Communications of the CCISA
Vol. 24 No. 2 Apr. 2018

自動化生成 Windows SEH 機制之滲透測試腳本

黃中擇[1]、王智弘[2*]
[1,2]國立嘉義大學資訊工程系
[1]s1033018@mail.ncyu.edu.tw、[2]wangch@mail.ncyu.edu.tw

摘要

緩衝區溢位 (Buffer Overflow) 漏洞是對電腦的安全性威脅相當嚴重的一個問題。因此，當滲透測試 (Penetration Testing) 進行時須先找到程式是否存在這類漏洞，並想辦法利用一些作業系統或是程式本身的特性來作為滲透測試的切入點，並且測試是否可以達到攻擊的效果。然而，若能利用程式自動化生成可利用某種特性的滲透測試腳本，將有助於降低安全測試的成本。本文將介紹我們所實作的一套系統，讓測試人員在發現緩衝區溢位漏洞時，可以透過圖形化操作，在不須探究攻擊手法的情況下，自動化生成可利用 Windows SEH (結構化異常處理) 機制的滲透測試腳本。不但降低測試人員編寫滲透測試腳本的時間，並可以快速測試腳本是否能成功達成目標。

關鍵詞：滲透測試、自動化腳本生成、緩衝區溢位、安全漏洞、結構化異常處理

Automatic Penetration Testing Script Generation for Windows SEH Mechanism

Zhong-Ze Huang[1], Chih-Hung Wang[2*]
[1,2]Department of Computer Science and Information Engineering, National Chiayi University
[1]s1033018@mail.ncyu.edu.tw, [2]wangch@mail.ncyu.edu.tw

Abstract

The vulnerability of buffer overflow is a serious threat for computer security. Therefore, it is necessary to test whether the target program has this kind of vulnerability and find the entry point for breaking as the penetration testing process being carried out. Finally, the tester also needs to understand the attacking effects when exploiting the vulnerability. However, if the penetration testing script can be automatically generated by a program tool, it can reduce the cost of development of the exploits by the tester. This article introduces an implementation of the system that can automatically generate the penetration testing script by using Windows SHE (Structured Exception Handling) mechanism. The system can be executed through GUI interface and the tester even needs not to investigate details of attacking skills. Thus, the

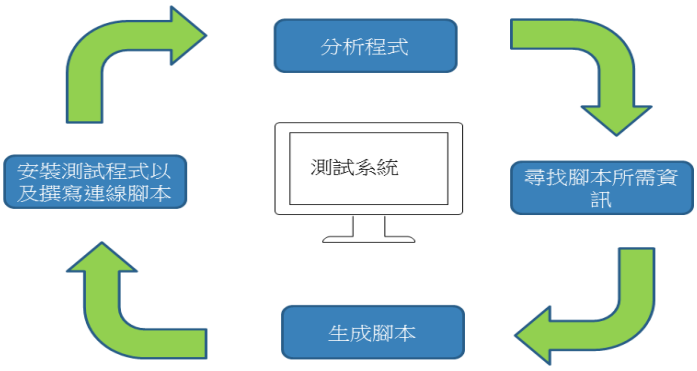
* 通訊作者 (Corresponding author.)

proposed system not only can reduce the development time of generating testing script but also can help the tester understand whether the generated scripts can successfully reach the purposes.

Keywords: Penetration Test, Automatic Script Generation, Buffer Overflow, Vulnerability, Structured Exception Handling

壹、前言

為了完成這套系統，我們依據滲透測試的步驟來進行對於程式的分析。整體架構如下：第一部分：為搭建測試環境，使用者設定好連線所使用的 IP、Port 以及連線所使用的腳本，第二部分：程式會對軟體進行分析，蒐集 IAT 表、程式可執行區段以及會造成緩衝區溢位攻擊截斷的壞字元，第三部分：利用蒐集到的資料嘗試自動化生成以利用 Windows SEH 機制進行為攻擊手法的滲透測試腳本，分為針對 Windows XP 以及針對 Windows XP 後作業系統兩種，第四部分：測試生成的滲透測試腳本確認程式是否存在可以使用 Windows SEH 機制作為攻擊手法的緩衝區溢位漏洞 (如圖一)。我們希望以上的方法可以減少使用者花時間對程式進行分析，從記憶體中尋找壞字元，測試滲透測試腳本使否可以使用，讓使用者可以把生成利用 Windows SEH 機制的攻擊手法的流程自動化。



圖一：自動化腳本生成系統架構

貳、相關背景與技術

我們介紹相關背景及技術如下。

2.1 位址空間配置隨機載入 (Address Space Layout Randomization, ALSR) 技術

目前大多數現代作業系統都會預設開啟的一套機制，若不開啟 ASLR 機制，程式每一次執行都會被載入相同的記憶體位置，一旦開啟 ASLR 機制，程式每一次執行都會被載入不同的記憶體位置 [2]，這會造成駭客無法控制程式跳轉到預載入的 shellcode，大幅提升緩衝區溢位攻擊的難度，但是，在 Windows 作業系統下並不是所有程式執行使用的模組都會開啟 ASLR 機制。

2.2 防止資料執行 (Data Execution Prevention, DEP) 技術

將程式執行的某塊記憶體上儲存的資料設定為不可執行狀態 [2]，對駭客的影響為寫好的 shellcode 會因所在記憶體區段開啟 DEP 防護而無法執行，造成駭客攻擊流程被中斷，同樣提升了緩衝區溢位攻擊的難度，現今大多數作業系統預設會開啟 DEP 防護。

2.3 除錯工具 Immunity debugger

這是一套免費的程式除錯工具，為駭客與資訊安全人員喜愛用來進行動態分析惡意程式以及找尋漏洞，提供資訊安全人員利用內部提供的功能自行編寫插件，在找尋漏洞的利用方式時，資訊安全人員常用來分析是否成功控制程式執行流程以及 shellcode 是否成功執行。

2.4 滲透測試工具 Metasploit

Metasploit 為一套滲透測試工具，將滲透測試流程簡化為設定好測試程式的版本、IP 與 Port，便能快速檢測使否存在漏洞，其程式碼都公開在網路上，大部分是針對 Windows 作業系統，可以做為資訊安全人員學習以及分析用，提供功能從弱點掃描、生成 shellcode 以及對 shellcode 進行編碼避免被防毒軟體與防火牆偵測。為目前市面上架構最完整與攻擊種類最多最新的滲透測試工具。

參、系統方法

以下幾個子章節說明我們建構自動化生成腳本系統的方法。

3.1 動態分析

相較於靜態分析使用分析程式原始碼的方式，動態則是分析直接執行程式，藉由執行過程中的各種操作，觀察程式執行時的暫存器數值以及記憶體變化，找出可能存在漏

洞的地方，在本系統中，主要是不斷用設計好的字串生成演算法生成字串並進行發送到架設在本機上的程式，直到程式發生崩潰或是到達設定的測試長度上限。假設發生崩潰時，本系統會利用 pydbg 呼叫 Windows 作業系統本身提供的 debugger API，將發生崩潰時的暫存器數據、stack 上的資料紀錄與 SEH (Structured Exception Handling) Chain，根據設計好的字串生成演算法去計算是否成功覆蓋到 SEH Chain，以及送多少長度字串會讓程式發生崩潰。

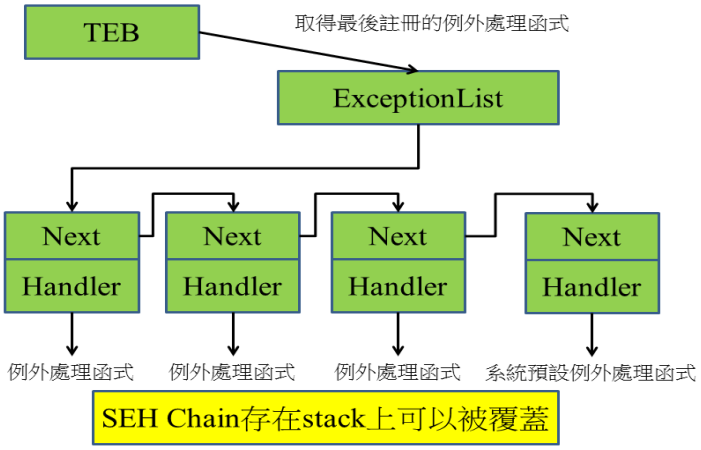
3.2 結構化異常處理 (Structured Exception Handling, SEH)

SEH 作為 Windows 作業程式執行中發生無法處理或開發者不允許的操作時，會將程式的控制權交由註冊在 SEH Chain 上的例外處理函式嘗試處理 [2]，通常程式開發者自行編寫的例外處理函式會靠近 SEH Chain 的頂端，系統預設的例外處理函式會靠近 SEH Chain 的底端。當例外發生時，程式作業系統會先將控制權交由 SEH Chain 頂端的錯誤處理函式，若例外能被解決，系統會將控制權歸還給程式，但當例外處理函式無法解決時，系統會將控制權交給 SEH Chain 上註冊的下一個錯誤處理函式直到交給 SEH Chain 上最後一個錯誤處理函式，系統預設的錯誤處理函式顯示程式出錯。(如圖二)

圖二：系統預設處理函式圖

由於 SEH 機制中，會將 SEH Chain 存在程式執行時的記憶體上，一旦程式存在緩衝區溢位漏洞，便有機會覆寫掉原本 SEH Chain 的資料結構，SEH Chain 為單向 link list，資料結構由 _EXCEPTION_REGISTRATION_RECORD *Next 與

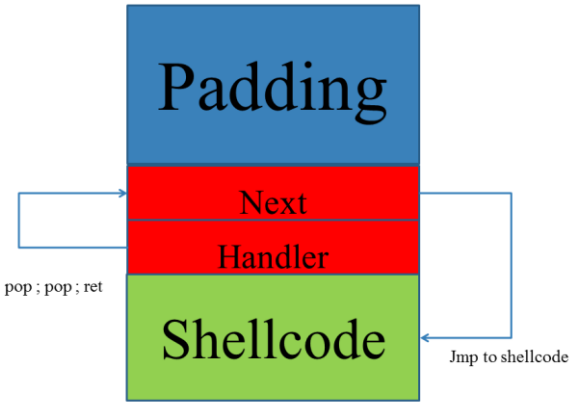
PEXCEPTION_ROUTINE Handler 組成，Next 指向下一個節點位址，Handler 指向錯誤處理函式的位址。(如圖三)



圖三：SEH 機制流程圖

在系統中自動化生成滲透測試腳本的部分，會以 Windows XP 作為分界，在 Windows XP 作業系統大多不會預設開啟 DEP 防護，但 Windows XP 後作業系統預設開啟 DEP 防護，因此，針對是否需要繞過 DEP 防護攻擊手法有所不同。

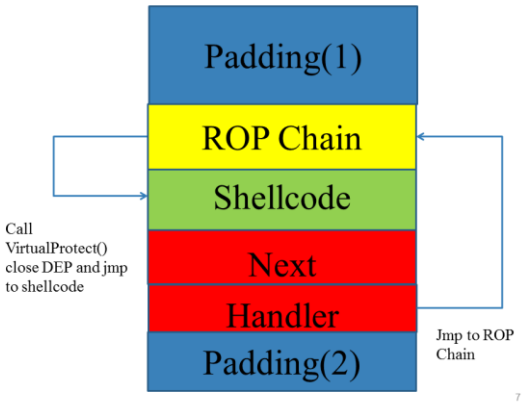
在通常不開啟 DEP 防護的 Windows XP 作業系統的情況下，通常會複寫 Next 變數為 jmp 到 shellcode 位置的指令，Handler 複寫為 pop ; pop ; ret 的 gadget 位址，使在移動 SEH Chain 時移動到 shellcode 位置。(如圖四)



圖四：Windows XP 利用 SEH 機制攻擊流程圖

若是針對 Windows XP 之後的作業系統會預設開啟 DEP 防護的作業系統，可以使用 ROP 的技術呼叫 Windows 系統內部函式 [4]，在本系統中，使用程式自動找出 VirtualProtect() 被存放在哪一個 DLL 的 IAT 表中，並利用 ROP 的技術改變程式執行流

程 [1]，完成呼叫 VirtualProtect() 關閉 DEP 防護並執行 shellcode。（如圖五）



圖五：Windows XP 之後的作業系統版本利用 SEH 機制攻擊流程圖

3.3 返回導向程式編寫 (Return-Oriented Programming, ROP)

ROP 藉由蒐集程式中的沒開啟 ASLR 和 DEP 防護可執行區段最後可以被解析為 ret 的指令，通常稱為 gadget [3]，由於，gadget 通常會從沒開啟 ASLR 和 DEP 防護可執行區段所取得，因此，每一個 gadget 都可以不受 ASLR 以及 DEP 防護影響對暫存器或是記憶體上的數據進行操作，攻擊者通常會分析所有 gadget 並找出，最後組成一段 ROP Chain 去執行某些惡意操作。在 Windows 作業系統上常被使用來呼叫 VirtualProtect() 函式關閉系統預設的 DEP 防護，並跳轉到攻擊者寫好的 shellcode 位置並執行。

肆、實作成果

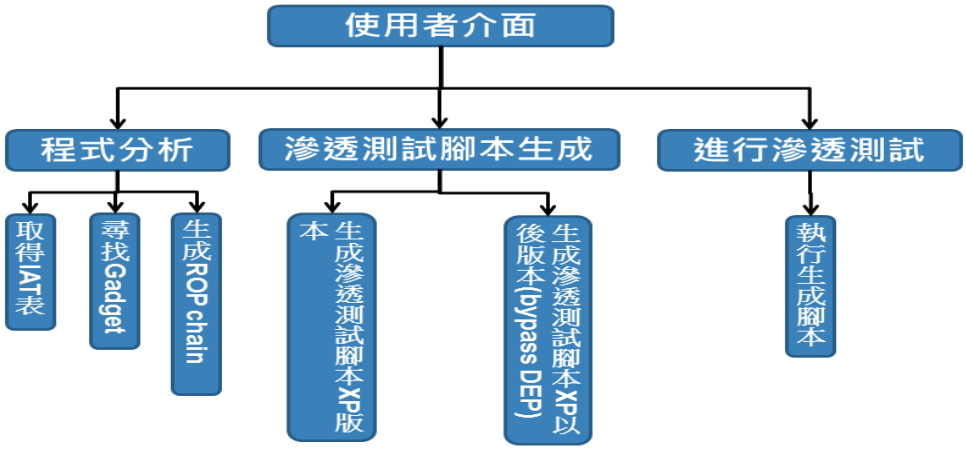
4.1 系統實作描述

本系統實作主要針對 Windows 作業系統的緩衝區溢位漏洞測試，主要測試程式是否存在緩衝區溢位漏洞、是否可以自動化生成利用覆寫 SEH 機制作為攻擊手法的滲透測試腳本以及是否生成的腳本可以成功進行滲透測試。自動化生成的滲透測試腳本分為針對 Windows XP 和 Windows XP 之後的作業系統版本兩種類型，主要分別為是否需要繞過 DEP 以及 ASLR 防護。

4.2 系統實作架構

在受測主機裝上受測程式，設定好 IP、Port 與連線腳本後，首先開始對受測程式進

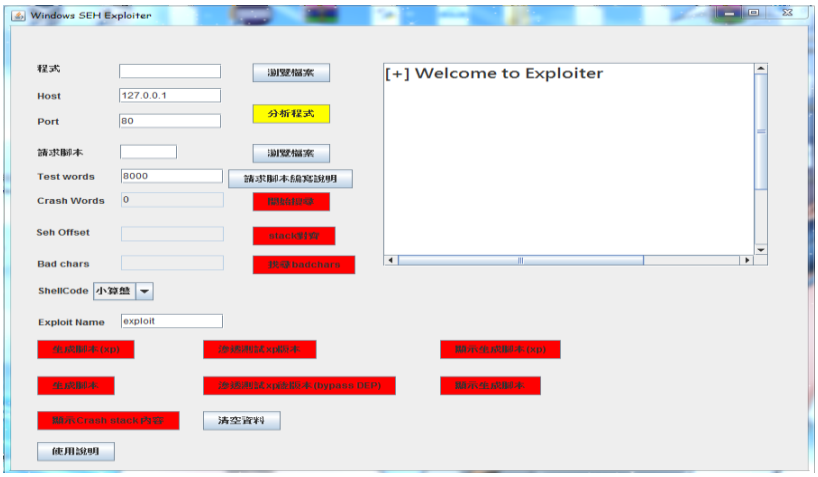
行分析，將所有執行時會載入的 DLL 以及受測程式執行檔本身進行分析，透過解析 PE 架構過濾掉系統本身的 DLL 以及有開啟 ASLR 以及 DEP 防護的模組，利用剩下的模組生成可以關閉 DEP 防護的 ROP Chain。接下來嘗試找出受測程式是否存在緩衝區溢位漏洞，若漏洞存在，便利用蒐集到攻擊所需要的資訊自動化生成利用 SEH 機制作為攻擊手法的受測測試腳本，最後嘗試執行生成的腳本以確認是否可以成功執行滲透測試。實作架構如圖六。



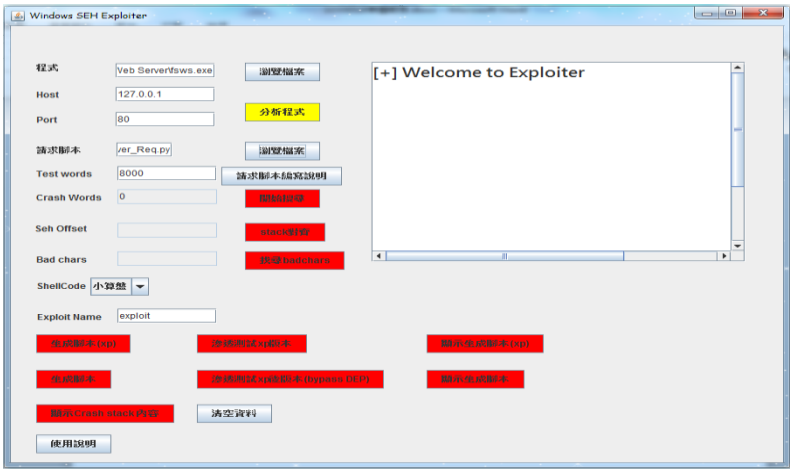
圖六：系統實作架構圖

4.3 系統操作介面及測試結果

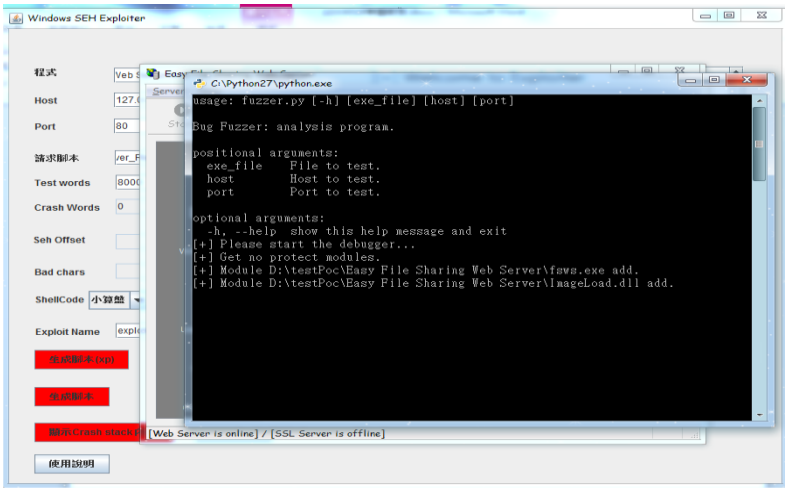
本系統的特色是可利用圖型化介面來設定主要的參數內容，包括攻擊主機之 IP 及 Port 等資訊（如圖七及圖八）。資訊填入完成之後，系統將會開始分析受測目標程式的各種資訊情況（如圖九），並搜集記錄相關內容（如圖十）。最後生成攻擊 Windows XP 及其他作業系統的腳本，儲存於系統中以便後續使用（如圖十一）。



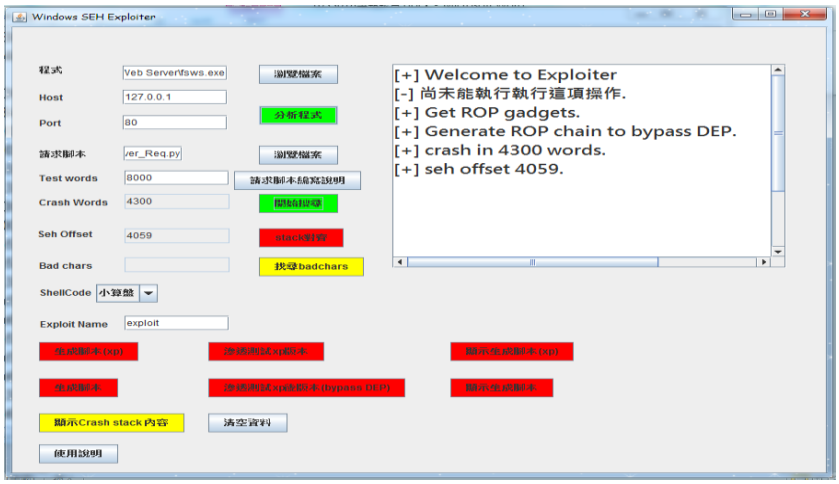
圖七：系統主畫面



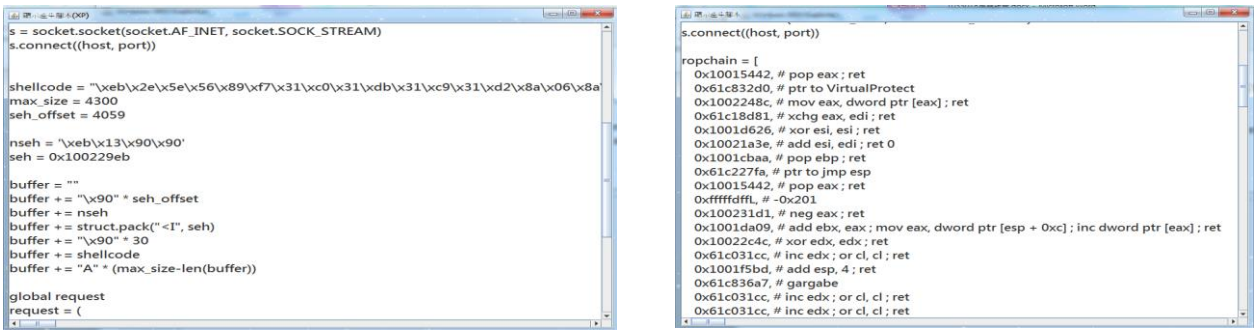
圖八：設定 IP、Port 與連線用腳本



圖九：對受測目標程式進行分析



圖十：蒐集生成腳本所需資料



```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))

shellcode = "\xeb\x2e\x5e\x56\x89\xf7\x31\xc0\x31\xdb\x31\xc9\x31\xd2\x8a\x06\x8a
max_size = 4300
seh_offset = 4059

nseh = '\xeb\x13\x90\x90'
seh = 0x100229eb

buffer = ""
buffer += "\x90" * seh_offset
buffer += nseh
buffer += struct.pack("<I", seh)
buffer += "\x90" * 30
buffer += shellcode
buffer += "A" * (max_size-len(buffer))

global request
request = (
```

```
s.connect((host, port))

ropchain = [
   0x10015442, # pop eax ; ret
   0x61c832d0, # ptr to VirtualProtect
   0x1002248c, # mov eax, dword ptr [eax] ; ret
   0x61c18d81, # xchg eax, edi ; ret
   0x1001d626, # xor esi, esi ; ret
   0x10021a3e, # add esi, edi ; ret 0
   0x1001cbaa, # pop ebp ; ret
   0x61c227fa, # ptr to jmp esp
   0x10015442, # pop eax ; ret
   0xfffffdffL, # -0x201
   0x100231d1, # neg eax ; ret
   0x1001da09, # add ebx, eax ; mov eax, dword ptr [esp + 0xc] ; inc dword ptr [eax] ; ret
   0x10022c4c, # xor edx, edx ; ret
   0x61c031cc, # inc edx ; or cl, cl ; ret
   0x1001f5bd, # add esp, 4 ; ret
   0x61c836a7, # gargabe
   0x61c031cc, # inc edx ; or cl, cl ; ret
   0x61c031cc, # inc edx ; or cl, cl ; ret
```

圖十一：自動化生成針對 Windows XP（左）及其之後作業系統（右）的腳本

最後，我們使用完成的腳本對於目標程式進行測試，以確認產生的滲透測試腳本之正確性（如圖十二）。

圖十二：測試腳本是否可以成功利用

伍、結論

本系統利用分析程式載入保護較少的模組以及觀察程式執行時是否可以覆蓋 SEH 機制，再透過整合分析過程中所蒐集的資料嘗試自動化生成針對 SEH 機制的滲透測試腳本，將以往編寫滲透測試腳本中的某些重要步驟予以自動化，縮短開發完成滲透測試腳本所需的時間。

然而系統尚在開發階段，目前能實際生成的腳本還不足，在 Windows 作業系統上其實還有很多很難繞過的防護措施存在，希望未來可以想辦法提高生成的腳本品質以及穩定性，設計一個更流暢的使用流程，讓整體系統更好使用，能實際對未知程式自動化生成一支滲透測試腳本。

[誌謝]

本文完成感謝科技部計畫 MOST 106-2221-E-415-003-MY2 的支持。

參考文獻

[1] L. Davi, A.-R. Sadeghi, D. Lehmann and F. Monrose, “Stitching the Gadgets: On the Ineffectiveness of Coarse-Grained Control-Flow Integrity Protection,” *Proceedings of 23rd USENIX Security Symposium*, pp. 401-416, 2014.

[2] G. Diathesopoulos, “Computer Laboratory Setup for the Assessment of State-of-the-art Penetration Testing Tools,” Master Thesis, Department of Digital Systems, University of Piraeus, 2017.

[3] M. Prandini and M. Ramilli, “Return-oriented Programming,” *IEEE Security & Privacy*, vol. 10, issue 6, pp. 84-87, 2012.

[4] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, “Return-oriented programming: Systems, Languages, and Applications,” *ACM Transactions on Information and System Security (TISSEC)* - Special Issue on Computer and Communications Security, vol. 15, no. 1, article no. 2, 2012.