COMPX508 – Malware Analysis

Week 10

Lecture 1: Code Injection -1

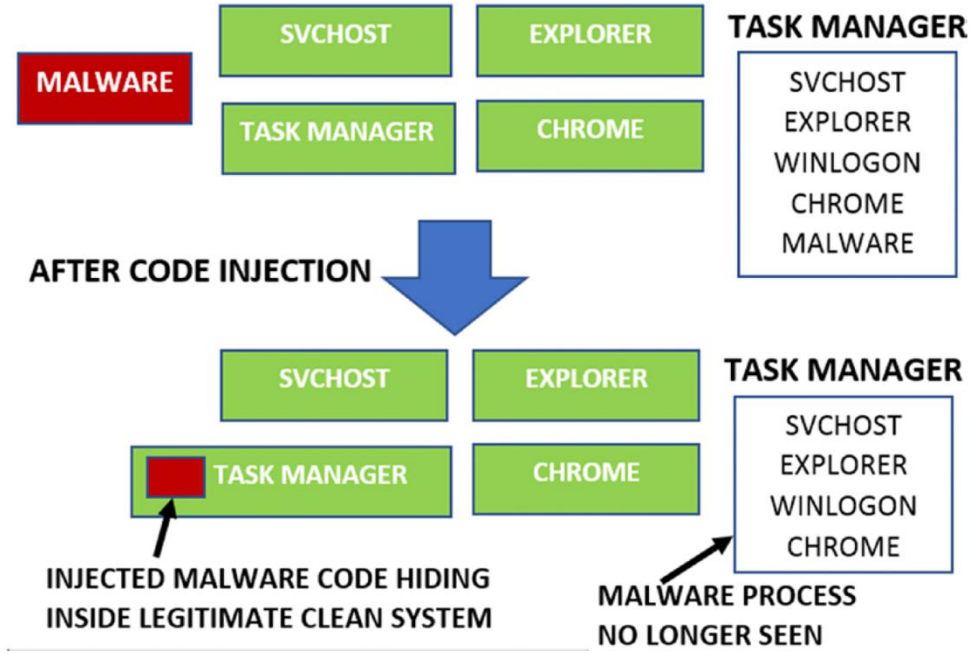
Vimal Kumar

Code Injection

- Code injection techniques are used to insert malicious code in benign processes
- Code injection is widely used by malware in many of the common malware functionalities, such as stealth, armouring, persistence etc.
- As with most techniques used by malware code injection was developed for use by legitimate software first and has since been adopted by malware authors
- Malware mainly uses code injection for the following reasons.
  - Hiding their presence
  - Process piggybacking
  - Altering functionality of another process

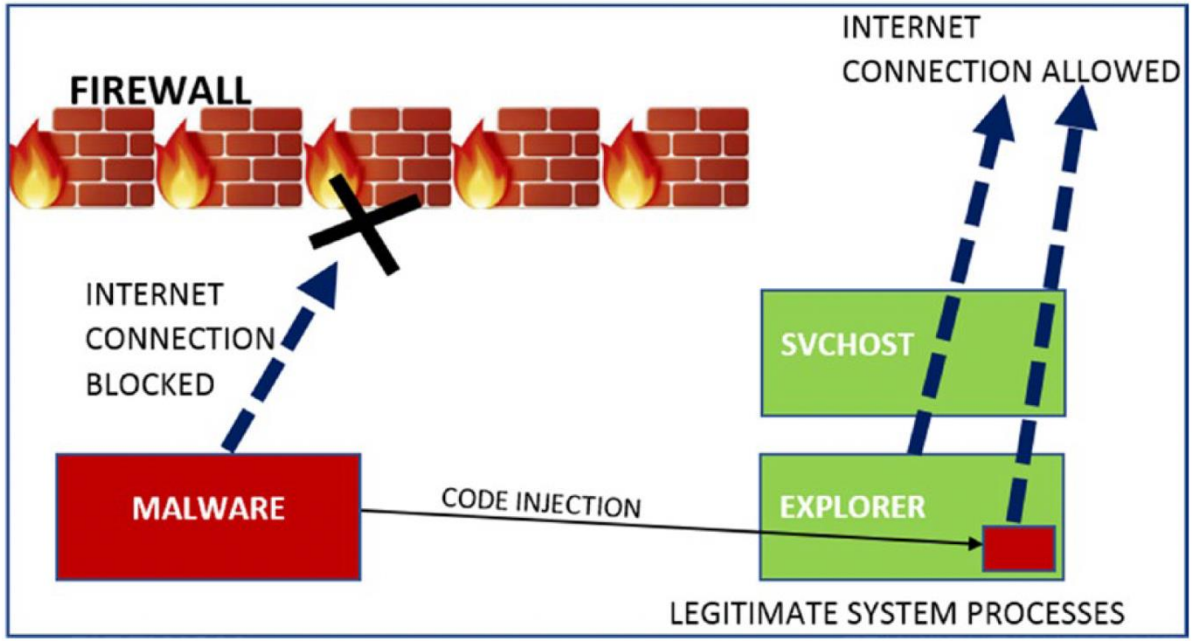
Hiding

- The malware process wants to hide its presence, and it does so by injecting all or part of its code into other legitimate processes running on the system (e.g., explorer, svchost, and winlogon) and exiting its primary malware process.
- Though the primary malware process now has exited, the malware is still running but as a part of another legitimate process via the code it earlier injected.



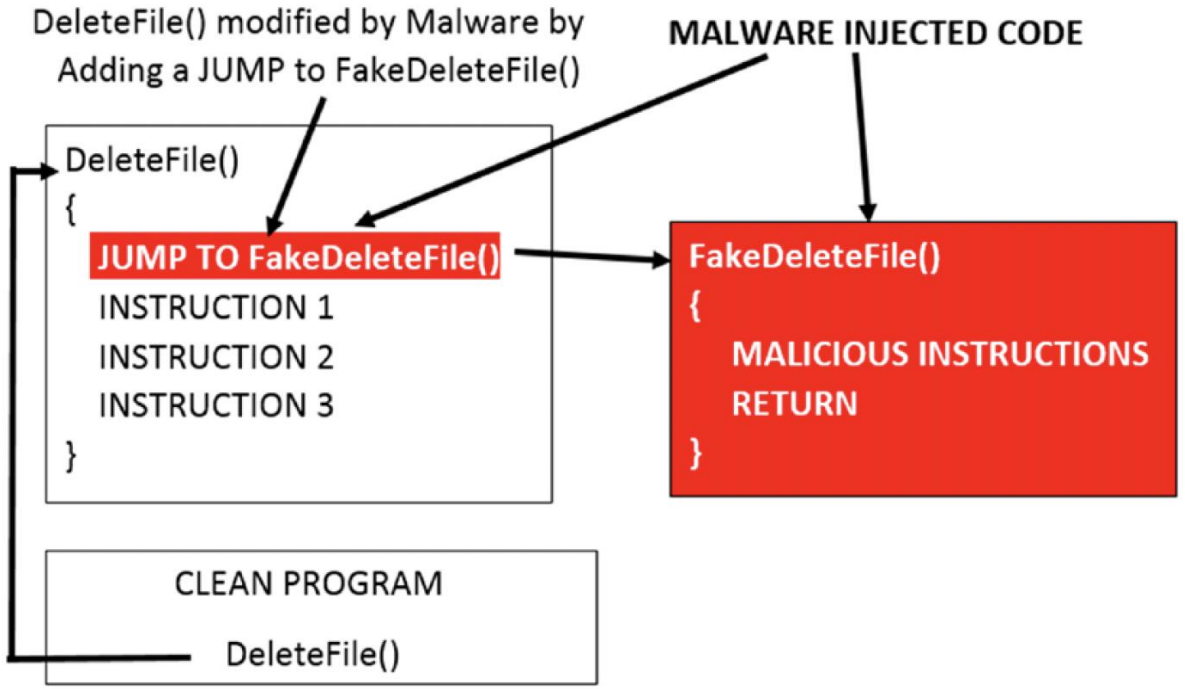
Process Piggybacking

- If malware wants to connect to the Internet, a firewall on the system might block this from happening, if it tries to connect from its own created process. The reason could be the firewall on the system might allow only a few well-known processes on the system to connect to the Internet.
- Malware can inject its code and run from other legitimate native processes like explorer, svchost, winlogon, and so forth, which have permission to connect to the Internet.
- So by piggybacking on the permission and privileges of these other legitimate processes, the malware can bypass restrictions put by the OS policies



Altering functionality of another process

- Another motive of code injection is to alter the functionality of certain processes or maybe even the entire OS/system
- This is usually used by rootkits to hide certain activities



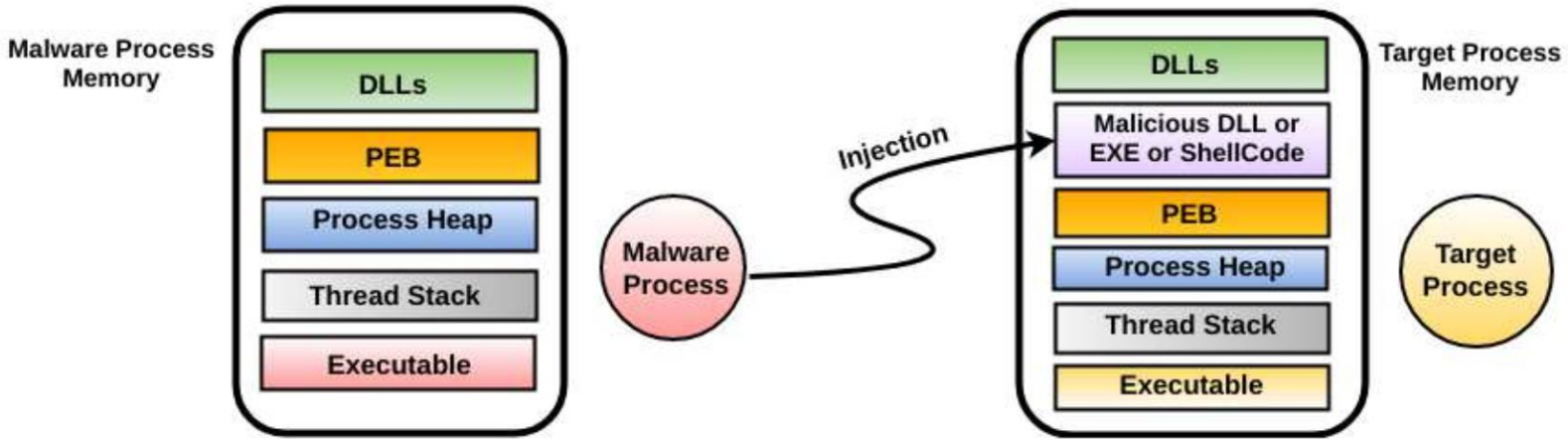
Binary Modification

- The most straightforward way to inject code in a binary is to directly modify it.
- Steps to modify a binary
  - Identify the code to modify using a disassembler
  - Edit the identified code in the binary using a hex editor

- Limitation
  - You can only edit existing code, you can’t add new code to the binary

Overview of Code Injection in a process

1. Locate the target process
2. Allocate memory in the target process
3. Write into the allocated memory
4. Execute the code



Locate the target process

- Malware would usually know the name of the process they are targeting but in order to programmatically access the process they need its PID.
- On Windows `CreateTool32HelpSnapshot()` function in the Win32 API is often used to get an object containing a list of all the current processes with their PIDs

```
std::vector<DWORD> find_pids_by_name(const std::wstring& target_name) {
    std::vector<DWORD> pids;

    HANDLE snap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (snap == INVALID_HANDLE_VALUE) return pids;

    PROCESSENTRY32W pe;
    pe.dwSize = sizeof(pe);

    std::wstring target_lc = to_lower(target_name);

    if (Process32FirstW(snap, &pe)) {
        do {
            std::wstring exe = pe.szExeFile;
            if (to_lower(exe) == target_lc) {
                pids.push_back(pe.th32ProcessID);
            }
        } while (Process32NextW(snap, &pe));
    }

    CloseHandle(snap);
    return pids;
}
```

Locate the target process

- Once the malware knows the PID, it will use the `OpenProcess()` function to get a handle for that process using the PID.

```
//Get a handle on the process
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
if (!hProcess) {
    std::cerr << "OpenProcess failed. Error: " << GetLastError() << "\n";
    return 1;
}
```

Locate the target process: Detection

- Usage of `CreateTool32HelpSnapshot()` and `OpenProcess()` functions is an indication that a process is trying to get a handle on another process.

- You might also see `Process32First()` and `Process32Next()` functions that are used to iterate over the object returned by `CreateTool32HelpSnapshot()`

Allocating memory and writing to it, in the target process

- Malware would use memory allocation functions to allocate memory of a specific size, usually with read, write execute permissions.
- In the Windows API `VirtualAllocEx()` function is used for memory allocation within a remote process. The function requires the handle obtained from `OpenProcess()`

```
SIZE_T allocSize = (dll_to_inject.size() + 1);

LPVOID remoteMem = VirtualAllocEx(
    hProcess,
    NULL,
    allocSize,
    MEM_COMMIT | MEM_RESERVE,
    PAGE_EXECUTE_READWRITE
);


std::cout << "Allocated " << allocSize
    << " bytes in process " << pid
    << " at address " << remoteMem << "\n";
```

Allocating memory and writing to it, in the target process

- In this next step the malicious code is actually written in the benign process's memory. The Win32 API function used for this is `WriteProcessMemory()`

```
SIZE_T bytesWritten = 0;
BOOL ok = WriteProcessMemory(
    hProcess,                   // writing into current process for demo
    remoteMem,                  // base address in target to start writing
    dll_to_inject1,             // source buffer (in this process)
    allocSize,                  // how many bytes to copy
    &bytesWritten               // receives number of bytes actually written
);
```

Allocating memory and writing to it, in the target process: Detection

- Usage of `VirtualAllocEx()` and `WriteProcessMemory()` functions is an indication that a process is allocating memory in the virtual address space of another process.

- Additionally functions such as `LookupPrivilegeValue()` `AdjustTokenPrivileges()` and `OpenProcessToken()` are used to set the right privilege level for the malware process.