ADVANCED STATIC ANALYSIS

MALWARE ANALYSIS AND INCIDENT FORENSICS
M.Sc. in Cyber Security

MALWARE ANALYSIS
M.Sc. in Engineering in Computer Science

A.Y. 2025/2026

SAPIENZA
UNIVERSITÀ DI ROMA

CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

ADVANCED STATIC ANALYSIS

Reverse engineer the sample under analysis

- Use IDA (or an alternative tool) to disassemble the code and inspect it
    - Such tools typically offer advanced functionalities to quickly skim through the code, search for specific snippets, and interpret the code flow
- Identify relevant functions
- Understand the relevant malware behavior
- Extract IoCs

To simplify this part, we will assume that the malware code is not obfuscated or crippled in strange ways

GLOBAL VS. LOCAL VARIABLES

- Global variables can be accessed and used by any function in a program
- Local variables can be accessed only by the function in which they are defined
- Both global and local variables are declared similarly in C, but they look completely different in assembly

GLOBAL VS. LOCAL VARIABLES

```
int x = 1;
int y = 2;

void main()
{
   x = x+y;
   printf("Total = %d\n", x);
}
```

```
00401003        mov     eax, dword_40CF60
00401008        add     eax, dword_40C000
0040100E        mov     dword_40CF60, eax ❶
00401013        mov     ecx, dword_40CF60
00401019        push    ecx
0040101A        push    offset aTotalD  ;"total = %d\n"
0040101F        call    printf
```

GLOBAL VS. LOCAL VARIABLES

```
void main()
{
   int x = 1;
   int y = 2;

   x = x+y;
   printf("Total = %d\n", x);
}
```

```
00401006        mov     dword ptr [ebp-4], 1
0040100D        mov     dword ptr [ebp-8], 2
00401014        mov     eax, [ebp-4]
00401017        add     eax, [ebp-8]
0040101A        mov     [ebp-4], eax
0040101D        mov     ecx, [ebp-4]
00401020        push    ecx
00401021        push    offset aTotalD  ; "total = %d\n"
00401026        call    printf
```

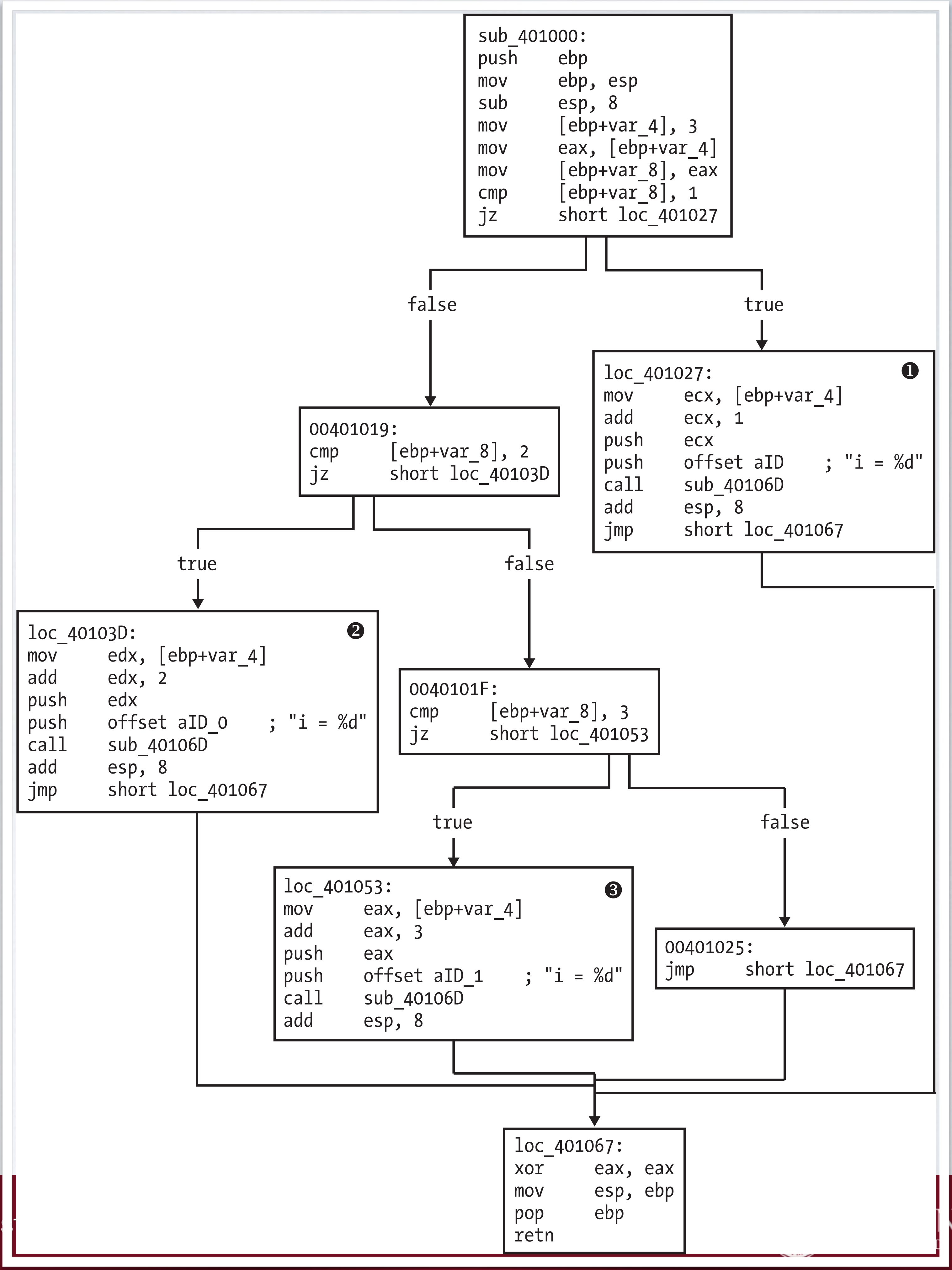
SWITCH

```
switch(i)
{
   case 1:
      printf("i = %d", i+1);
      break;
   case 2:
      printf("i = %d", i+2);
      break;
   case 3:
      printf("i = %d", i+3);
      break;
   default:
      break;
}
```

```
00401013          cmp     [ebp+var_8], 1
00401017          jz      short loc_401027 ❶
00401019          cmp     [ebp+var_8], 2
0040101D          jz      short loc_40103D
0040101F          cmp     [ebp+var_8], 3
00401023          jz      short loc_401053
00401025          jmp     short loc_401067 ❷
00401027 loc_401027:
00401027          mov     ecx, [ebp+var_4] ❸
0040102A          add     ecx, 1
0040102D          push    ecx
0040102E          push    offset unk_40C000 ; i = %d
00401033          call    printf
00401038          add     esp, 8
0040103B          jmp     short loc_401067
0040103D loc_40103D:
0040103D          mov     edx, [ebp+var_4] ❹
00401040          add     edx, 2
00401043          push    edx
00401044          push    offset unk_40C004 ; i = %d
00401049          call    printf
0040104E          add     esp, 8
00401051          jmp     short loc_401067
00401053 loc_401053:
00401053          mov     eax, [ebp+var_4] ❺
00401056          add     eax, 3
00401059          push    eax
0040105A          push    offset unk_40C008 ; i = %d
0040105F          call    printf
00401064          add     esp, 8
```

SWITCH

```
switch(i)
{
    case 1:
        printf("i = %d", i+1);
        break;
    case 2:
        printf("i = %d", i+2);
        break;
    case 3:
        printf("i = %d", i+3);
        break;
    default:
        break;
}
```



```
sub_401000:
push    ebp
mov     ebp, esp
sub     esp, 8
mov     [ebp+var_4], 3
mov     eax, [ebp+var_4]
mov     [ebp+var_8], eax
cmp     [ebp+var_8], 1
jz      short loc_401027
```

```
00401019:
cmp     [ebp+var_8], 2
jz      short loc_40103D
```

```
loc_401027:                          ❶
mov     ecx, [ebp+var_4]
add     ecx, 1
push    ecx
push    offset aID     ; "i = %d"
call    sub_40106D
add     esp, 8
jmp     short loc_401067
```

```
loc_40103D:                          ❷
mov     edx, [ebp+var_4]
add     edx, 2
push    edx
push    offset aID_0   ; "i = %d"
call    sub_40106D
add     esp, 8
jmp     short loc_401067
```

```
0040101F:
cmp     [ebp+var_8], 3
jz      short loc_401053
```

```
loc_401053:                          ❸
mov     eax, [ebp+var_4]
add     eax, 3
push    eax
push    offset aID_1   ; "i = %d"
call    sub_40106D
add     esp, 8
```

```
00401025:
jmp     short loc_401067
```

```
loc_401067:
xor     eax, eax
mov     esp, ebp
pop     ebp
retn
```

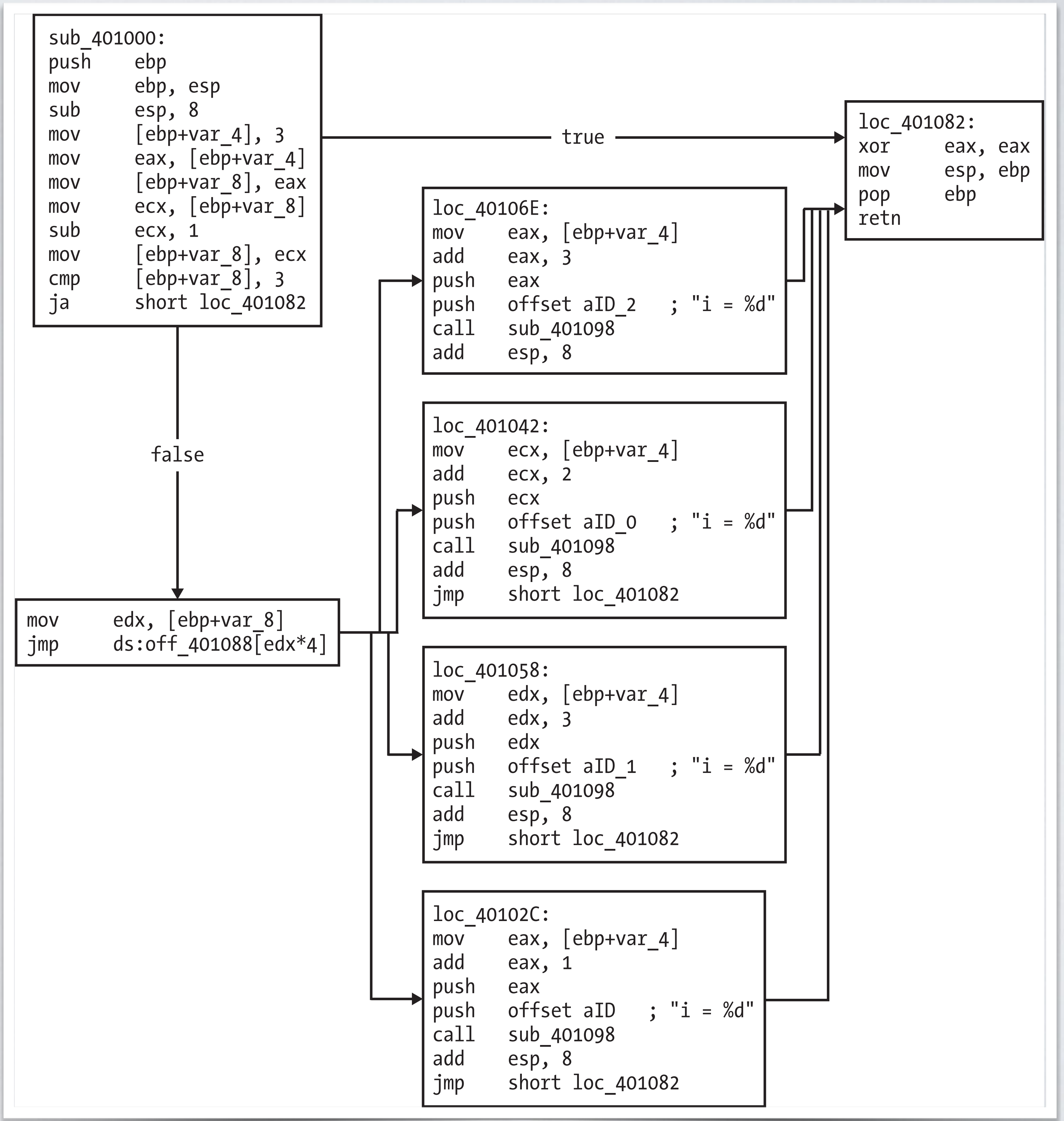
SWITCH

```
switch(i)
{
   case 1:
      printf("i = %d", i+1);
      break;
   case 2:
      printf("i = %d", i+2);
      break;
   case 3:
      printf("i = %d", i+3);
      break;
   case 4:
      printf("i = %d", i+3);
      break;
   default:
      break;
}
```

```
00401016         sub     ecx, 1
00401019         mov     [ebp+var_8], ecx
0040101C         cmp     [ebp+var_8], 3
00401020         ja      short loc_401082
00401022         mov     edx, [ebp+var_8]
00401025         jmp     ds:off_401088[edx*4] ❶
0040102C    loc_40102C:
              ...
00401040         jmp     short loc_401082
00401042    loc_401042:
              ...
00401056         jmp     short loc_401082
00401058    loc_401058:
              ...
0040106C         jmp     short loc_401082
0040106E    loc_40106E:
              ...
00401082    loc_401082:
00401082         xor     eax, eax
00401084         mov     esp, ebp
00401086         pop     ebp
00401087         retn
00401087    _main   endp
00401088   ❷off_401088  dd offset loc_40102C
0040108C                dd offset loc_401042
00401090                dd offset loc_401058
00401094                dd offset loc_40106E
```

SWITCH

```
switch(i)
{
   case 1:
      printf("i = %d", i+1);
      break;
   case 2:
      printf("i = %d", i+2);
      break;
   case 3:
      printf("i = %d", i+3);
      break;
   case 4:
      printf("i = %d", i+3);
      break;
   default:
      break;
}
```



```
sub_401000:
push     ebp
mov      ebp, esp
sub      esp, 8
mov      [ebp+var_4], 3
mov      eax, [ebp+var_4]
mov      [ebp+var_8], eax
mov      ecx, [ebp+var_8]
sub      ecx, 1
mov      [ebp+var_8], ecx
cmp      [ebp+var_8], 3
ja       short loc_401082
```

```
mov      edx, [ebp+var_8]
jmp      ds:off_401088[edx*4]
```

```
loc_40106E:
mov     eax, [ebp+var_4]
add     eax, 3
push    eax
push    offset aID_2    ; "i = %d"
call    sub_401098
add     esp, 8
```

```
loc_401042:
mov     ecx, [ebp+var_4]
add     ecx, 2
push    ecx
push    offset aID_0    ; "i = %d"
call    sub_401098
add     esp, 8
jmp     short loc_401082
```

```
loc_401058:
mov     edx, [ebp+var_4]
add     edx, 3
push    edx
push    offset aID_1    ; "i = %d"
call    sub_401098
add     esp, 8
jmp     short loc_401082
```

```
loc_40102C:
mov     eax, [ebp+var_4]
add     eax, 1
push    eax
push    offset aID    ; "i = %d"
call    sub_401098
add     esp, 8
jmp     short loc_401082
```

```
loc_401082:
xor      eax, eax
mov      esp, ebp
pop      ebp
retn
```

ARRAYS

```
int b[5] = {123,87,487,7,978};
void main()
{
    int i;
    int a[5];

    for(i = 0; i<5; i++)
    {
        a[i] = i;
        b[i] = i;
    }
}
```

```
00401006          mov     [ebp+var_18], 0
0040100D          jmp     short loc_401018
0040100F loc_40100F:
0040100F          mov     eax, [ebp+var_18]
00401012          add     eax, 1
00401015          mov     [ebp+var_18], eax
00401018 loc_401018:
00401018          cmp     [ebp+var_18], 5
0040101C          jge     short loc_401037
0040101E          mov     ecx, [ebp+var_18]
00401021          mov     edx, [ebp+var_18]
00401024          mov     [ebp+ecx*4+var_14], edx ❶
00401028          mov     eax, [ebp+var_18]
0040102B          mov     ecx, [ebp+var_18]
0040102E          mov     dword_40A000[ecx*4], eax ❷
00401035          jmp     short loc_40100F
```

STRUCTS

```
struct my_structure { ❶
     int x[5];
     char y;
     double z;
};

struct my_structure *gms; ❷

void test(struct my_structure *q)
{
     int i;
     q->y = 'a';
     q->z = 15.6;
     for(i = 0; i<5; i++){
           q->x[i] = i;
     }
}

void main()
{
     gms = (struct my_structure *) malloc(
     sizeof(struct my_structure));
     test(gms);
}
```

```
00401050         push    ebp
00401051         mov     ebp, esp
00401053         push    20h
00401055         call    malloc
0040105A         add     esp, 4
0040105D         mov     dword_40EA30, eax
00401062         mov     eax, dword_40EA30
00401067         push    eax ❶
00401068         call    sub_401000
0040106D         add     esp, 4
00401070         xor     eax, eax
00401072         pop     ebp
00401073         retn
```

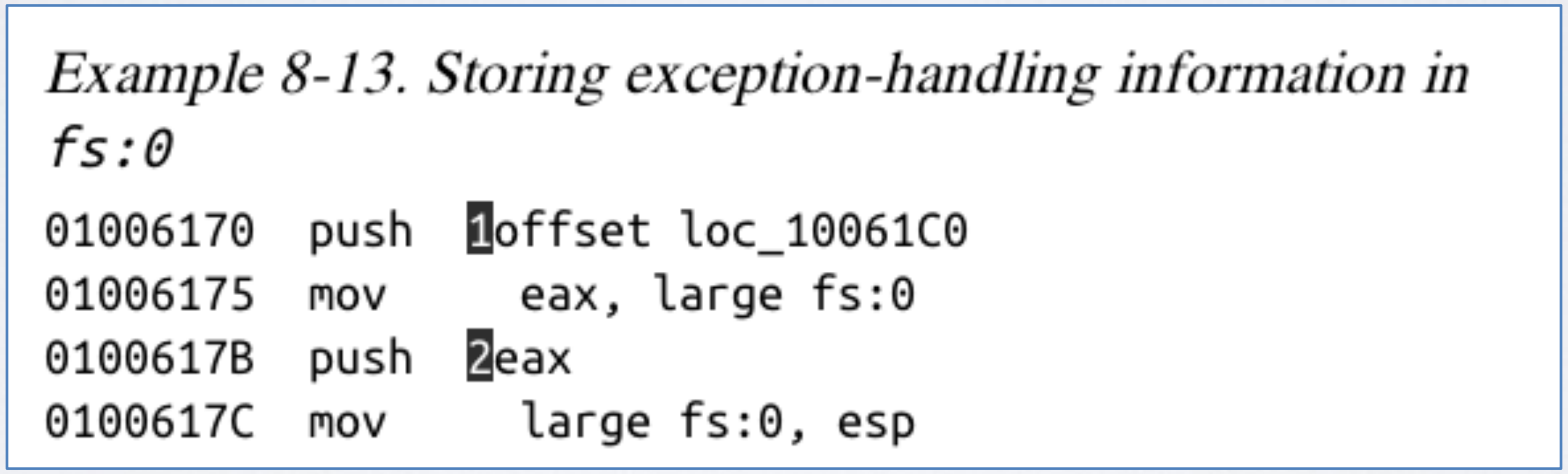
STRUCTS

```
struct my_structure { ❶
     int x[5];
     char y;
     double z;
};

struct my_structure *gms; ❷

void test(struct my_structure *q)
{
     int i;
     q->y = 'a';
     q->z = 15.6;
     for(i = 0; i<5; i++){
           q->x[i] = i;
     }
}

void main()
{
     gms = (struct my_structure *) malloc(
     sizeof(struct my_structure));
     test(gms);
}
```

```
00401000          push    ebp
00401001          mov     ebp, esp
00401003          push    ecx
00401004          mov     eax,[ebp+arg_0]
00401007          mov     byte ptr [eax+14h], 61h
0040100B          mov     ecx, [ebp+arg_0]
0040100E          fld     ds:dbl_40B120 ❶
00401014          fstp    qword ptr [ecx+18h]
00401017          mov     [ebp+var_4], 0
0040101E          jmp     short loc_401029
00401020 loc_401020:
00401020          mov     edx,[ebp+var_4]
00401023          add     edx, 1
00401026          mov     [ebp+var_4], edx
00401029 loc_401029:
00401029          cmp     [ebp+var_4], 5
0040102D          jge     short loc_40103D
0040102F          mov     eax,[ebp+var_4]
00401032          mov     ecx,[ebp+arg_0]
00401035          mov     edx,[ebp+var_4]
00401038          mov     [ecx+eax*4],edx ❷
0040103B          jmp     short loc_401020
0040103D loc_40103D:
0040103D          mov     esp, ebp
0040103F          pop     ebp
00401040          retn
```

EXCEPTIONS

- Exceptions are caused by errors, such as division by zero or invalid memory access
- When an exception occurs, execution transfers to the *Structured Exception Handler*
  - *FS is one of the six Segment Registers*



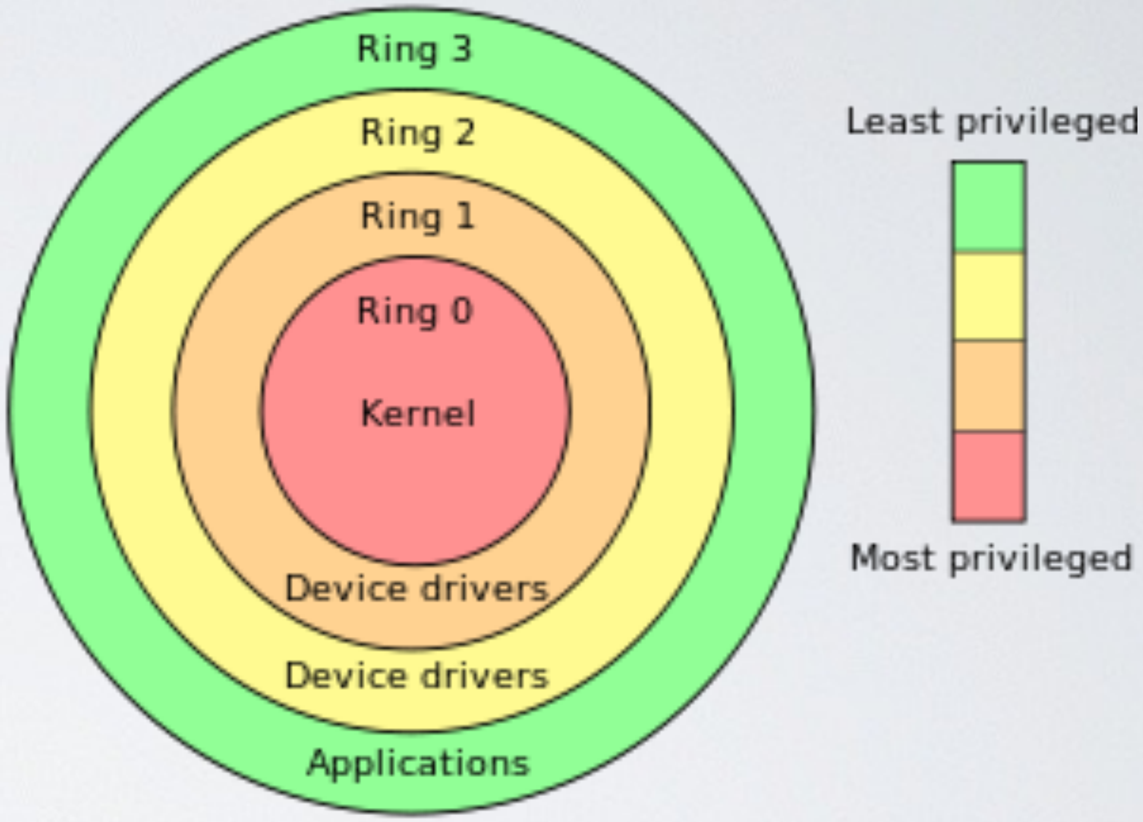
Example 8-13. Storing exception-handling information in fs:0

```
01006170  push  ❶offset loc_10061C0
01006175  mov     eax, large fs:0
0100617B  push  ❷eax
0100617C  mov     large fs:0, esp
```

  - When an exception occurs, Windows looks in fs:0 for the stack location that stores the exception information, and then the exception handler is called.

KERNEL VS USER MODES

- Ring 0: Kernel Mode
- Ring 3: User mode
- Rings 1 and 2 are not used by Windows



USER MODE

- Nearly all code runs in user mode
    - Except OS and hardware drivers, which run in kernel mode
- User mode cannot access hardware directly
- Restricted to a subset of CPU instructions
- Can only manipulate hardware through the Windows API

USER MODE PROCESSES

- Each process has its own memory, security permissions, and resources
- If a user-mode program executes an invalid instruction and crashes, Windows can reclaim the resources and terminate the program

CALLING THE KERNEL

- It is not possible to jump directly from user mode to the kernel
- SYSENTER, SYSCALL, INT 0x2E instructions use lookup tables to locate predefined functions
  - Their presence is an indicator that the code runs functions at the kernel level

KERNEL PROCESSES

- All kernel processes share resources and memory addresses
- Fewer security checks
- If kernel code executes an invalid instruction, the OS crashes with the Blue Screen of Death
- Some security solutions have kernel-mode components
- Kernel-mode malware is more rare, specialized (e.g., rootkits) and sophisticated than user-mode malware

THE NATIVE API

- Lower-level interface for interacting with Windows
  - Ntdll.dll manages interactions between user space and the kernel
  - Ntdll functions make up the Native API
- Rarely used by goodware
- Popular among malware writers as it can be more powerful and stealthier than Windows API calls
- Limited documentation

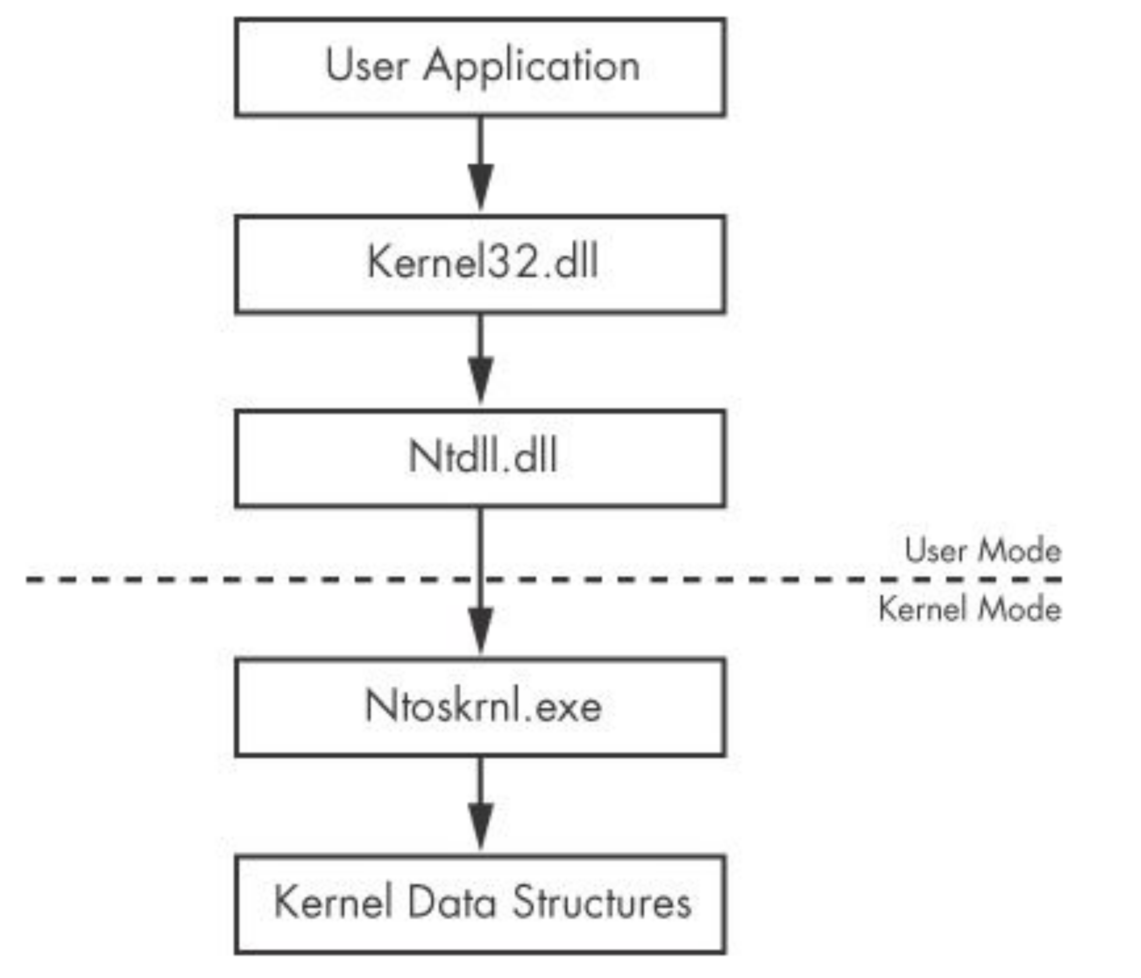


Figure 8-3. User mode and kernel mode

POPULAR NATIVE API CALLS IN MALWARE

- Some Native API calls that can be used to get information about the system, processes, threads, handles, and other items
  - `NTtQuerySystemInformation`
  - `NTtQueryInformationProcess`
  - `NTtQueryInformationThread`
  - `NTtQueryInformationFile`
  - `NTtQueryInformationKey`
- Provide much more information than any available Win32 calls

POPULAR NATIVE API CALLS IN MALWARE

`NtContinue`

- Returns from an exception
- Can be used to transfer execution in complicated ways
- Used to confuse analysts and make a program more difficult to debug

WINDOWS API

Governs how programs interact with Microsoft libraries

Concepts

- Types and Hungarian Notation
- Handles
- File System Functions
- Special Files

TYPES AND HUNGARIAN NOTATION

Windows API has its own names to represent C data types

- Such as DWORD for 32-bit unsigned integers and WORD for 16-bit unsigned integers

Hungarian Notation

- Variables that contain a 32-bit unsigned integer start with the prefix dw

Type (prefix)	
WORD (w)	16-bit unsigned value
DWORD (dw)	32-bit unsigned value
Handle (H)	A reference to an object
Long Pointer (LP)	Points to another type

HANDLES

Items opened or created in the OS, like

- Process, menu, file, window…

Handles are like **immutable** pointers to those OS objects

- You cannot operate on them with arithmetic operations
- You can store it and use it later in the program to refer to the same object
- Sometimes you can check if valid against `INVALID_HANDLE_VALUE`

Example

- The `CreateWindowEx` function returns an `HWND`, a handle to the window
- To do anything to that window (such as `DestroyWindow`), use that handle

FILE SYSTEM FUNCTIONS

`CreateFile`, `ReadFile`, `WriteFile`

- Normal file input/output

`CreateFileMapping`, `MapViewOfFile`

- Frequently used by malware, loads file contents into RAM
  - `CreateFileMapping` loads a file in memory
  - `MapViewOfFile` returns a pointer to the base address of file in memory for access
- Can be used to execute a file without using the Windows loader

SPECIAL FILES

Shared files like `\\server\share`

- Or `\\?\server\share`
  - Disables string parsing, allows longer filenames

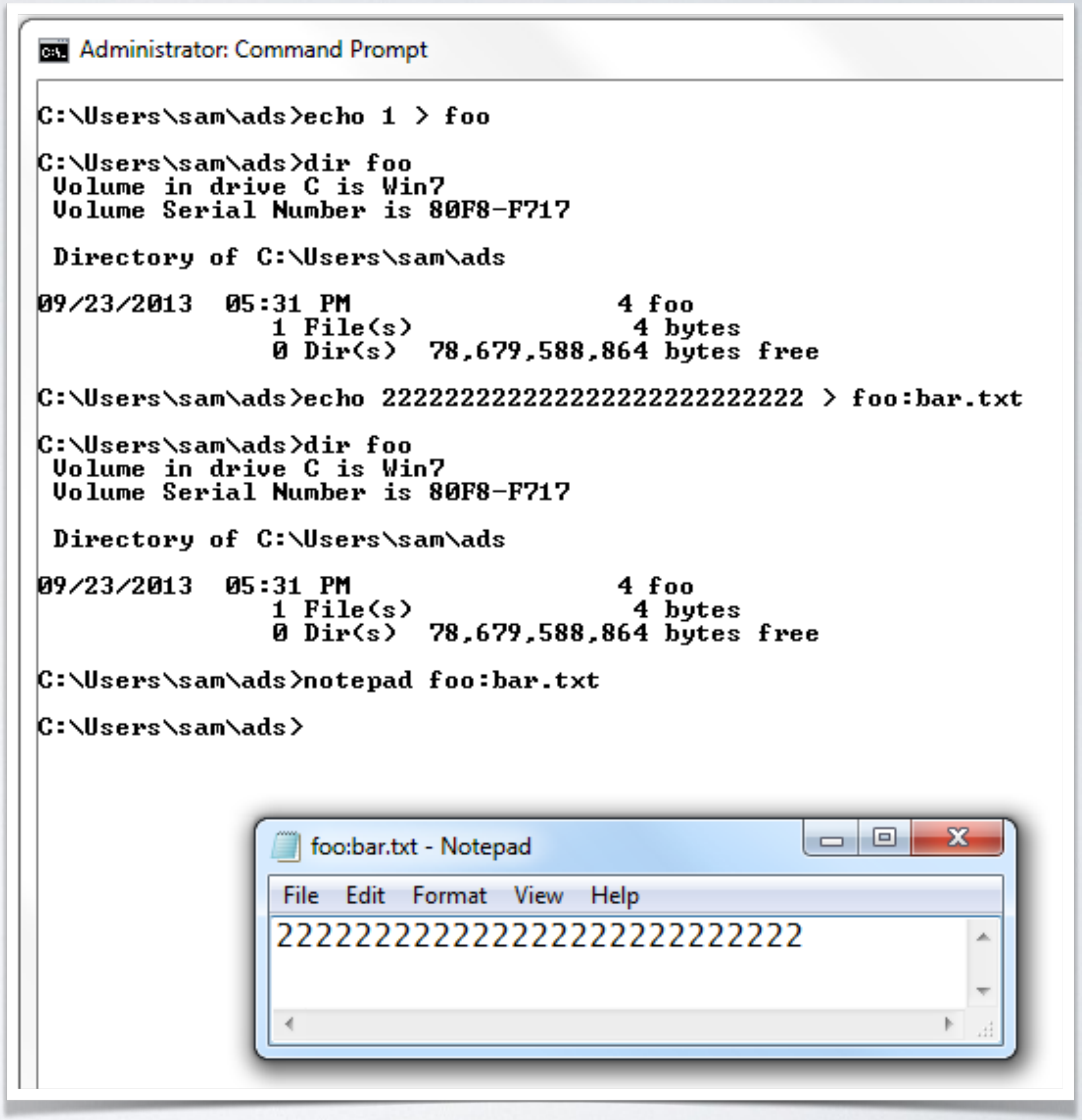
Namespaces

- Special folders in the Windows file system
- `\` Lowest namespace, contains everything
- `\\.\` Device namespace used for direct disk input/output
- Witty worm wrote to `\\.\PhysicalDisk1` to corrupt the disk

SPECIAL FILES

Alternate Data Streams

- Second stream of data attached to a filename
- `File.txt:otherfile.txt`
- Feature of NTFS filesystem



```
C:\Users\sam\ads>echo 1 > foo

C:\Users\sam\ads>dir foo
 Volume in drive C is Win7
 Volume Serial Number is 80F8-F717

 Directory of C:\Users\sam\ads

09/23/2013  05:31 PM                 4 foo
               1 File(s)              4 bytes
               0 Dir(s)  78,679,588,864 bytes free

C:\Users\sam\ads>echo 222222222222222222222222222 > foo:bar.txt

C:\Users\sam\ads>dir foo
 Volume in drive C is Win7
 Volume Serial Number is 80F8-F717

 Directory of C:\Users\sam\ads

09/23/2013  05:31 PM                 4 foo
               1 File(s)              4 bytes
               0 Dir(s)  78,679,588,864 bytes free

C:\Users\sam\ads>notepad foo:bar.txt

C:\Users\sam\ads>
```

```
222222222222222222222222222
```

WINDOWS REGISTRY

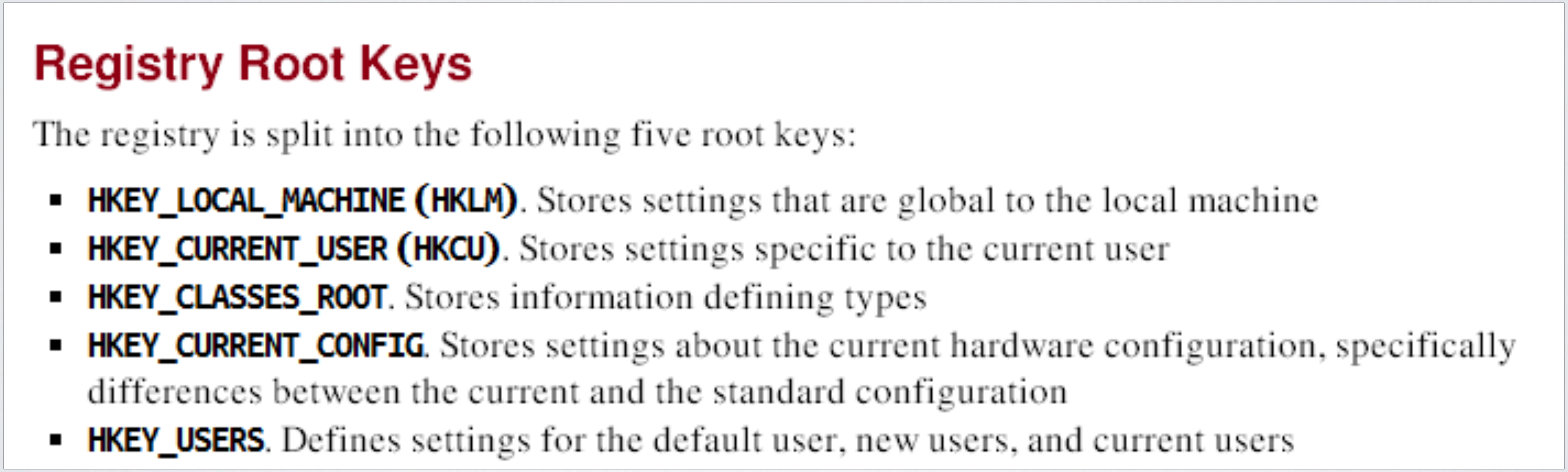
Store operating system and program configuration settings

- Desktop background, mouse preferences, etc.

Malware may use the registry for persistence

- Making malware re-start when the system reboots

5 root keys:



Registry Root Keys

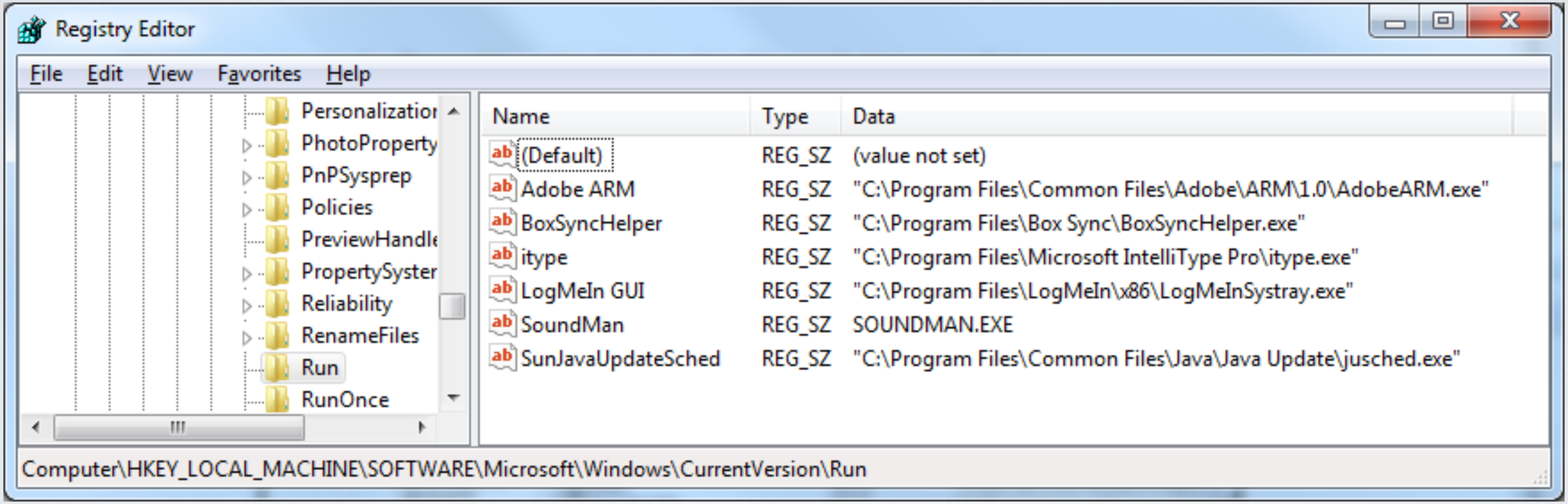
The registry is split into the following five root keys:

- **HKEY_LOCAL_MACHINE (HKLM)**. Stores settings that are global to the local machine
- **HKEY_CURRENT_USER (HKCU)**. Stores settings specific to the current user
- **HKEY_CLASSES_ROOT**. Stores information defining types
- **HKEY_CURRENT_CONFIG**. Stores settings about the current hardware configuration, specifically differences between the current and the standard configuration
- **HKEY_USERS**. Defines settings for the default user, new users, and current users

EXAMPLE

`HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`

- Executables that start when a user logs on



Name	Type	Data
ab (Default)	REG_SZ	(value not set)
ab Adobe ARM	REG_SZ	"C:\Program Files\Common Files\Adobe\ARM\1.0\AdobeARM.exe"
ab BoxSyncHelper	REG_SZ	"C:\Program Files\Box Sync\BoxSyncHelper.exe"
ab itype	REG_SZ	"C:\Program Files\Microsoft IntelliType Pro\itype.exe"
ab LogMeIn GUI	REG_SZ	"C:\Program Files\LogMeIn\x86\LogMeInSystray.exe"
ab SoundMan	REG_SZ	SOUNDMAN.EXE
ab SunJavaUpdateSched	REG_SZ	"C:\Program Files\Common Files\Java\Java Update\jusched.exe"

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

COMMON REGISTRY FUNCTIONS

`RegOpenKeyEx`

- Opens a registry key for editing and querying

`RegSetValueEx`

- Adds a new value to the registry & sets its data

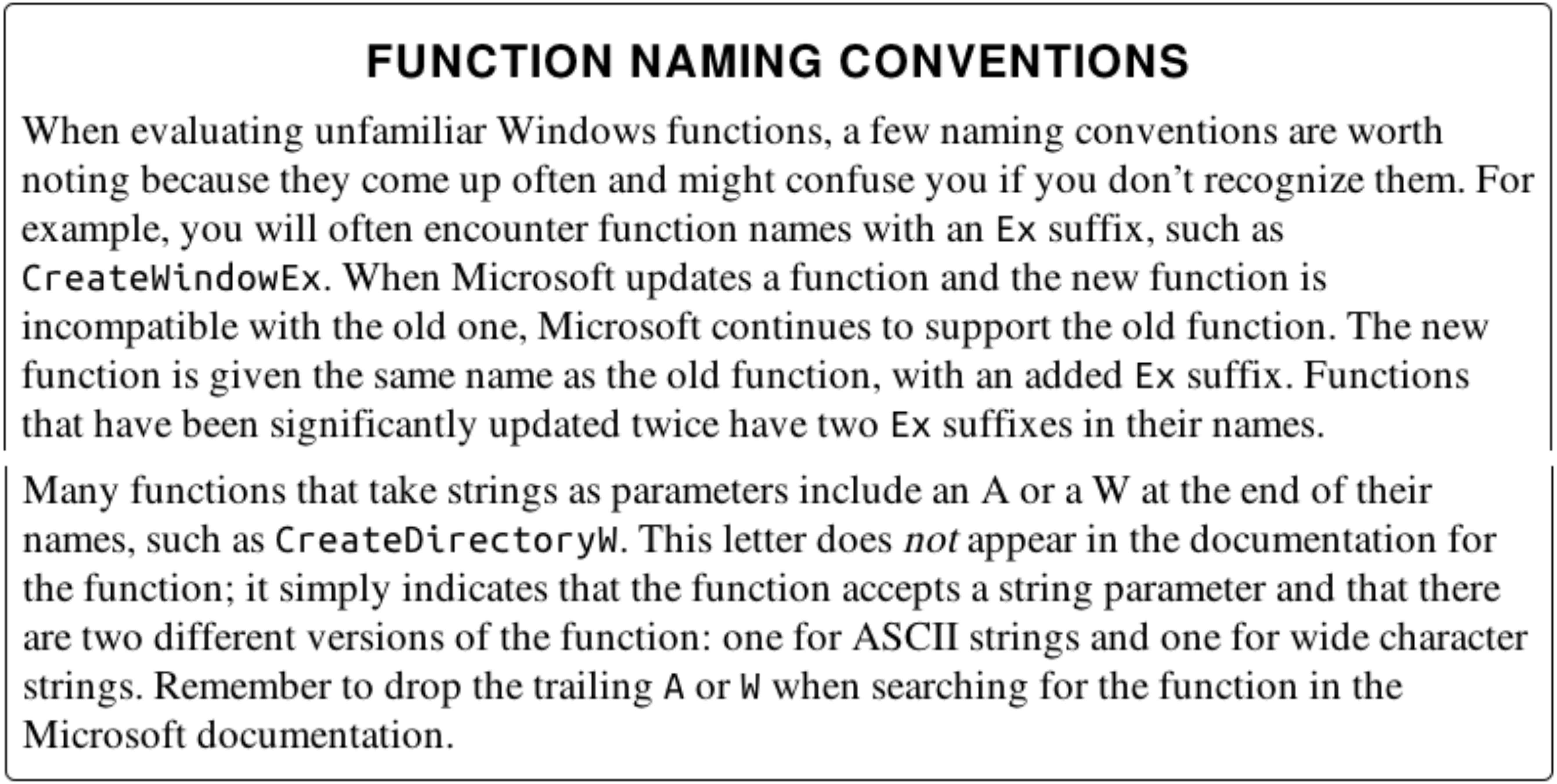
`RegGetValue`

- Returns the data for a value entry in the Registry

Documentation will omit the trailing W (wide) or A (ASCII) character in a call like `RegOpenKeyExW`

EX, A, AND W SUFFIXES

- From book’s Chapter 2



FUNCTION NAMING CONVENTIONS

When evaluating unfamiliar Windows functions, a few naming conventions are worth noting because they come up often and might confuse you if you don’t recognize them. For example, you will often encounter function names with an `Ex` suffix, such as `CreateWindowEx`. When Microsoft updates a function and the new function is incompatible with the old one, Microsoft continues to support the old function. The new function is given the same name as the old function, with an added `Ex` suffix. Functions that have been significantly updated twice have two `Ex` suffixes in their names.

Many functions that take strings as parameters include an A or a W at the end of their names, such as `CreateDirectoryW`. This letter does *not* appear in the documentation for the function; it simply indicates that the function accepts a string parameter and that there are two different versions of the function: one for ASCII strings and one for wide character strings. Remember to drop the trailing `A` or `W` when searching for the function in the Microsoft documentation.

REGISTRY CODE

```
0040286F    push    2                   ; samDesired
00402871    push    eax                 ; ulOptions
00402872    push    offset SubKey       ; "Software\\Microsoft\\Windows\\CurrentVersion\\Run"
00402877    push    HKEY_LOCAL_MACHINE ; hKey
0040287C  ❶call    esi ; RegOpenKeyExW
0040287E    test    eax, eax
00402880    jnz     short loc_4028C5
00402882
00402882 loc_402882:
00402882    lea     ecx, [esp+424h+Data]
00402886    push    ecx                 ; lpString
00402887    mov     bl, 1
00402889  ❷call    ds:lstrlenW
0040288F    lea     edx, [eax+eax+2]
00402893  ❸push    edx                 ; cbData
00402894    mov     edx, [esp+428h+hKey]
00402898  ❹lea     eax, [esp+428h+Data]
0040289C    push    eax                 ; lpData
0040289D    push    1                   ; dwType
0040289F    push    0                   ; Reserved
004028A1  ❺lea     ecx, [esp+434h+ValueName]
004028A8    push    ecx                 ; lpValueName
004028A9    push    edx                 ; hKey
004028AA    call    ds:RegSetValueExW
```

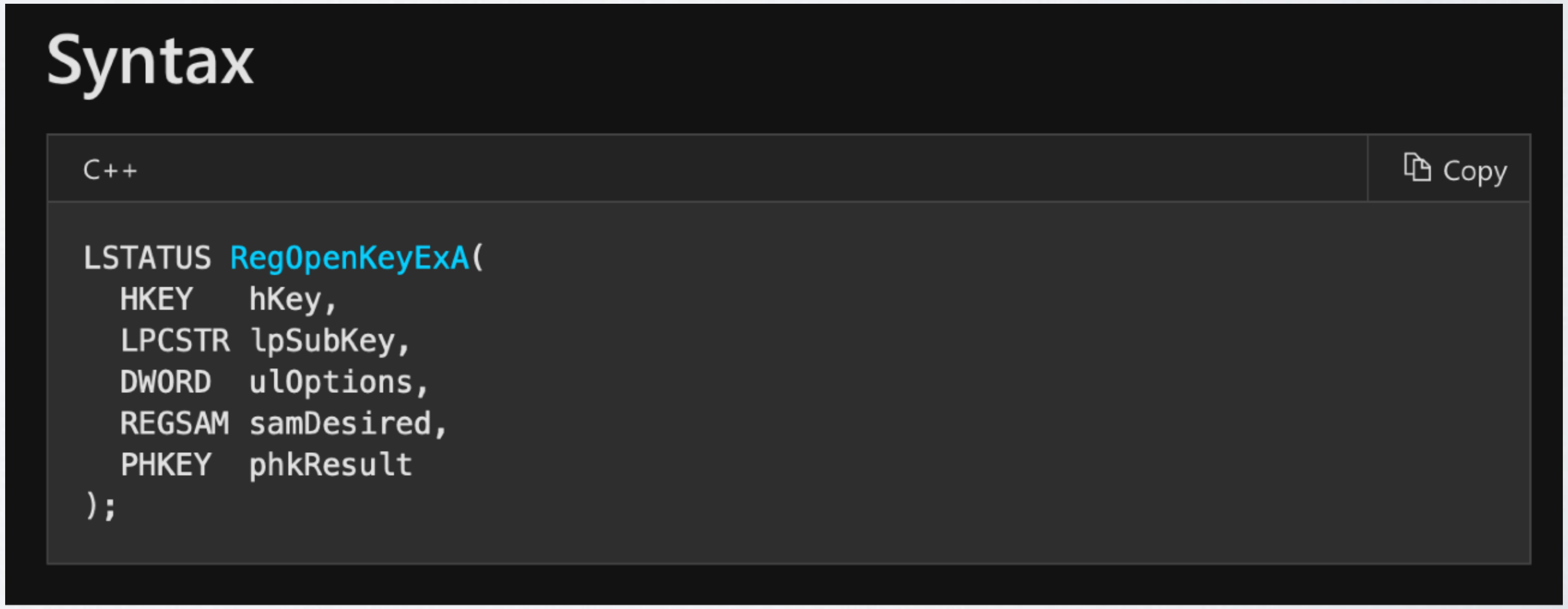
REGISTRY CODE

5 parameters for the call

Location for storing the output handle

Access level

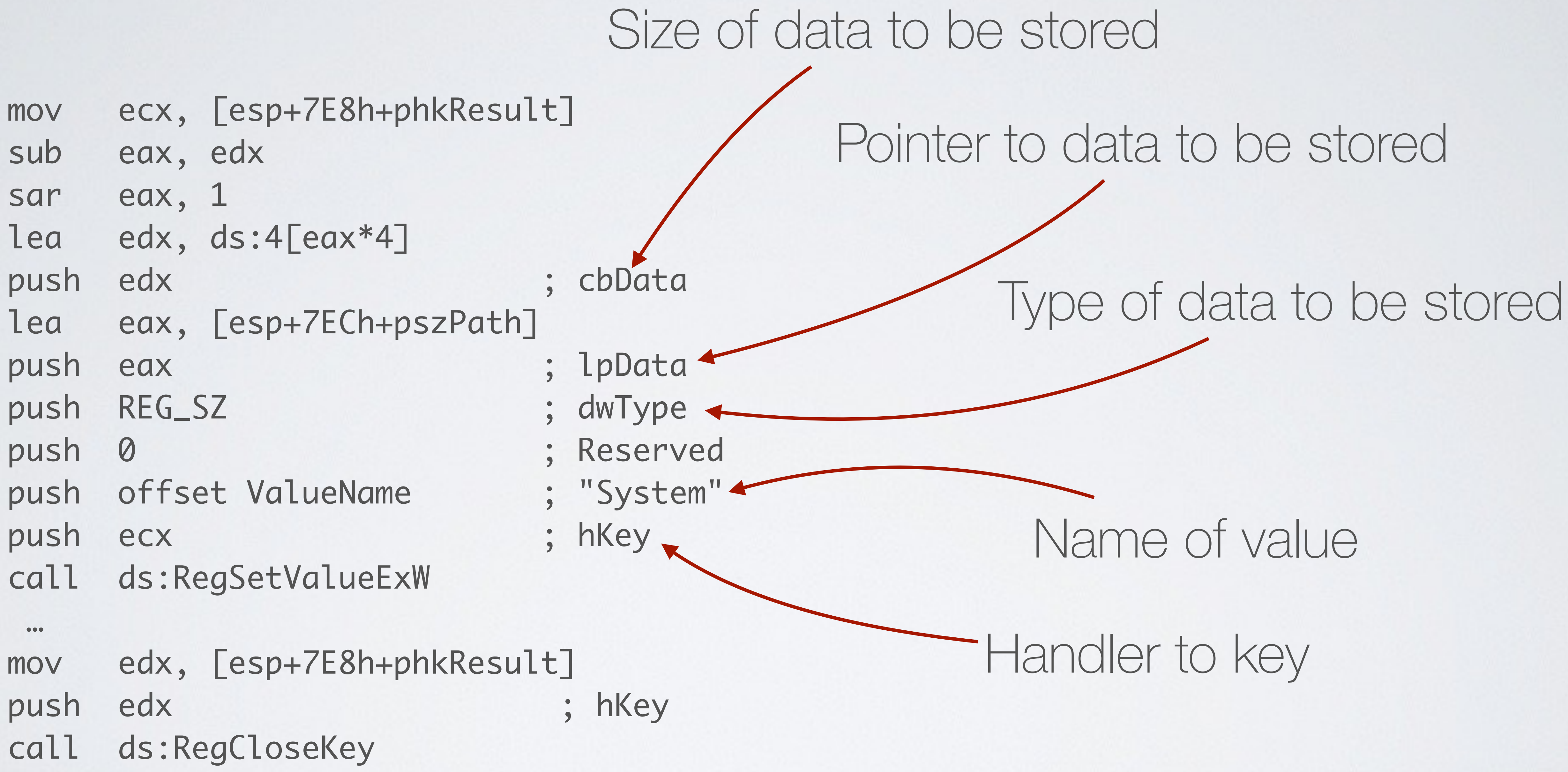
```
lea  ecx, [esp+7E8h+phkResult]
push ecx                            ; phkResult
push 20006h                         ; samDesired KEY_WRITE
push 0                              ; ulOptions
push offset aSoftwareMicros         ; Software\Microsoft\Windows\CurrentVersion\Run
push HKEY_CURRENT_USER              ; hKey
call ds:RegOpenKeyExW
```



Syntax

```
LSTATUS RegOpenKeyExA(
  HKEY   hKey,
  LPCSTR lpSubKey,
  DWORD  ulOptions,
  REGSAM samDesired,
  PHKEY  phkResult
);
```

REGISTRY CODE



```
mov   ecx, [esp+7E8h+phkResult]
sub   eax, edx
sar   eax, 1
lea   edx, ds:4[eax*4]
push  edx                       ; cbData
lea   eax, [esp+7ECh+pszPath]
push  eax                       ; lpData
push  REG_SZ                    ; dwType
push  0                         ; Reserved
push  offset ValueName          ; "System"
push  ecx                       ; hKey
call  ds:RegSetValueExW
 …
mov   edx, [esp+7E8h+phkResult]
push  edx                        ; hKey
call  ds:RegCloseKey
```

Size of data to be stored

Pointer to data to be stored

Type of data to be stored

Name of value

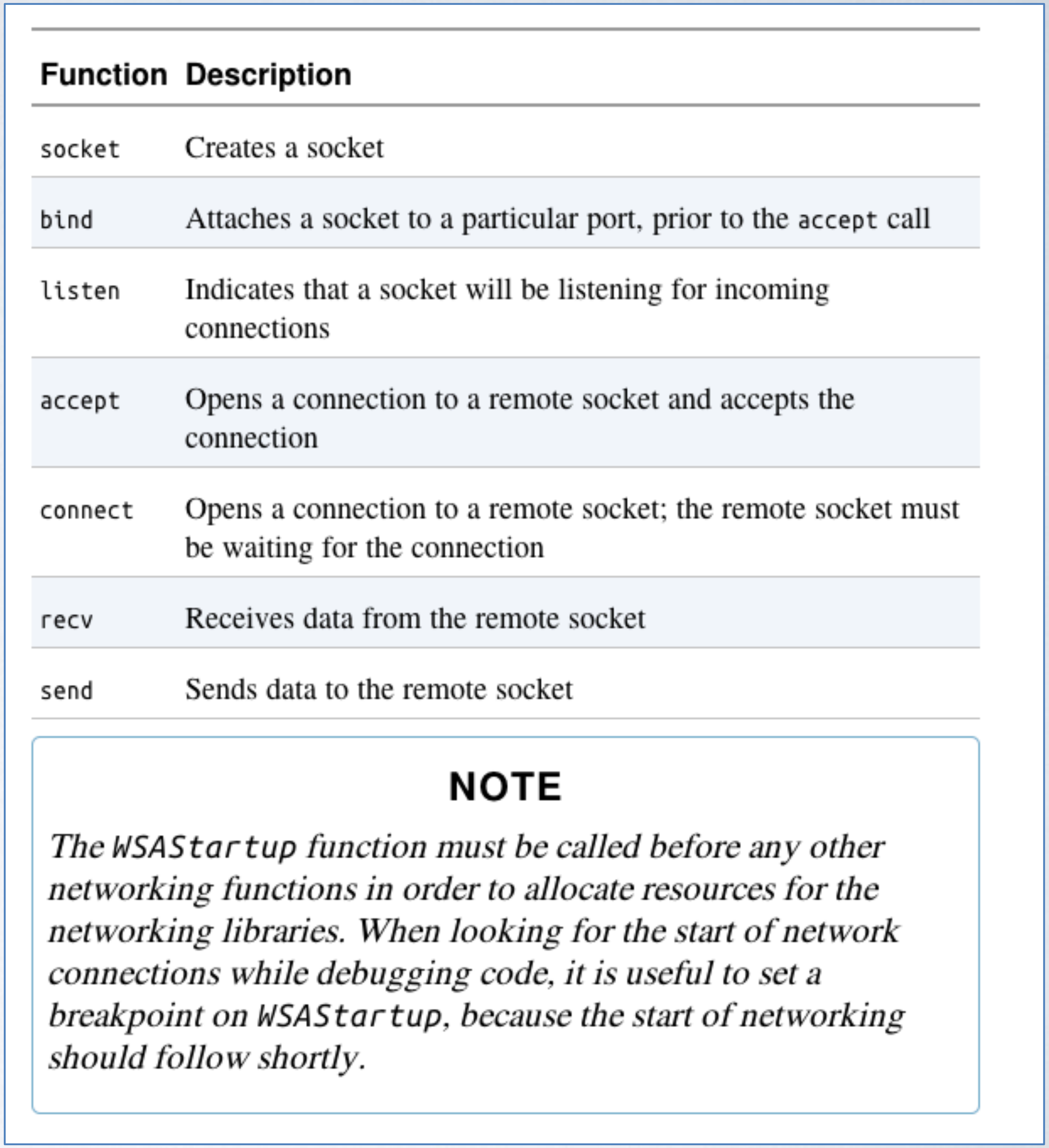
Handler to key

NETWORK API

Berkeley Compatible Sockets

Winsock libraries, primarily in `ws2_32.dll`

- Almost identical in Windows and Unix



Function	Description
`socket`	Creates a socket
`bind`	Attaches a socket to a particular port, prior to the `accept` call
`listen`	Indicates that a socket will be listening for incoming connections
`accept`	Opens a connection to a remote socket and accepts the connection
`connect`	Opens a connection to a remote socket; the remote socket must be waiting for the connection
`recv`	Receives data from the remote socket
`send`	Sends data to the remote socket

NOTE

The `WSAStartup` function must be called before any other networking functions in order to allocate resources for the networking libraries. When looking for the start of network connections while debugging code, it is useful to set a breakpoint on `WSAStartup`, because the start of networking should follow shortly.

SERVER AND CLIENT SIDES

Server side

- Maintains an open socket waiting for connections
- Calls, in order, `socket`, `bind`, `listen`, `accept`
- Then `send` and `recv` as necessary

Client side

- Connects to a waiting socket
- Calls, in order, `socket`, `connect`
- Then `send` and `recv` as necessary

THE WININET API

- Higher-level API than Winsock
- Functions in `Wininet.dll`
- Implements Application-layer protocols like HTTP and FTP
- `InternetOpen` – initializes use of api
- `InternetOpenURL` –connects to a URL
- `InternetReadFile` –reads data from a URL

TRANSFERRING EXECUTION

`jmp` and call `transfer` execution to another part of code, but there are other ways:

- DLLs
- Processes
- Threads
- Mutexes
- Services
- Component Object Model (COM)
- Exceptions

DLL (DYNAMIC LINK LIBRARIES)

Share code among multiple applications

DLLs export code that can be used by other applications

Static libraries were used before DLLs. Still exist, but more rare

Using DLLs already included in Windows makes code smaller

Software companies can also make custom DLLs

- Distribute DLLs along with EXEs

HOW MALWARE AUTHORS USE DLLS

Store malicious code in DLL

- Sometimes load malicious DLL into another process

Using Windows DLLs

- Nearly all malware uses basic Windows DLLs

Using third-party DLLs

- Use Firefox DLL to connect to a server, instead of Windows API

BASIC DLL STRUCTURE

- DLLs are very similar to EXEs
- PE file format
- A single flag indicates that it is a DLL instead of an EXE
- DLLs have more exports & fewer imports
- `DllMain` is the main function, not exported, but specified as the entry point in the PE Header
  - Called when a function loads or unloads the library

PROCESSES

- Every program being executed by Windows is a *process*
- Each process has its own resources
  - Handles, memory
- Each process has one or more *threads*
- Older malware ran as an independent process
- Newer malware executes its code as part of another process

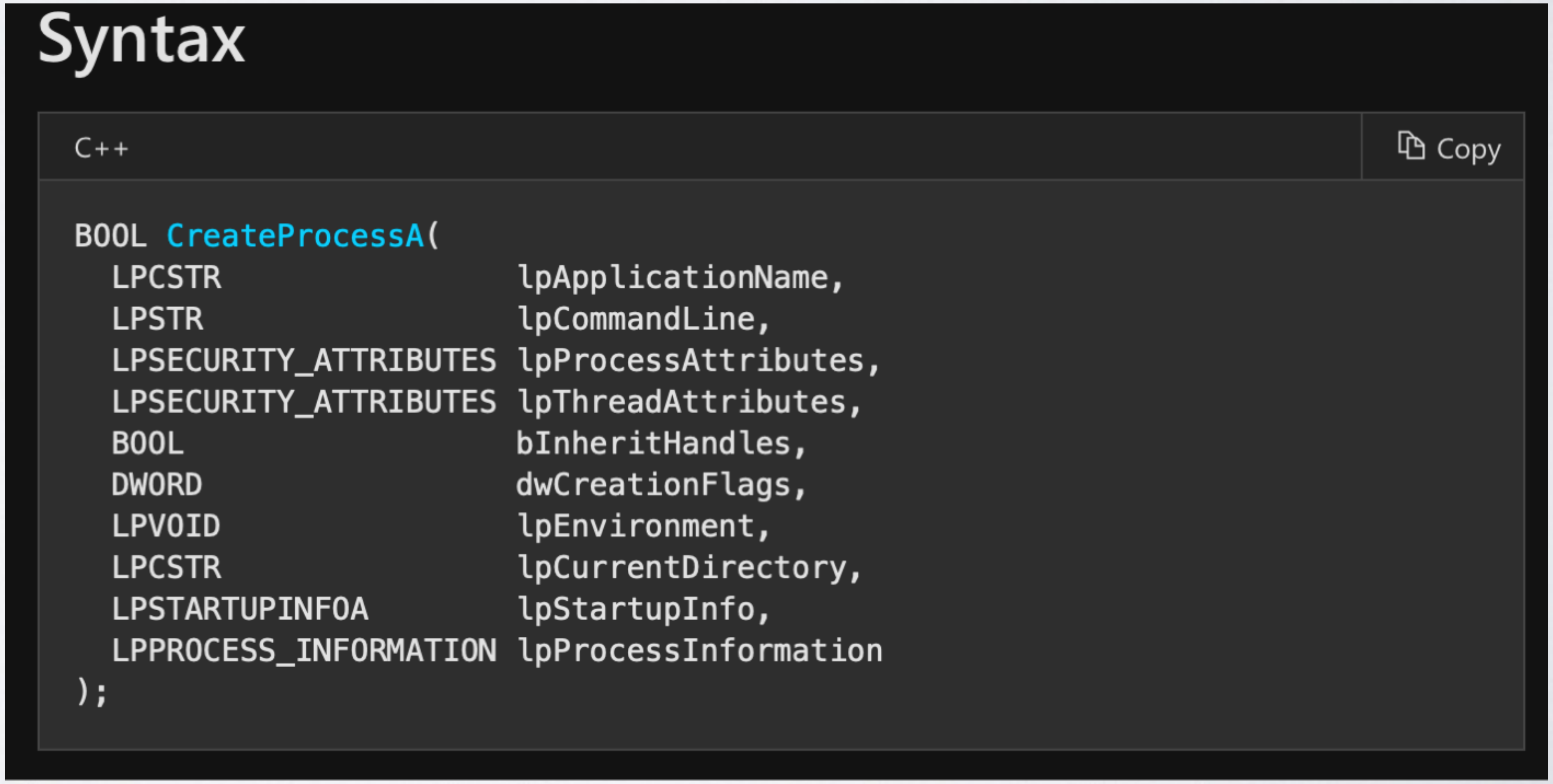
MEMORY MANAGEMENT

- Each process uses resources, like CPU, file system, and memory
- OS allocates memory to each process
- Two processes accessing the same memory address actually access different locations in RAM
  - Virtual address space

CREATING A NEW PROCESS

CreateProcess

- Can create a simple remote shell with one function call
- STARTUPINFO parameter contains handles for standard input, standard output, and standard error streams
- Can be set to a socket, creating a remote shell



Syntax

```cpp
BOOL CreateProcessA(
  LPCSTR                lpApplicationName,
  LPSTR                 lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL                  bInheritHandles,
  DWORD                 dwCreationFlags,
  LPVOID                lpEnvironment,
  LPCSTR                lpCurrentDirectory,
  LPSTARTUPINFOA        lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation
);
```

THREADS

Processes are containers

- Each process contains one or more threads

Threads are what Windows actually executes

- Independent sequences of instructions
- Executed by CPU without waiting for other threads
- Threads within a process share the same memory space
- Each thread has its own registers and stack

THREAD CONTEXT

- When a thread is running, it has complete control of the CPU
- Other threads cannot affect the state of the CPU
- When a thread changes a register, it does not affect any other threads
- When the OS switches to another thread, it saves all CPU values in a structure called the *thread context*

CREATING A THREAD

`CreateThread`

- Caller specified a `start` address, also called a `start` function

How malware coders can use threads

- Manipulate other running processes (later in the course…)
- Create two threads, for input and output
  - Used to communicate with a running application

COORDINATION WITH MUTEXES

- Mutexes are global objects for inter-process communication
- They can help coordinate multiple processes and threads
  - In the kernel, they are called mutants
- Mutexes often use hard-coded names which can be used to identify malware
  - Good source of IoCs

FUNCTIONS FOR MUTEXES

`WaitForSingleObject`

- Gives a thread access to the mutex
- Any subsequent threads attempting to gain access to it must wait

`ReleaseMutex`

- Called when a thread is done using the mutex

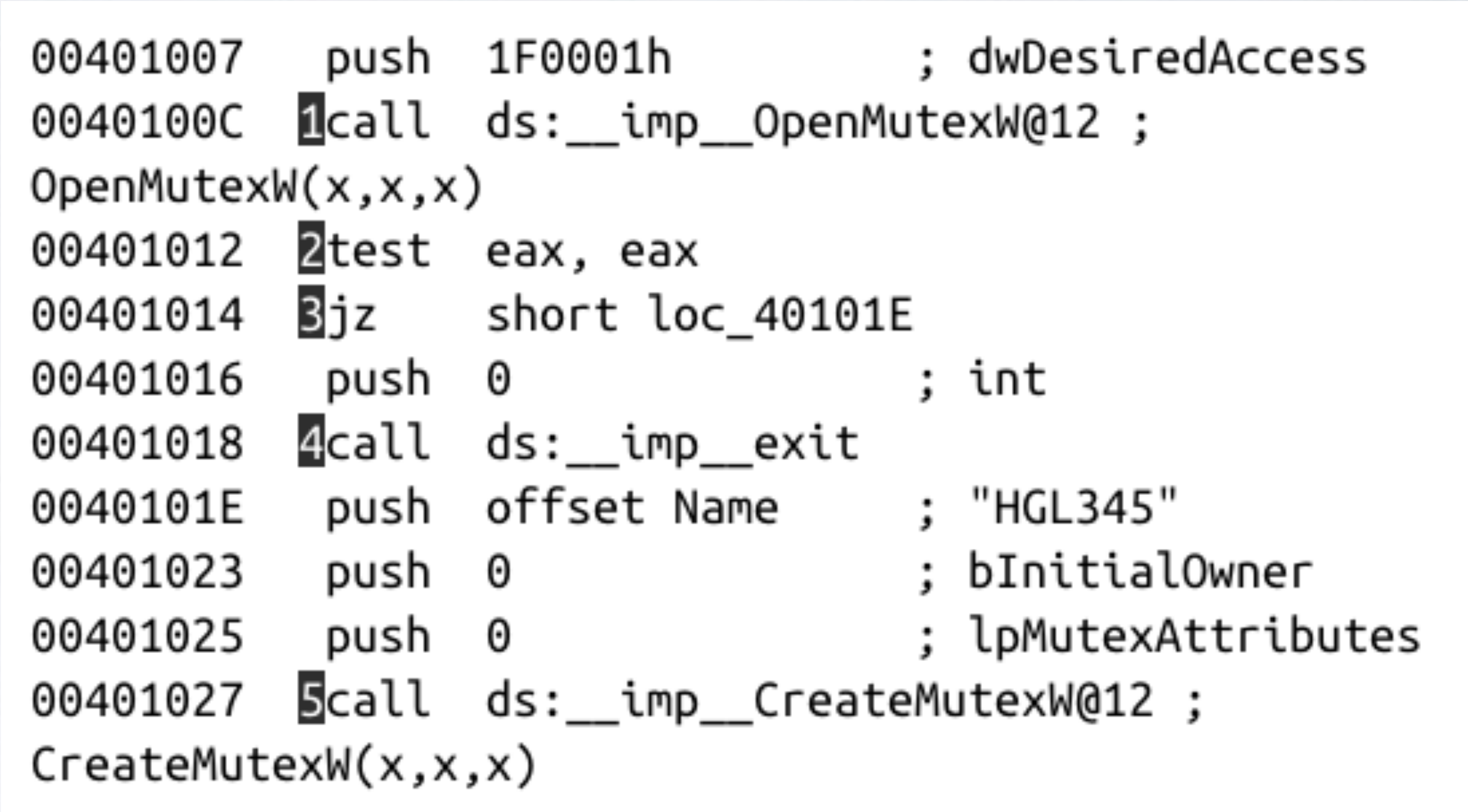
`CreateMutex`

`OpenMutex`

- Gets a handle to another process's mutex

CHECK ONLY ONE COPY OF MALWARE IS RUNNING

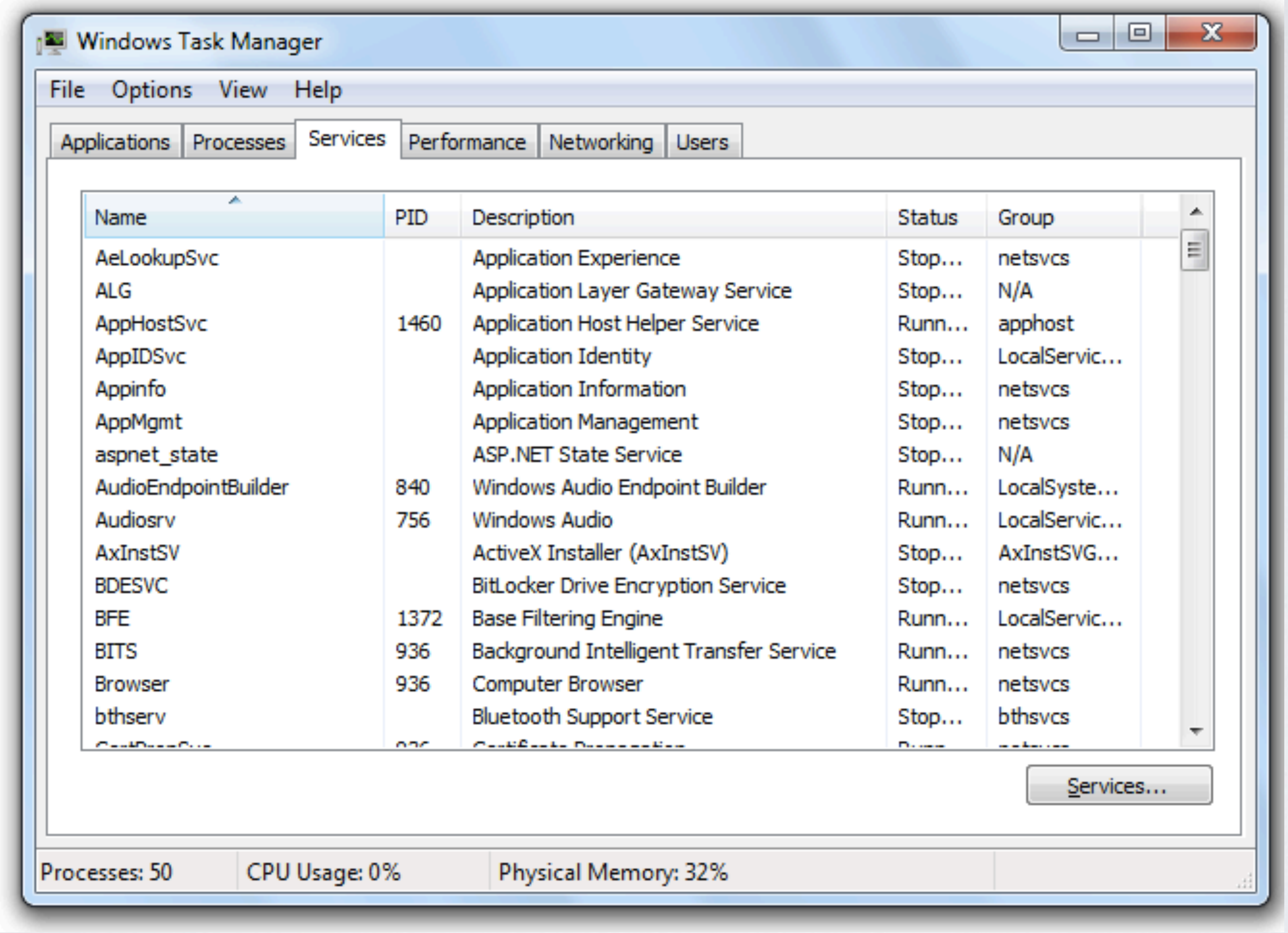
- `OpenMutex` checks if HGL345 exists
- If not, it is created with `CreateMutex`
- `test eax, eax`
  sets `z` flag if `eax` is zero



```
00401007    push  1F0001h         ; dwDesiredAccess
0040100C   1call  ds:__imp__OpenMutexW@12 ;
OpenMutexW(x,x,x)
00401012   2test  eax, eax
00401014   3jz    short loc_40101E
00401016    push  0               ; int
00401018   4call  ds:__imp__exit
0040101E    push  offset Name     ; "HGL345"
00401023    push  0               ; bInitialOwner
00401025    push  0               ; lpMutexAttributes
00401027   5call  ds:__imp__CreateMutexW@12 ;
CreateMutexW(x,x,x)
```

SERVICES

- Services run in the background without user input



Name	PID	Description	Status	Group
AeLookupSvc		Application Experience	Stop...	netsvcs
ALG		Application Layer Gateway Service	Stop...	N/A
AppHostSvc	1460	Application Host Helper Service	Runn...	apphost
AppIDSvc		Application Identity	Stop...	LocalServic...
Appinfo		Application Information	Stop...	netsvcs
AppMgmt		Application Management	Stop...	netsvcs
aspnet_state		ASP.NET State Service	Stop...	N/A
AudioEndpointBuilder	840	Windows Audio Endpoint Builder	Runn...	LocalSyste...
Audiosrv	756	Windows Audio	Runn...	LocalServic...
AxInstSV		ActiveX Installer (AxInstSV)	Stop...	AxInstSVG...
BDESVC		BitLocker Drive Encryption Service	Stop...	netsvcs
BFE	1372	Base Filtering Engine	Runn...	LocalServic...
BITS	936	Background Intelligent Transfer Service	Runn...	netsvcs
Browser	936	Computer Browser	Runn...	netsvcs
bthserv		Bluetooth Support Service	Stop...	bthsvcs

Processes: 50 CPU Usage: 0% Physical Memory: 32%

SYSTEM ACCOUNT

- Services often run as SYSTEM which is even more powerful than the Administrator
- Services can run automatically when Windows starts
    - An easy way for malware to maintain persistence
    - Persistent malware survives a restart

SERVICE API FUNCTIONS

`OpenSCManager`

- Returns a handle to the Service Control Manager

`CreateService`

- Adds a new service to the Service Control Manager
- Can specify whether the service will start automatically at boot time

`StartService`

- Only used if the service is set to start manually

SVCHOST.EXE

`WIN32_SHARE_PROCESS`

- Most common type of service used by malware
- Stores code for service in a DLL
- Combines several services into a single shared process named **svchost.exe**

OTHER COMMON SERVICE TYPES

`WIN32_OWN_PROCESS`

- Runs as an EXE in an independent process

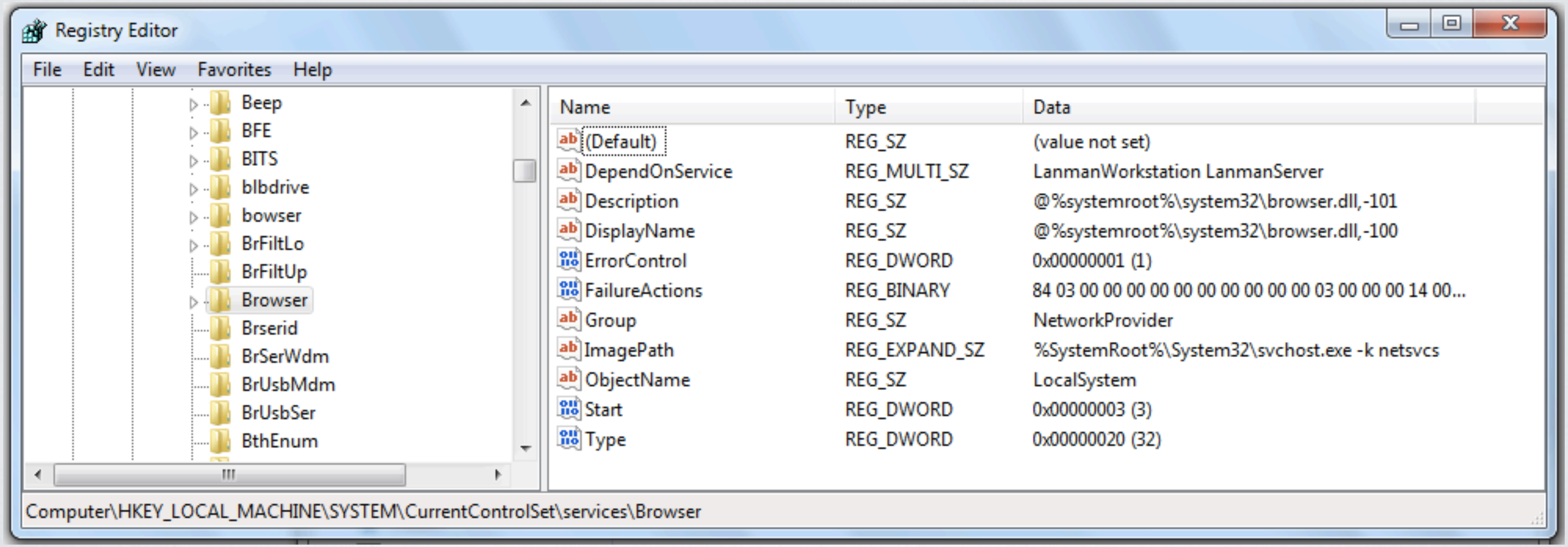
`KERNEL_DRIVER`

- Used to load code into the Kernel

SERVICE INFORMATION IN THE REGISTRY

`HKLM\System\CurrentControlSet\Services`

- Start value = 0x03 for "Load on Demand"
- Type = 0x20 for WIN32_SHARE_PROCESS



Name	Type	Data
(Default)	REG_SZ	(value not set)
DependOnService	REG_MULTI_SZ	LanmanWorkstation LanmanServer
Description	REG_SZ	@%systemroot%\system32\browser.dll,-101
DisplayName	REG_SZ	@%systemroot%\system32\browser.dll,-100
ErrorControl	REG_DWORD	0x00000001 (1)
FailureActions	REG_BINARY	84 03 00 00 00 00 00 00 00 00 00 00 03 00 00 00 14 00...
Group	REG_SZ	NetworkProvider
ImagePath	REG_EXPAND_SZ	%SystemRoot%\System32\svchost.exe -k netsvcs
ObjectName	REG_SZ	LocalSystem
Start	REG_DWORD	0x00000003 (3)
Type	REG_DWORD	0x00000020 (32)

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Browser

COMPONENT OBJECT MODEL (COM)

Allows different software components to share code

Every thread that uses COM must call `OleInitialize` or `CoInitializeEx` before calling other COM libraries

COM objects are accessed via Globally Unique Identifiers (GUIDs)

There are several types of GUIDs, including

- Class Identifiers (CLSIDs)
  - in Registry at HKEY_CLASSES_ROOT\CLSID
- Interface Identifiers (IIDs)
  - in Registry at HKEY_CLASSES_ROOT\Interface