Divan van Kruiselbergen

26378396

Christiaan Markus Madeleyn

27211452

Keyboard Hooks

Keyboard Hooking Report

ITRW316 Assignment 4C

Contents

[Introduction**:** 2](#_Toc3815913)

[Body: 2](#_Toc3815914)

Source Modification……………………………………………………………………………………………………………………2

Runtime Modification…………………………………………………………………………………………………………………2

Virtual Method Table Hooking……………………………………………………………………………………………………[4](#_Toc3815915)

[**Conclusion:**……………………………………………………………………………………………………………………………………………..5](#_Toc3815932)

[Bibliography 6](#_Toc3815933)

# Introduction**:**

By rekenaarprogrammering dek die term 'hooking' 'n verskeidenheid tegnieke wat gebruik word om die gedrag van 'n bedryfstelsel, toepassings, ander sagteware-komponente te verander, versterk deur funksie oproepe, boodskappe of gebeure tussen sagteware komponente te onderskep. Kode wat sulke onderskepte funksie oproepe, gebeure of boodskappe hanteer, word 'n hook genoem.

Hooking word vir baie doeleindes gebruik, insluitende debugging en uitbreiding van funksionaliteit. Voorbeelde hiervan kan die insluit opneem van sleutelbord of muis gebeurtenis boodskappe voordat hulle 'n aansoek bereik, die oproepe van die bedryfstelsel onderskep om gedrag te monitor, die funksie van 'n program of ander komponent te verander.

Tipies word hooks ingevoeg terwyl sagteware reeds aan die gang is, maar hooking is 'n taktiek wat ook gebruik kan word voordat die program begin. Beide hierdie tegnieke word hieronder in meer detail beskryf (D, D and weergeven, 2019).

## Body:

**Source modification**

Deur die source van die library te verander voordat 'n program aan die gang is, kan jy ook deur middel van tegnieke van omgekeerde ingenieurswese gebruik maak. Dit word tipies gebruik om funksie oproepe te onderskep om hulle heeltemal te monitor of te vervang.

'n Alternatiewe metode vir die bereiking van function hooking is deur funksie oproepe te onderskep deur middel van 'n wrapper library. Wanneer jy 'n wrapper maak, maak jy jou eie weergawe van 'n library wat 'n aansoek laai, met dieselfde funksionaliteit van die oorspronklike library wat dit sal vervang. Al die funksies wat toeganklik is, is dieselfde tussen die oorspronklike en die vervanging. Hierdie wrapper library kan ontwerp word om enige van die funksies van die oorspronklike library te noem of om dit te vervang met 'n heeltemal nuwe stel logika (D, D and weergeven, 2019).

**Runtime modification**

Bedryfstelsels en sagteware kan event hooks maklik bied om in te voer. Dit is beskikbaar op voorwaarde dat die prosessering van die hook voldoende toestemming kry. Met Microsoft Windows kan u byvoorbeeld hooks insit wat gebruik kan word om system event en application events vir dialog, scrollbars, en menus sowel as ander items te verwerk of te verander. Dit laat ook 'n hook toe om sleutelbord- en muisgebeurtenisse in te voeg, te verwyder, te verwerk of te verander. Linux bied nog 'n voorbeeld waar hooks op soortgelyke wyse gebruik kan word om netwerkgebeurtenisse binne die kern deur NetFilter te verwerk (Nguyen, 2019).

**Virtual Method Table hooking**

Wanneer 'n klas 'n virtuele funksie of metode definieer, voeg kompileerders 'n hidden veranderlike by die klas wat na 'n virtuele metode tabel verwys. Die meeste compilers plaas die hidden VMT pointer by die eerste 4 bytes van elke instance. 'n VMT is basies 'n verskeidenheid wenke vir al die virtuele funksies wat gevalle van die klas kan roep. At runtime word hierdie pointers aangedui op die regte funksies, by compile tyd is dit nog nie bekend is of die base funksie geroep moet word of as 'n overridden weergawe van die funksie van 'n afgeleide klas nie. Daarom kan virtuele funksies gehook word deur die verwysings na hulle te vervang binne enige VMT wat hulle voorkom. Die onderstaande kode toon 'n voorbeeld van 'n tipiese VMT-haak in Microsoft Windows, geskryf in C ++ (D, D and weergeven, 2019).

#include *<iostream>*

#include *"windows.h"*

**using** **namespace** std;

**class** **VirtualClass**

{

**public**:

int number;

**virtual** void VirtualFn1() *//This is the virtual function that will be hooked.*

{

cout << "VirtualFn1 called " << number++ << "**\n\n**";

}

};

**using** VirtualFn1\_t = void(\_\_thiscall\*)(void\* thisptr);

VirtualFn1\_t orig\_VirtualFn1;

void **\_\_fastcall** hkVirtualFn1(void\* thisptr, int edx) *//This is our hook function which we will cause the program to call instead of the original VirtualFn1 function after hooking is done.*

{

cout << "Hook function called" << "**\n**";

orig\_VirtualFn1(thisptr); *//Call the original function.*

}

int main()

{

VirtualClass\* myClass = **new** VirtualClass(); *//Create a pointer to a dynamically allocated instance of VirtualClass.*

void\*\* vTablePtr = \***reinterpret\_cast**<void\*\*\*>(myClass); *//Find the address that points to the base of VirtualClass' VMT (which then points to VirtualFn1) and store it in vTablePtr.*

DWORD oldProtection;

VirtualProtect(vTablePtr, 4, PAGE\_EXECUTE\_READWRITE, &oldProtection); *//Removes page protection at the start of the VMT so we can overwrite its first pointer.*

orig\_VirtualFn1 = **reinterpret\_cast**<VirtualFn1\_t>(\*vTablePtr); *//Stores the pointer to VirtualFn1 from the VMT in a global variable so that it can be accessed again later after its entry in the VMT has been*

*//overwritten with our hook function.*

\*vTablePtr = &hkVirtualFn1; *//Overwrite the pointer to VirtualFn1 within the virtual table to a pointer to our hook function (hkVirtualFn1).*

VirtualProtect(vTablePtr, 4, oldProtection, 0); *//Restore old page protection.*

myClass->VirtualFn1(); *//Call the virtual function from our class instance. Because it is now hooked, this will actually call our hook function (hkVirtualFn1).*

myClass->VirtualFn1();

myClass->VirtualFn1();

**delete** myClass;

**return** 0;

}

## Program

Die program maak gebruik van een Form(form1) en twee klasse(LowLevelKeyboardHook.class en Mouse32.class). Die LowLevelKeyboardHook klas bevat al die nodige metodes om die aktiwiteite van die sleutel bord op te tel. Die Mouse32 klas maak dit moontlik om met die muis(curser) te kommunikeer deur die Point struct te skep wat ‘n x en ‘n y veranderlike bevat wat verwys na die posisie van die muis(curser). Beide die klasse maak gebruik van die User32.dll en/of die Kernal32.dll klasse om met die operating sisteem direk te kommunikeer. Die User32 en Kernel32.dll klasse implementeer verskeie sub seksies van die Win32 API. Die User32.dll implementeer die windows-gebruiker(USER) komponent wat die standaardelemente van die Windows-gebruikerskoppelvlak skep en manipuleer soos die “desktop”, “windows” en “menus”. In die LowLevelKeyboardHook klas roep die program funksies vanaf die windows-gebruiker(USER) deer die user32.dll klas om windows boodskappe te ontvang vanaf die sleutel bord. Mouse32 klas maak ook van user32dll gebruik, maar om boodskappe vanaf die muis te ontvang of na die muis stuur. Die Kerrnel32.dll word deur die LowLevelKeyboardHook klas gebruik om die process API’s van die Win32 bases bloot te stel aan die program.

Wanneer die program hardloop maak form1 oop met n button wat se “Start keyboard hook”. Die knoppie inisieer n thread wat konstant luister vir gebruikers inset, en maak form1 toe sodat die program nie sigbaar is nie en in die agtergrond hardloop. Die insette in die sleutel bord word as n string gestoor en na n text leer gestoor as die gebruiker die kombinasie van Ctrl+s in die sleutel bord in hou. N KeyPressed en KeyUnPressed event handler word invoke om kop te hou van wanneer die sleutel gedruk en gelos word en is helpvol om te identifiseer as kombinasies in die sleutel bord ingesit word. As gebruiker kan ook die posisie van die muis(curser) verander met gebruik van die pyltjies op die sleutelbord.

Die grootste struikelblok wat ons ervaar het was om die metodes en werking van die User32.dll en kernel32.dll te verstaan, dit was ons eerste keer wat ons daarmee moes werk en moes baie navorsing doen. Verder het ons inisieel die program as n Console applikasie geskryf en dit het goed gegaan, die keyboard hook het gewerk en ons kon die getikte informasie na n text leer stoor. Dit was toe ons die funksie wat die muis(curser) rond beweeg soos die pyltjies getik word wat ons probleme ervaar het. Die enigste oplossing wat ons kon vind was om die program oor te skryf as n Form Applikasie. Met klein veranderings aan die kode en n ander implementasie van threads kon ons, ons klasse steeds gebruik en die program het gewerk.

Die volgende ekstensies moes geïmplementeer word vir die nodige operasies van die program:

* System.Runtime.InteropServices; Gebruik om die User32.dll en die kernel32.dll te implementeer.
* System.IO; Om ’n text leer te skep.

System.Threading; Vir die gebruik van threads sodat die program op die agtergrond kan hardloop sonder dat die gebruiker daarvan weet.

# **Conclusion:**

By rekenaarprogrammering dek die term 'hooking' 'n verskeidenheid tegnieke wat gebruik word om die gedrag van 'n bedryfstelsel, toepassings, ander sagteware-komponente te verander, versterk deur funksie oproepe, boodskappe of gebeure tussen sagteware komponente te onderskep. Kode wat sulke onderskepte funksie oproepe, gebeure of boodskappe hanteer, word 'n hook genoem.

Bedryfstelsels en sagteware kan event hooks maklik bied om in te voer. Dit word tipies gebruik om funksie oproepe te onderskep om hulle heeltemal te monitor of te vervang.

Wanneer 'n klas 'n virtuele funksie of metode definieer, voeg kompileerders 'n hidden veranderlike by die klas wat na 'n virtuele metode tabel verwys. Daarom kan virtuele funksies gehook word deur die verwysings na hulle te vervang binne enige VMT wat hulle voorkom.

# Bibliography

D, J., D, J. and weergeven, M. (2019). *Hooking explained: detouring library calls and vtable patching in Windows/Linux/MAC-OSX*. [online] Ntvalk.blogspot.com. Available at: http://ntvalk.blogspot.com/2013/11/hooking-explained-detouring-library.html [Accessed 18 May 2019].

Nguyen, B. (2019). *Linux Dictionary*. [online] Tldp.org. Available at: http://www.tldp.org/LDP/Linux-Dictionary/html/ [Accessed 18 May 2019].