C/C++ Runtime Library Code Tampering in Software Supply Chain Attacks

Mohamad Mokbel Trend Micro

CASE STUDY - SHADOWHAMMER

The actual implementation of the ShadowHammer poisoned function __crtExitProcess() resides inside the file crt0dat.c as follows:

```
void __cdecl __crtExitProcess (
        int status
        )
{
        __crtCorExitProcess(status);

        /*
         * Either mscoree.dll isn't loaded,
         * or CorExitProcess isn't exported from mscoree.dll,
         * or CorExitProcess returned (should never happen).
         * Just call ExitProcess.
         */

        ExitProcess(status);
}
```

The CRT function __crtCorExitProcess() is responsible for checking if the process is part of a managed app, and if so, it calls the CorExitProcess(), otherwise it calls ExitProcess(). Said function is also defined in the crt0dat.c. The object file crt0dat.obj resides inside the library file libcmt.lib.

Contrast above benign implementation with ShadowHammer's implementation as shown in figure 1:

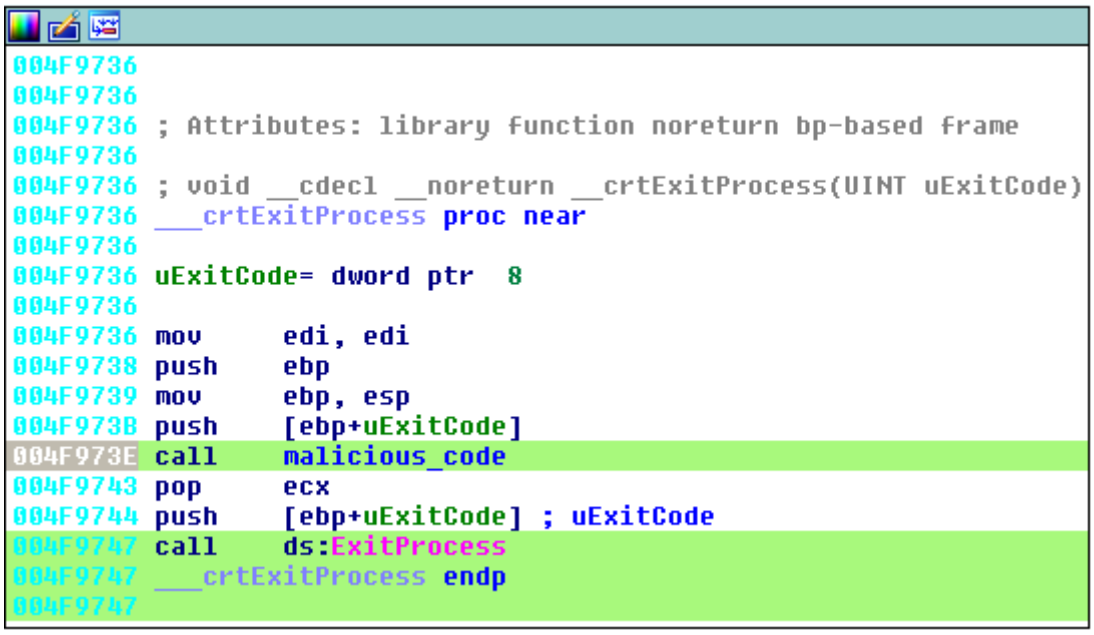

Figure 1 ShadowHammer poisoned __crtExitProcess() runtime function

It is clear that the CRT function __crtCorExitProcess() was overwritten with a malicious function that contains the malware's shellcode (the call at address 0x004F973E). This is such an insidious modification that is very hard to detect.

Figure 2 shows the cross-reference graph of the __crtExitProcess() CRT function as referenced by the ShadowHammer compiled code. The graph shows all call paths (reachability) that lead to it, and all other calls it make itself. The actual call path that leads to executing ShadowHammer code is:

Start() -> __tmainCRTStartup() -> _fast_error_exit() -> __crtExitProcess() -> malicious_code()

The malicious_code() function is also reachable via the CRT functions, _malloc(), _doexit() and __mtinitlocknum().

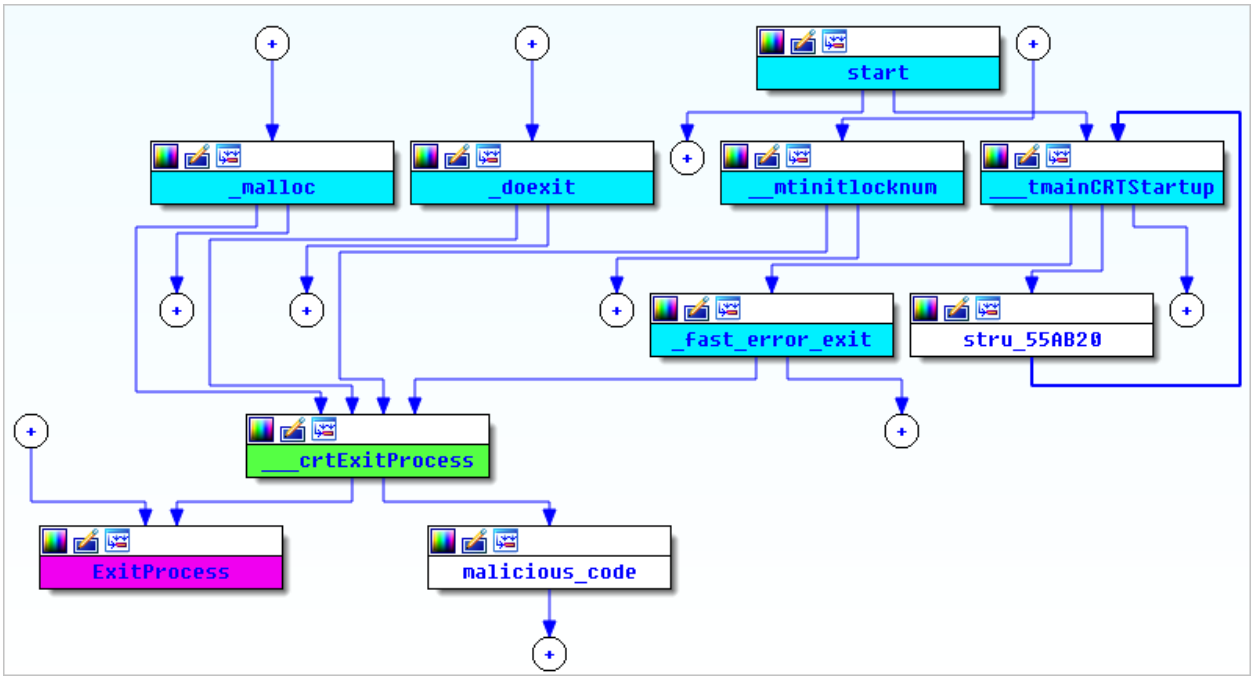

Figure 2 ShadowHammer poisoned function – call xref