Write XOR Execute

姓名：李名智
學號：1102924

一、W^X 介紹

「寫入 XOR 執行」（Write XOR Execute，簡稱 W^X）是一種安全功能，最早由 OpenBSD 操作系統引入。其核心理念是將記憶體區域設置為「可寫」或「可執行」，但不能同時具備這兩種權限。這樣的設計是為了防止攻擊者利用可寫記憶體區域來注入並執行惡意程式碼。

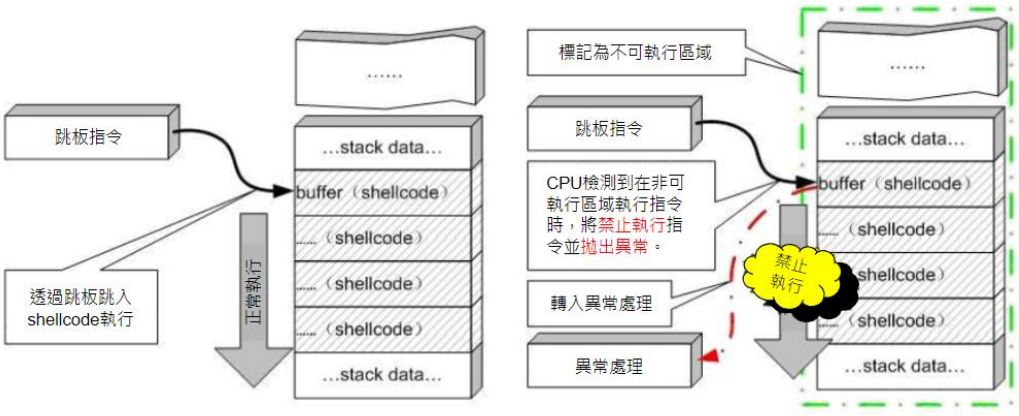
補充：OpenBSD 是一個開源的 UNIX-like 作業系統，專注於安全性、程式碼品質和加密技術。它的開發重點包括安全性、正確性以及程式碼的簡潔和清晰。OpenBSD 在許多方面都採取了一系列安全措施，其中包括寫入 XOR 執行（W^X）等技術來防止緩衝區溢位等攻擊。

OpenBSD 標誌

二、W^X 的防護

- **目的：**為了防止攻擊者利用可寫記憶體區域來注入並執行惡意程式碼。
- **原理：**W^X 確保記憶體區域要麼是可寫的，要麼是可執行的，但不能同時具備這兩種權限。這樣，即使攻擊者設法注入惡意程式碼，也無法執行。
- **保護範圍：**防止記憶體區域同時具備可寫和可執行權限，而阻止惡意程式碼注入和執行攻擊。
- **W^X 防護流程圖：**

圖片來源



- **有關記憶體區域的攻擊：**

1. 緩衝區溢出攻擊：

   Buffer 不做邊界檢查，覆蓋 Buffer 後面的記憶體區域，導致控制權被搶走。

2. 程式碼注入攻擊：

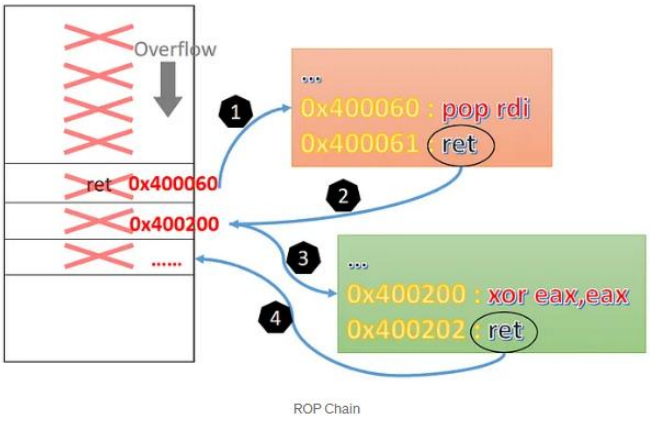
   將惡意程式碼注入到程式的記憶體，設法執行此程式碼。

3. 格式化字串攻擊：

   當不安全地使用格式化字串時，攻擊者可以讀取或寫入任意記憶體位置。

4. ROP 攻擊：

   利用存在於可執行文件中的合法程式碼，透過修改返回地址來執行這些程式碼以達到惡意目的。不需要注入新的程式碼，只需重新組合已有的程式碼。



ROP Chain

- **可防止攻擊的案例：**緩衝區溢出攻擊

過程：

1. 攻擊者利用緩衝區溢位漏洞，將惡意程式碼（如 shellcode）寫入目標程式的記憶體。
2. 攻擊者篡改函數的返回地址，使其指向已注入的惡意程式碼。
3. 當函數返回時，執行惡意程式碼。

防護：

記憶體頁面只能被寫或被執行，但不能同時具有這兩種權限。這樣防止了先寫入後執行的攻擊行為。

應用：

不可執行堆疊（即堆疊可寫但不可執行）是這種策略的一個例子。

- **無法防止攻擊的案例：**Return-into-libc（ROP 的一種）攻擊

過程：

1. 攻擊者不注入新的惡意程式碼，而是重用已有的程式碼片段（如庫

函數）。

2. 攻擊者篡改函數的返回地址，使其指向現有程式碼序列（如 libc 中的 system 函數）。

3. 傳遞重新設定好的參數，使程式按攻擊者的期望運行。

原因：

這種攻擊方式不需要同時寫入和執行程式碼，因此可以繞過 W^X 保護策略。攻擊者利用漏洞程式中已有的函數進行攻擊，避免了資料執行保護策略對程式碼注入和執行的防護。

三、W^X 保護的編譯方式

- **編譯指令介紹**

1.`-fstack-protector`：啟用堆疊保護機制。

作用:

在函數的返回地址和本地變數之間插入一個「Canary Value」。在函數返回時會檢查這個值是否被修改過，若發現修改，則程式會立即終止，防止堆疊溢出攻擊。

優點：

提高程式的安全性。

2.`-fno-stack-protector`：不啟用堆疊保護機制，不插入 Canary Value。

3.`-z execstack`：（可執行）

作用:

設置成可執行檔，使其堆疊區域可執行。（若未設置，編譯器預設為堆疊不可執行）

使用時機：

需要動態生成並執行程式碼的應用程式。

缺點：

會降低安全性。

- **實現 W^X 保護方法：**（在編譯過程中不使用`-z execstack`）

```
gcc -o safeTest Test.c
gcc -o safeTest Test.c -z noexecstack
```

- **不使用 W^X 保護方法：**

```
gcc -o unsafeTest Test.c -z execstack
```

四、實作

- **未保護：**

```
┌──(kali㉿kali)-[~/Test3]
└─$ gcc -o unsafetest  -z execstack TestBufferOF.c
TestBufferOF.c: In function 'myprivatetest':
TestBufferOF.c:25:1: warning: implicit declaration of functi
on 'system' [-Wimplicit-function-declaration]
   25 | system("/usr/bin/xeyes");
      | ^~~~~~
```

使用 checksec 看保護的設定：

```
┌──(kali㉿kali)-[~/Test3]
└─$ checksec --file=./unsafetest
[*] '/home/kali/Test3/unsafetest'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX unknown - GNU_STACK missing
    PIE:      PIE enabled
    Stack:    Executable
    RWX:      Has RWX segments
```

Arch:　amd64-64-little （架構）
→64 位元的 x86 架構，並使用 Little-Endian。

RELRO:　Partial RELRO
→全名為 read only relocation，表示部分的只讀重定位（Partial RELRO），一部分的重定位表被標記為只讀。

Stack:　No canary found （沒有找到 Stack 保護機制）
→代表未開啟 Stack 保護機制，用 Canary value 來確認有無 Buffer OverFlow。

NX:　NX unknown - GNU_STACK missing （NX 狀態未知）
→NX (No-Execute)，表示該文件沒有設置 GNU_STACK 標記，因此 checksec 無法確定 NX 保護的狀態。通常表示 NX 保護未啟用。

PIE:　PIE enabled （PIE 已啟用）
→PIE（Position Independent Executable），負責讓 .text、.data、.bss 等等 section 隨機化，有助於防止某些類型的攻擊，例如返回導向編程（ROP）。

Stack:　Executable
→表示 Stack 頁面是可執行的，攻擊者可能會利用這一點來執行惡意程式碼。

RWX:　Has RWX segments
→有 RWX segments，表示存在一些記憶體 segment 同時具有讀、寫和執行許可權。

- **有保護：**

```
┌──(kali㉿kali)-[~/Test3]
└─$ gcc -o safetest TestBufferOF.c
TestBufferOF.c: In function 'myprivatetest':
TestBufferOF.c:25:1: warning: implicit declaration of functi
on 'system' [-Wimplicit-function-declaration]
   25 | system("/usr/bin/xeyes");
      | ^~~~~~
```

使用 checksec 看保護的設定：

```
┌──(kali㉿kali)-[~/Test3]
└─$ checksec --file=./safetest
[*] '/home/kali/Test3/safetest'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Arch: amd64-64-little（64 位元 x86 架構）
RELRO: Partial RELRO（部分只讀重定位）
Stack: No canary found（沒有 Stack 保護）
NX: NX enabled（NX 保護已啟用）
→表示記憶體頁面不能同時具有寫入和執行許可權，是一種防止程式碼執行攻擊的保護機制（W^X）。
PIE: PIE enabled

名稱	用處
Arch	架構
RELRO	`read only relocation`,用來限制程序中重定向位置的可寫區域,通常 `Partial RELRO` 代表 `GOT表` 可寫
Canary	`stack` 保護機制，利用 `cookie` 來確認是否被攻擊，因為bof攻擊通常會將整個 `stack` 改寫而覆蓋掉 `cookie`
NX	No Execute(Win平台上稱之為 DEP),不能再stack上執行指令
PIE	`Position Independent Code` 或稱 `ASLR(address space layout randomization)`,記憶體地址隨機化

- **使用 SSP 與 W^X 保護機制**

```
┌──(kali㉿kali)-[~/Test3]
└─$ gcc -o pro -fstack-protector TestBufferOF.c -z noexecstack
┌──(kali㉿kali)-[~/Test3]
└─$ checksec --file=./pro
[*] '/home/kali/Test3/pro'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

檢查有無 Buffer OverFlow → 可成功防止攻擊

```
┌──(kali㉿kali)-[~/Test3]
└─$ ./pro $(cat input.in)
Wrong username and password!!!!
```

本實驗因函數 myprivatetest 為合法執行之程式碼，故仍可執行透過覆蓋返回地址而執行該函數內的指令。

五、W^X 與 SSP 比較

保護方法	W^X（Write XOR Execute）	SSP（Stack Smashing Protection）
目的	防止記憶體區域同時具備可寫和可執行權限。	防止堆疊溢位攻擊。
原理	確保記憶體區域要麼是可寫的，要麼是可執行的。	在堆疊中插入 canary 值，檢查是否被修改。
保護範圍	程式碼注入和執行攻擊。	堆疊溢位攻擊。
實現方法（編譯過程中）	不使用 -z execstack	使用 -fstack-protector
優勢	防止程式碼注入，確保記憶體安全	提高對堆疊溢位攻擊的防護能力

- **W^X 保護範圍廣泛：**
  W^X 適用於整個程序的所有記憶體區域，防止多種記憶體攻擊（程式碼注入、ROP 等），而 SSP 僅保護函數的局部堆疊幀。
- **W^X 防止執行：**
  W^X 主要防止記憶體頁面的權限誤配置，使得攻擊者無法執行惡意程式碼，而 SSP 主要是防止堆疊溢出覆蓋返回地址。
- **SSP 專注於堆疊溢出：**
  SSP 針對特定的堆疊溢出攻擊，通過運行時檢查 Canary Value 來檢測和防止攻擊。

參考資料：

W^X - 維琪百科，自由的百科全書 (wikipedia.org)
OpenBSD - 維基百科，自由的百科全書 (wikipedia.org)
Stack buffer overflow protection 學習筆記 - Stack canaries mechanism in User space - SZ Lin with Cybersecurity & Embedded Linux
https://linux.vbird.org/linux_basic/centos7/0210filepermission.php
https://ithelp.ithome.com.tw/articles/10336777?sc=rss.iron
https://blog.csdn.net/hanchaoqi/article/details/39157955
https://introspelliam.github.io/2017/09/30/linux%E7%A8%8B%E5%BA%8F%E7%9A%84%E5%B8%B8%E7%94%A8%E4%BF%9D%E6%8A%A4%E6%9C%BA%E5%88%B6/
https://ithelp.ithome.com.tw/articles/10252772