**HW3 Writeup**

**R10922043 黃政瑋**

**CTF account: cwhuang1937**

1. **fifo**
   1. 靜態分析題目

首先用IDA反編譯後，該程式會fork一個child process，而child所拿到的pid等於0，故會進入if裡面execve一個file。一張含有 文字 的圖片

自動產生的描述

* 1. 動態分析題目

直接在0x0000555555555514設breakpoint後，即可知道child執行的file path為**/tmp/khodsmeogemgoe** 。

一張含有 文字 的圖片

自動產生的描述

接著在0x0000555555555583再設一個breakpoint，並步過該指令後，再**開另外個gdb 掛上child process**。此時先繼續debug parent的程式，會發現在0x00005555555555ea處，會call mkfifo，上網查之後，parent會藉由這個檔案來當作與child溝通的管道(感覺有點類似pipe的概念)。

一張含有 文字 的圖片

自動產生的描述

* 1. 對child執行的檔案做靜態分析

接著再用IDA對/tmp/khodsmeogemgoe做分析，果真發現該程式會**從/tmp/bnpkevsekfpk3/aw3movsdirnqw這個檔案讀資料到buf，並試著去執行他**。

* 1. 繼續做動態分析

Parent繼續往後debug時，當open /tmp/khodsmeogemgoe時，此時會卡住，接著換對child按一次步過後，此時parent即可繼續執行並寫入東西到/tmp/bnpkevsekfpk3/aw3movsdirnqw。

繼續對child做debug後，發現他讀parent寫進去的data到buf後，即可成功看到FLAG了，此時child去執行該function就會當掉，這邊只能手動kill掉才能終止程式。

一張含有 文字 的圖片

自動產生的描述

**速解:**

這題我最一開始只是單純parent執行下去，這時會看到child的pid，直接用gdb掛上去該pid即可在stack拿到flag了。

一張含有 文字 的圖片

自動產生的描述

1. **giveUflag**
   1. 分析題目

直接點開程式後發現會整個卡住。接著用IDA反編譯後，從main開始往後點幾個subroutine，發現在sub\_4015F3藏有一些不尋常的code(puts的網址點開是一些梗圖)，故決定先從這邊開始逆。

一張含有 文字 的圖片

自動產生的描述

* 1. 動態debug

接著在sub\_4015F3處設breakpoint(00000000004015f3)，接著不斷F8步過，會發現在000000000040179B處會call rax，其中rax為**kernel32.Sleep**。而sleep的時間為604800000ms，恰好符合梗圖的7天。

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

* 1. 改RIP來跳過kernel32.Sleep

重新執行後，這次直接將breakpoint設在call rax的地方(000000000040179B)，到該breakpoint後，搭配IDA查看，將RIP設成00000000004017DD後，即可成功跳過sleep的部分，接著直接用ctrl+F9跳到ret，即可成功看到flag顯示在視窗上。

一張含有 文字 的圖片

自動產生的描述

**題外話:**

在動態debug前，從sub\_401550有發現他藉由爬PEB來拿一些API，並將其回傳出去，然後當作參數傳給上述可能藏有code的sub\_4015F3。但動態跑下去後發現call rax那邊就直接看到rax是什麼了，故這邊並沒有手動去爬PEB，即可成功知道rax為kernel32.Sleep。

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

1. **nani**
   1. 解parker

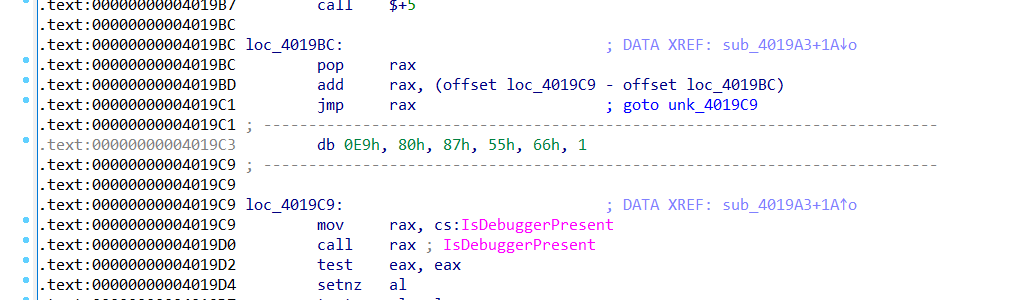
用DIE查看，會發現用**UPX3.96加殼過**，上網找對應的upx做decompress，即可unpack。一張含有 文字, 監視器, 螢幕擷取畫面, 螢幕 的圖片

自動產生的描述一張含有 文字 的圖片

自動產生的描述

* 1. 解Anti-Disassembly

查看原本的graph-view發現怪怪的，仔細看text-view會發現似乎有被塞trash code導致無法成功反編譯，**故這邊手動undefined後，把往後的一行也塞進來trash code**即可成功反編譯。



* 1. 解Anti-Debug

從上面反編譯後，發現它會偵測Debugger，若被偵測到會直接exit，因此這邊**將x64dbg裝上ScyllaHide**，並將options全開即可繞過。

一張含有 文字 的圖片

自動產生的描述

* 1. 解Anti-VM

執行完cpuid後，會將字串存在RBX、RCX、RDX裡面，用python解析出來後果真是他的6個blacklist之一，因此這邊**直接用RIP跳到0x0000000000401990**，即可成功繞過anti-VM。

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

* 1. 逆exception handler

繞過前面的VM後，從IDA會看到有一個exception handler，因此x86dbg在sub\_4016FB的地方設斷點，接著繼續用IDA靜態分析看handler，會發現該handler會對**byte\_4015AF**做一些操作。

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

x64dbg跟下去後，byte\_4015AF仍無法看到FLAG，推測byte\_4015AF可能是一段code，這邊用**RIP直接設成4015AF跳過去**，執行下去即可拿到FLAG。

一張含有 文字 的圖片

自動產生的描述