The eXtended subsystem: Native Module Execution

The idea behind this subsystem is to provide a way to execute arbitrary code in the private process's address space on the server. Be aware that the implementation of the server is not compelled to implement any of the extended subsystems. The client can test if this subsystem is implemented by using one of its commands. If the server responses "UNKNOWN", you will realize that the subsystem is not implemented.

XS_NTMEX

XS_NTMEX makes the server enters the subsystem. Once inside the subsystem, the server remains there until the client explicitly exits the module with the proper commands. The return status for this command could be:

OK
UNKNOWN

The UNKNOWN response is due the lack of implementation on the server.
The OK response means the server is inside the module.

INSKEY "key"

INSKEY Install a pre-shared key that enables the process -on the server side- to execute native code in its private address space. The first parameter is a "key" -whatever this means- that the server recognizes. The return status for this command could be:

OK
FAILED 3 : Key already set.
FAILED 1 : Failed to read key's database.
FAILED 2 : Failed to set the key.

This command should be executed before any other in order to gain access to the rest of the commands of NTMEX.

LOAD “path/to/module”

LOAD loads into a handle the shared object -in native code- specified as the first parameter. This command should be called before **RUN**. The return status for this command could be:

OK
FAILED 5 : Access key have not installed yet.
FAILED 1 : Access denied to location.
FAILED 6 : Failed to load the module.

RUN

RUN runs the entry-point inside the previously loaded module -shared object-. The entry point is a defined function that is executed to handover the program pointer - control flow - to the module and its program logic. The definition of the entry-point’s function prototype -method signature- must meet some specifications:

The below indications are made by having C programming language in mind as it is one of the most used languages for server’s programming. However, if the server is implemented in any other language, the following specifications must be met:

The first parameter of the entry-point function must be a pointer -delegate, pointer to method, interface, and so on. - to a function that returns void, take as its first parameter a pointer to char and as its second parameter, a 64bit signed integer. This function pointer should point to the basic read function of the protocol. The second parameter is the same as the first one, except that the pointer is intended to point to basic write function of the protocol. Then comes two more function pointers with the same prototype and for the same purpose, except that it points to a couple of read/write function with extended or reduced attributes. Whatever ‘extended or reduced’ attributes mean, it is implementation-dependent. There is a last parameter in the entry-point function: it´s a pointer to a function that returns **int** and takes as first parameter a pointer to char and as a second parameter, another pointer to char. The purpose of this function is to receive through its first parameter a ‘source’ path and copy it to its second parameter but concatenating a token at the beginning. This is intended to ‘in-jail’ any path ‘-the second parameter-’ in the protocol’s root directory.

The prototype of the entry-point function in C could be as follows:

```
typedef void (*Ntmexentry)(int64_t (*r_io)(char *buf, int64_t len),
    int64_t (*w_io)(char *buf, int64_t len),
    int64_t (*rex_io)(char *buf, int64_t len),
    int64_t (*wex_io)(char *buf, int64_t len),
    int (*jd)(const char *src,char *dst));
```

Normally that function’s pointers provide a convenient way to use the transmission and encryption layers of the protocol.

The return status of this command could be:

OK
FAILED 5 : Access key have not installed yet.

FAILED 4 : Entry-point not assigned.

If OK is returned, then from that point on, all the further logic is defined by the executed module.

GOBACK

GOBACK returns the program pointer - control flow - to the TFProtocol command interface without unloading the module and without unsetting the entry-point. This means that once the **XS_NTMEX** command is called again, the **RUN** command can be executed without executing first **INSKEY** and **LOAD**, except in the case that another module has to be loaded. In that case calling LOAD will be sufficient.

The return status of this command could be:

OK

EXIT

EXIT exits the module unloading the module and it does the unset of the entry-point. If the module is entered again, it must call first **INSKEY** and **LOAD** in order to be able to call **RUN**.

The return status of this command could be:

OK

SYSNFO

SYSNFO returns valuable information of the execution environment of the TFProtocol's server daemon. This information should be taking into account to build the native module 'NTMEX'. The information tells about processor architecture, operating system, kernel version, and so on. The exact returned information and its format is implementation-depended.

The return status of this command could be:

OK 'the returned information'
FAILED 5 : Access key have not installed yet.
FAILED 7 : Information unavailable.