# Anti-Anti-VM Detection

Aviad Yifrah
Supervisor: Igor Smolyar
3/4/17

## Malware analysis - basic

- Malware researches often use VM in order to analyze file behavior.
- The cons in this method are:
  - Prevention of unwanted effects on the machine like infection/data leak/credential theft etc.
  - Ability to revert machine status and compare by diff.
  - Better sniffing ability.
  - Etc.

## Basic Malware anti VM methods

- Because of that – very common behavior of malwares is to check if it is running inside VM.
- There are a lot of methods to perform those checks.
- Usually malwares check for evidence of the main virtualization products: VMWare, Virtual box, Qemu.
- The main methods used by malwares focus on six main sections:
  - VM common hardware configuration
  - CPU opcodes.
  - Typical Registry keys.
  - Typical Files.
  - Typical Processes.
  - Typical Services.
- There are more techniques which can be used like Timing Based techniques but those are the main methods I found.

## Existing solutions

- Manually configuration.
- Cuckoo sandbox – https://cuckoosandbox.org/
  malware analysis system which one of its components is a driver which hook syscalls – but its really hard to configure and had a lot of dependencies. Not plug&play.
- https://github.com/wbenny/avmext -
  Anti-Anti-VM solution via Windows Driver
- https://github.com/nsmfoo/antivmdetection -
  Script to create templates to use with VirtualBox to make vm detection harder

## My goal – Unique advantages

- Plug&Play – no dependencies, no need to install.
- Easy to configure – just add file name\registry value\process name and it would automatically hooked.

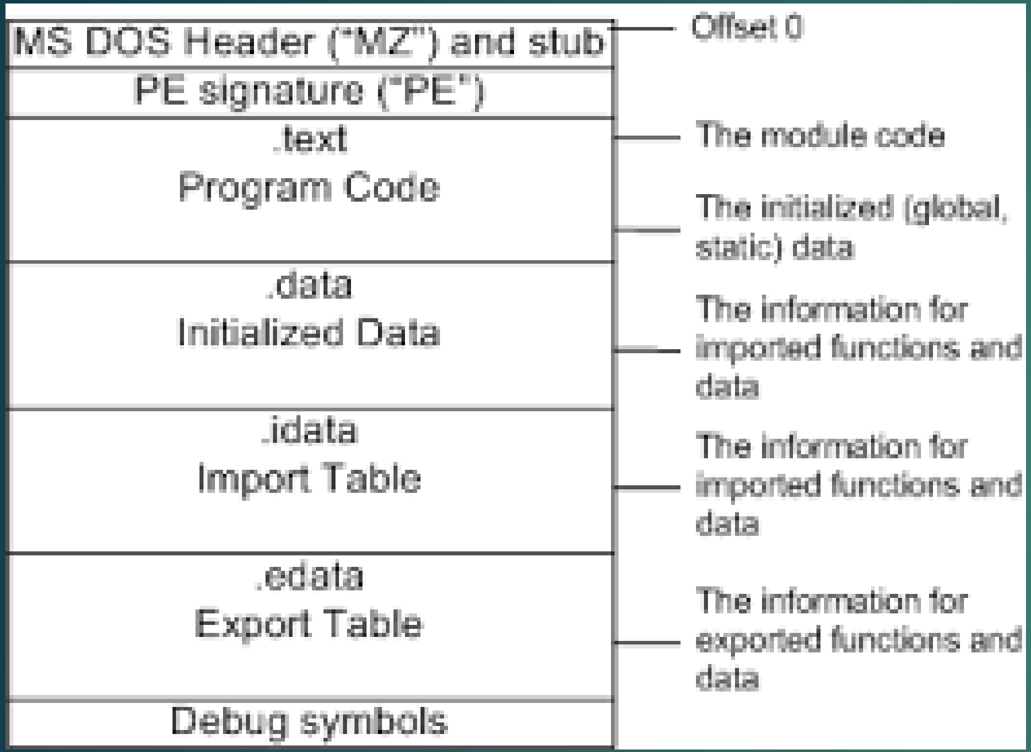
## Project goals & milestones

- Learn WIN32 API user mode Hooking methods
- Learn about code injection into windows process.
- Seek for open source c library which implement hooking by code injection – found “MinHook” library.
- Learn library usage examples.
- Read articles about common VM detection Techniques used by malwares in order to evade from analysis by malware researchers.
- Write simple to use, simple to configure, well documented, and stable tool that would hide VMWare machine characteristics as files, processes, and registry values from user mode process.
- Found test cases/tools to check success.
- Automate usage process.
- Write documentation.
- TODO list for Follow-up projects.

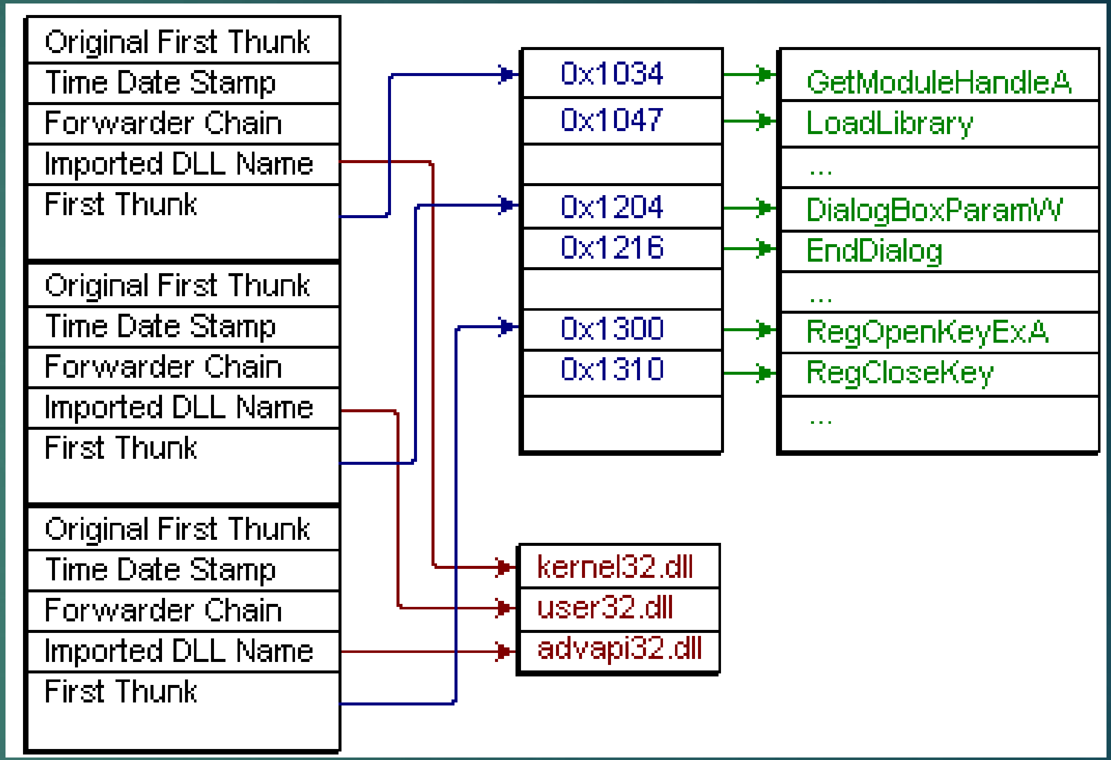
## Into to win32 API Hooking

- Replace the code of some function in process with our code while the process is running.
- Two main methods:
  - IAT (Import Address Table) hooking
  - Inline hooking.

# Import Address Table – PE format

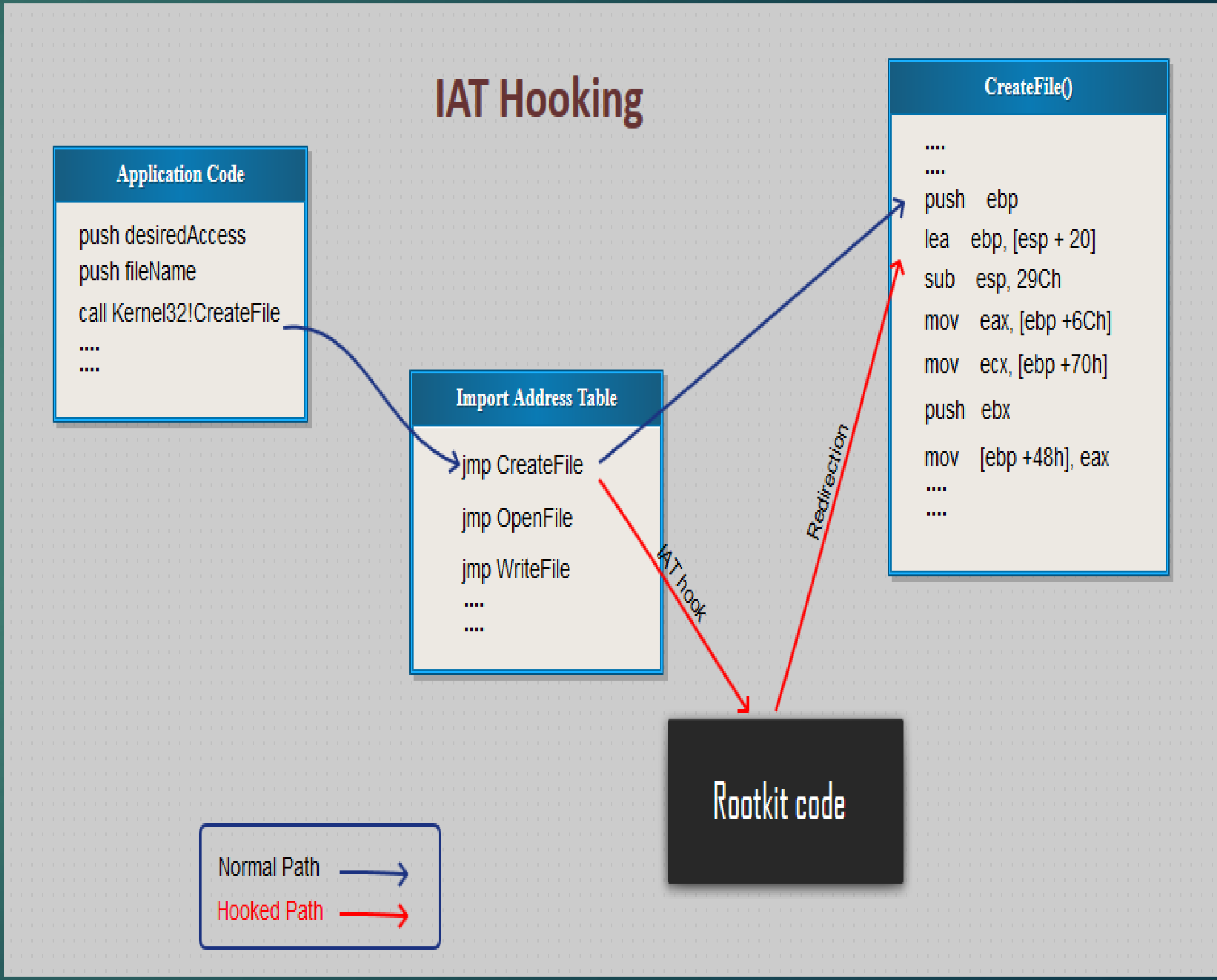


MS DOS Header ("MZ") and stub
PE signature ("PE")
.text
Program Code
.data
Initialized Data
.idata
Import Table
.edata
Export Table
Debug symbols
Offset 0
The module code
The initialized (global, static) data
The information for imported functions and data
The information for imported functions and data
The information for exported functions and data



# IAT Hooking

- When calling arbitrary API indirectly in programming language the compiler does not link direct calls to the module but it links call to IAT on jmp instruction which will be filled by process loader while OS is loading process to the memory.
- Process loader will fill out direct jmp instructions in IAT which is used by our calls from the program code
- So, if we are able to find out specific function in IAT which we want to hook, we can easily change jmp instruction there and redirect code to our address.



```
push desiredAccess
push fileName
call Kernel32!CreateFile
....
....
```

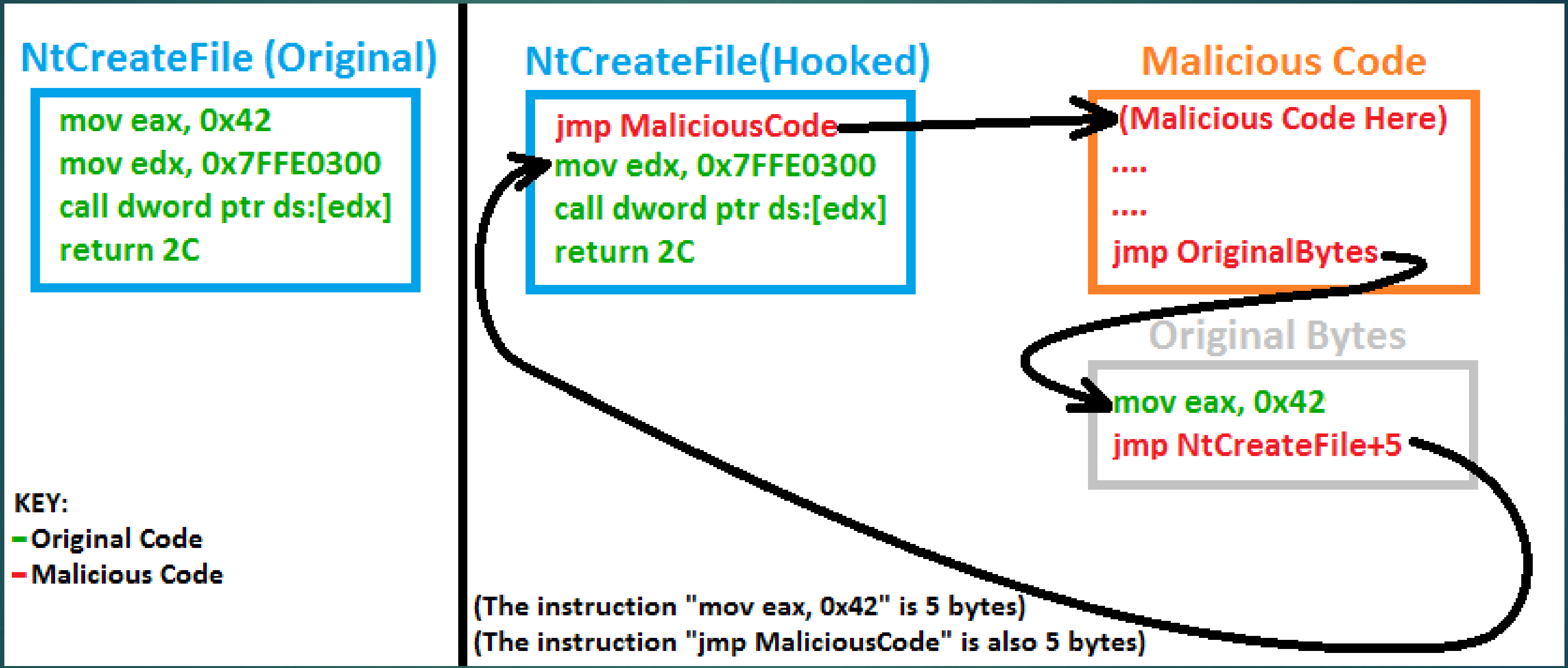
```
jmp CreateFile
jmp OpenFile
jmp WriteFile
....
....
```

```
....
....
push   ebp
lea    ebp, [esp + 20]
sub    esp, 29Ch
mov    eax, [ebp +6Ch]
mov    ecx, [ebp +70h]
push   ebx
mov    [ebp +48h], eax
....
....
```

## Inline hooking

- change a page protection.
- rewriting first few instructions on the function entry point – usually function prologue (16 bytes ).
- On this address we will insert relative jump to our code.
- Copy original instruction into the end of the new function.
- Add JMP instruction to the original function.

# Inline hooking



NtCreateFile (Original)

```
mov eax, 0x42
mov edx, 0x7FFE0300
call dword ptr ds:[edx]
return 2C
```

NtCreateFile(Hooked)

```
jmp MaliciousCode
mov edx, 0x7FFE0300
call dword ptr ds:[edx]
return 2C
```

Malicious Code

```
(Malicious Code Here)
....
....
jmp OriginalBytes
```

Original Bytes

```
mov eax, 0x42
jmp NtCreateFile+5
```

KEY:
- Original Code
- Malicious Code

(The instruction "mov eax, 0x42" is 5 bytes)
(The instruction "jmp MaliciousCode" is also 5 bytes)

## code injection - basic

- One of few methods to inject code into running process.
- technique used for running code within the address space of another process by forcing it to load a dynamic-link library.
- DLL injection methods:
  - DLLs listed in the registry entry HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs are loaded into every process that loads User32.dll during the initial call of that DLL.
  - Process manipulation functions such as CreateRemoteThread can be used to inject a DLL into a program after it has started.
  - Etc.
  - I choose the first method listed here.

## MinHook – hooking open source library

- https://www.codeproject.com/Articles/44326/MinHook-The-Minimalistic-x-x-API-Hooking-Libra
- The Minimalistic x86/x64 API Hooking Library
- Using inline hooking.
- replaces the prologue of the target function with x86's JMP (unconditional jump) instruction to the detour function.
- Updated open source and recommended by win32api developers.

## VM common hardware configuration

- No internet access/ external IP ownership
- NIC with specific MAC address for those vendors.
- Office RecentFiles property.
- File name – consist of md5/”malware”/”test”…
- No browser history.
- Low RAM/small Hard disk.

# Assembly opcodes checks

- **Checking CPU Instructions –**
  - *CPUID:*
    - Opcode that allowing software to discover details of the processor.
    - This instruction is executed with EAX=1 as input, the return value describes the processors features. The 31st bit of ECX on a physical machine will be equal to 0. On a guest VM it will equal to 1.
  - Hypervisor brand":
    - by calling CPUID with EAX=40000000 as input,1 the malware will get, as the return value, the virtualization vendor string in EAX, ECX, EDX. For example: Microsoft: "Microsoft HV", VMware : "VMwareVMware".

## Typical Registry keys

- HKLM\SOFTWARE\Vmware Inc.\\\Vmware Tools
- HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier
- SYSTEM\CurrentControlSet\Enum\SCSI\Disk&Ven_VMware_&Prod_VMware_Virtual_S
- SYSTEM\CurrentControlSet\Control\CriticalDeviceDatabase\root#vmwvmcihostdev
- SYSTEM\CurrentControlSet\Control\VirtualDeviceDrivers

## Typical processes

- Vmware
  - Vmtoolsd.exe
  - Vmwaretrat.exe
  - Vmwareuser.exe
  - Vmacthlp.exe
- VirtualBox
  - vboxservice.exe
  - vboxtray.exe

## Typical files

- *VMware*
  - C:\windows\System32\Drivers\Vmmouse.sys
  - C:\windows\System32\Drivers\vm3dgl.dll
  - C:\windows\System32\Drivers\vmdum.dll
  - *Etc…*
- *VirtualBox*
  - C:\windows\System32\vboxoglpassthroughspu.dll
  - C:\windows\System32\vboxservice.exe
  - C:\windows\System32\vboxtray.exe
  - Etc…

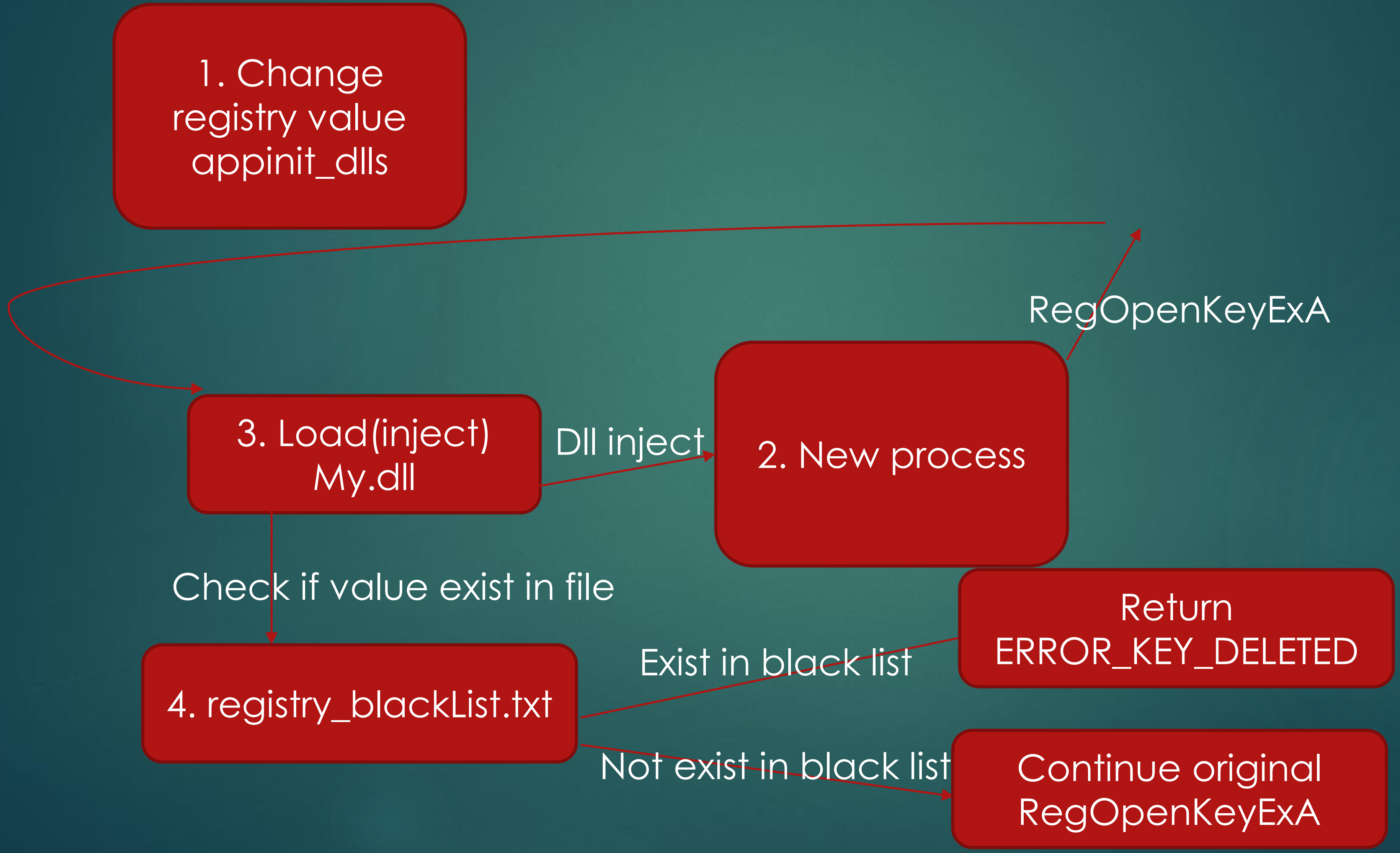
## Typical services

- VMTools
- Vmhgfs
- VMMEMCTL
- Vmmouse
- Vmrawdsk
- Vmusbmouse
- Vmvss
- Vmscsi
- Vmxnet
- vmx_svga
- Vmware Tools
- Vmware Physical Disk Helper Service

# Project description

- I focused on hooking win32 API calls to prevent detection by files, reg keys and processes.
- Project files:
  - HidingDLL.dll – main DLL which injected into processes and implement hooking.
  - files_blackList.txt + process_blackList.txt + registry_blackList.txt |
    – black list configuration files.
  - editReg.exe – edit appInit_Dlls and LoadAppInit_DLLs registry value to point to our dll.
- hooked functions:
  - MessageBox (test case - training)
  - RegOpenKeyExA
  - GetFileAttributesA
  - CreateFileW
- Hooking failure: Process32First, Process32Next.

# Project flow



## How to use:

- create win7 32bit vmware machine.
- Download https://github.com/AlicanAkyol/sems and run sems.exe to view vm detection result.
- Copy runFiles directory (all files listed above are included inside) to VM and run editReg.exe
- Diff between the results.

## How to configure

- Read minhook api documentation
- Create new dll project in visual studio
- Add minhook.h to include path
- Change relevant (through VS and OS version) libMinHook.lib to the right name.
- Carefully inject dll to process in any way u choose

## Testing

- Tested over win 7 32.
- Test case was with open source anti VM tool named sems.exe:
  - https://github.com/AlicanAkyol/sems
  - Described by its author as:
    sems is a tool which is created to help malware researchers by checking their environments for the signatures of any virtualization techniques, malware sandbox tools or well know malware analysis tools. sems is using the same techniques and looking for the same footprints that evasive malwares do in order to detect if it is running in a controlled environment. So it is useful for malware researchers to check if the analysis environment is inevasible.

## TODO list:

- Add support in process(debug failure) + services(complicated) hooking!!
- Evade detection by anti-hooking functions.
- Hook more relevant functions.
- Add ability to configure answer returned by hook function.
- Add another evasion techniques from checks listed above.
- Kernel mode hooking for better evasion.

# references

- Hooking:
  - https://www.codeproject.com/Articles/44326/MinHook-The-Minimalistic-x-x-API-Hooking-Libra
  - http://vxheaven.org/lib/vhf01.html
- AppInit_DLLs: https://support.microsoft.com/he-il/kb/197571
- Anti VM methods:
  - https://www.sans.org/reading-room/whitepapers/forensics/detecting-malware-sandbox-evasion-techniques-36667
  - https://www.zscaler.com/blogs/research/malicious-documents-leveraging-new-anti-vm-anti-sandbox-techniques
  - https://www.cyberbit.net/endpoint-security/anti-vm-and-anti-sandbox-explained/
  - http://resources.infosecinstitute.com/anti-debugging-and-anti-vm-techniques-and-anti-emulation/#gref
- anti vm\sandbox malware samples\PoC :
  - Sems tool:https://github.com/AlicanAkyol/sems