**System Call Modules**

* A system call is the mechanism a program uses to request service from the kernel.
  + E.G:
    - We have a program that reads data from one file and copies it into another file.
    - The information this program needs: data *from the user* with the *names* of the two files (input, output).
    - This flow will require the following system calls:
      * Write a message (the prompt) to the screen.
      * Read from an input to the OS (from the keyboard hardware.)
  + <https://www.guru99.com/images/1/121119_0451_SystemCalli1.png>
* When we write our own system call, what we are essentially doing is writing a kernel module that can be called from userland code.

**Requisite Functions/Structs**

* The System Call function:
  + This is the actual function that implements the system call.
  + It takes its arguments from user space, but the function itself will execute in kernel space. Note that this means arguments must be passed in by value (not reference).
  + typedef int sy\_call\_t(struct thread \*, void \*);
    - This is the prototype of the function.
    - It’s a return type int, and expects to be able to take in a thread pointer arg, and a void pointer arg.
  + The system call args should be declared within their own struct (e.g., struct sc\_example\_ags { … };

*NB because I forgot this:* (From GeeksForGeeks) A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be typecast to any type.

* An overview of the system call is defined by an entry in a **sysent** struct:

struct sysent {

int sy\_narg; /\* number of arguments \*/

sy\_call\_t \*sy\_call; /\* implementing function \*/

au\_event\_t sy\_auevent; /\* audit event associated with system call \*/

};

* + I say this is a sort of overview because an array of these structs exists in the system call table.
  + When a system call is installed, this sysent struct is placed within a free slot in *sysent[]*.
* Offset Value:
  + This is a distinct integer between 0 – 456 that is assigned to each system call.
  + It indicates the sysent struct’s offset in the sysent[] system call table.
  + In a system call module, this must be explicitly declared. This can be done with:
    - static int offset = NO\_SYSCALL; // NO\_SYSCALL sets offset to the next available free index in sysent[]
* SYSCALL\_MODULE Macro
  + From the docs:
    - **SYSCALL**\_**MODULE**(*name*, *int* *\*offset*, *struct* *sysent* *\*new*\_*sysent*, *modeventhand*\_*t* *evh*, *void* *\*arg*);
    - <https://www.freebsd.org/cgi/man.cgi?query=SYSCALL_MODULE&sektion=9>
  + We could load our own syscall module into the kernel using DECALRE\_MODULE (it is still a module after all). However, this would become pretty clunky – we’d need to shove all the extra data SYSCALL\_MODULE requires into another struct (we would call it something like [name]\_syscall\_mod), which would then contain `evh, arg, offset, new\_sysent, { 0 , NULL}. See below for an example from *Designing BSD Rootkits*:

#define SYSCALL\_MODULE(name, offset, new\_sysent, evh, arg) \

static struct syscall\_module\_data name##\_syscall\_mod = { \

evh, arg, offset, new\_sysent, { 0, NULL } \

}; \

\

static moduledata\_t name##\_mod = { \

#name, \

syscall\_module\_handler, \

&name##\_syscall\_mod \

}; \

DECLARE\_MODULE(name, name##\_mod, SI\_SUB\_DRIVERS, SI\_ORDER\_MIDDLE)

* + The highlighted line is where the extra args are passed into DECLARE\_MODULE (recall that when we last used DECLARE\_MODULE, we just left this blank.
  + ## is a pre-processor directive. Whenever something within that define has name##, it will be replaced by whatever we typed into the name var in SYSCALL\_MODULE. This is why we don’t need to put the name as a string.
* For the Syscall Macro, the new arguments mean the following:
  + Offset – as above.
  + New\_sysent – as above, passed as a struct sysent pointer.
  + Evh – event hander function.
  + Arg – for our purposes, this will be NULL.

**Executing the System Call:**

We need a couple of things first:

* Modfind function:
  + Int modfind(const char \*modname); Returns a modid, which is used to identify each module in the system.
* Modstat function:
  + Returns the status of a kernel module from its modid.
  + Int modstat(int modid, struct module\_stat \*stat);
  + The returned info is stored in the stat module\_stat structure we passed in by reference.
* Module\_stat struct:

struct module\_stat {

int version;

char name[MAXMODNAME]; /\* module name \*/

int refs; /\* number of references \*/

int id; /\* module id number \*/

modspecific\_t data; /\* module specific data \*/

};

typedef union modspecific {

int intval; /\* offset value \*/

u\_int uintval;

long longval;

u\_long ulongval;

} modspecific\_t;

* Syscall function:
  + Executes the system call specified by its system call number, which can be found above with modstat(it’s the ***intval*** var).
  + Int systemcall(int number, …)
  + The three dots after the first argument makes this function a *variadic function*. This means that it will accept an arbitrary number of parameters after the one that has already been defined.