
  - A Linux VM is an ideal tool for accessing online banking
  - UNIX machines do not run as root by default
  - Using MacOS is less safe (mainly because it is more popular) – MacOS malware has appeared

# The Most Common UNIX Attack

---

- Brute force attacks against services such as ssh, ftp, telnet
  - If you check server logs, frequently, you see repeated attempts for random user names (e.g., admin, root, etc.)
  - These are often bots who try brute force attacks against Internet hosts
  - A simple defense technique: Run your SSH server on a different port (e.g., 800)
    - The downside: Firewalls might be problematic (e.g., Eurecom has this policy)

# Random Number Bug in Debian

---

- On May 13th, 2008 the Debian project announced that Luciano Bello found an interesting bug
  - The random number generation was flawed in *md\_rand.c*
  - The following lines were removed from code:

```
MD_Update(&m,buf,j);  
[ .. ]  
MD_Update(&m,buf,j); /* purify complains */
```
  - These lines caused Valgrind and Purify to complain (i.e., warnings)
  - Removing the code caused the crippling of the seeding process for OpenSSL

# Random Number Bug in Debian

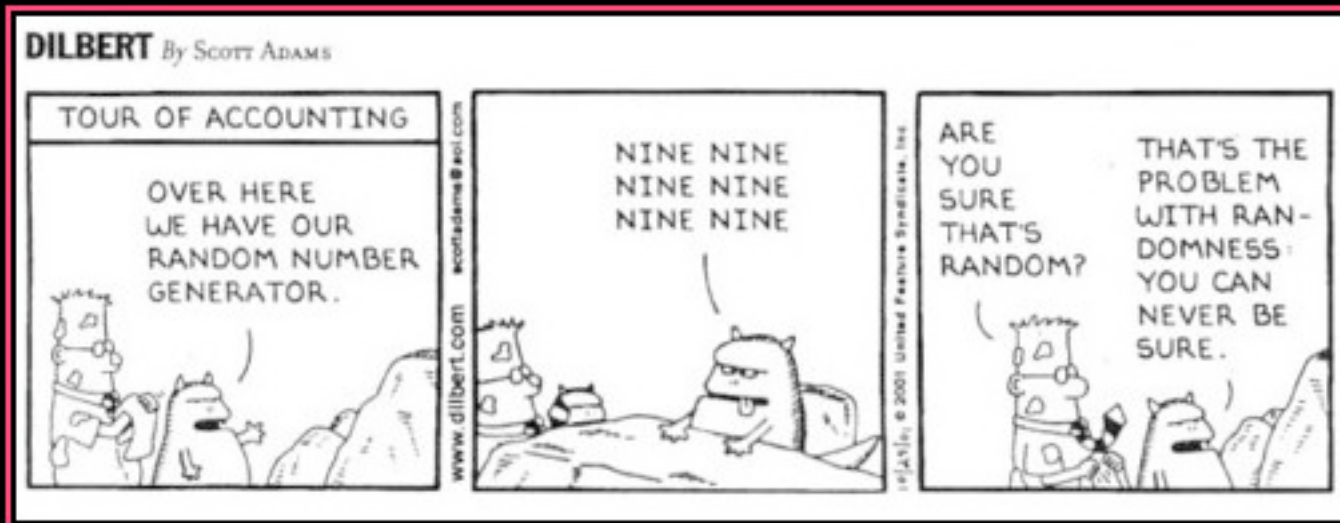
---

- What does this bug mean?
  - It means that public / secret keys generated after the bug was introduced are not as random as one might think
  - In fact, there are lists on the Internet that one can now download
  - With these, it is possible to decode SSL traffic, login to a remote account, etc.
  - <http://www.metasploit.com/users/hdm/tools/debian-openssl/>

# Random Number Bug in Debian

I'LL JUST COMMENT  
ON THESE LINES

IN THE RUSH TO CLEAN



# DEBIAN

YOU CAN NEVER BE SURE.

# Conclusion

---

- Remember: send me e-mail for registration
  - If you have not done so yet...
- We looked at UNIX security today
- Good luck with Challenge 1