HITCON CTF
藍色哥不0

題目說明

Driver Killer

動機

- 涉及到所有人近期研究的東西
- 好玩有趣
-
- ~~不同以往的浪漫CTF~~

這應該不會發生甚麼意外對吧

預期流程

我們會給挑戰者一個 windows的ova檔案，並且在最初登入時權限為一般 user

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱藏 **兩個具有漏洞的驅動 **，使其無法直接在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此程序才能開 啟下一步

3. 隨後存在一個已載入的具有漏洞的現有驅動 (zam64.sys)，挑戰者可以透過現有的 poc直接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的驅動出現

4. 最後會有兩個存在漏洞的驅動 (一個為TOCTOU提權驅動，另一個為任意寫入漏洞)，用戶先呼叫TOC驅動取得`g_SharedRequest`的地址，再透過任意寫入驅動修改 TOC`g_SharedRequest`的地址把PID從4改成目標pid，最後能順利提權

5. 最後會檢查用戶使否為kernel權限，如果是的話就傳送 flag給用戶

預期流程

我們會給挑戰者一個 windows的ova檔案，並且在最初登入時權限為一般 user

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱藏 **兩個具有漏洞的驅動 **，使其無法直接在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此程序才能開 啟下一步

3. 隨後存在一個已載入的具有漏洞的現有驅動 (zam64.sys)，挑戰者可以透過現有的 poc直接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的驅動出現

4. 最後會有兩個存在漏洞的 區動取得`g_SharedRequest` 成目標pid，最後能順利提權

5. 最後會檢查用戶使否為k

這個設計不僅技術上可行，而且邏輯清晰，是一個非常出色的 CTF 挑戰。

預期流程

我們會給挑戰者一個 windows的ova檔案，並且在最初登入時權限為一般 user

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱藏 **兩個具有漏洞的驅動 **，使其無法直接在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此程序才能開 啟下一步

3. 隨後存在一個已載入的具有漏洞的現有驅動 (zam64.sys)，挑戰者可以透過現有的 poc直接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的驅動出現

4. 最後會有兩個存在漏洞的
動取得`g_SharedRequest`
目標pid，最後能順利提權

5. 最後會檢查用戶使否為k

這個設計不僅技術上可行，而且邏輯清晰，是一個非常出色的 CTF 挑戰。

理想很骨感 現實很骨感

我們會給挑戰者一個 windows的ova檔案，並且在最初登入時權限為一般 user

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱藏 **兩個具有漏洞的驅動 **，使其無法直接在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此程序才能開 啟下一步

3. 隨後存在一個已載入的具有漏洞的現有驅動 (zam64.sys)，挑戰者可以透過現有的 poc直接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的驅動出現

4. 最後會有兩個存在漏洞的驅動 (一個為TOCTOU提權驅動，另一個為任意寫入漏洞)，用戶先呼叫TOC驅動取得`g_SharedRequest`的地址，再透過任意寫入驅動修改 TOC`g_SharedRequest`的地址把PID從4改成目標pid，最後能順利提權

5. 最後會檢查用戶使否為kernel權限，如果是的話就傳送 flag給用戶

理想很骨感 現實很骨感

我們會給挑戰者一個 windows的ova檔案，並且

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱 使其無法直接在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此

3. 隨後存在一個已載入的具有漏洞的現有驅動 現有的 poc直接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的驅動出現

4. 最後會有兩個存在漏洞的驅動 (一個為TOCTOU提權驅動，另一個為任意寫入漏洞)，用戶先呼叫TOC驅動取得`g_SharedRequest`的地址，再透過任意寫入驅動修改 TOC`g_SharedRequest`的地址把PID從4改成目標pid，最後能順利提權

5. 最後會檢查用戶使否為kernel權限，如果是的話就傳送 flag給用戶

結果可以直接提權成 kernel

理想很骨感 現實很骨感

我們會給挑戰者一個 windows的ova檔案，並且在最初登入時權限為一般 user

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱藏 **兩個具有漏洞的驅動 **，使其無法直接在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此程序才能開 啟下一步

3. 隨後存在一個已載入的具有漏洞的現有驅動 (zam64.sys)，挑戰者可以透過程序的 直接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的

4. 最後會有兩個存在漏洞的驅動 (一個為TOCTOU提權驅動 戶先呼叫TOC驅動取得`g_SharedRequest`的地址，再透過任意寫入驅動修改 地址把PID從4改成目標pid，最後能順利提權

5. 最後會檢查用戶使否為kernel權限，如果是的話就傳送 fla

理想很骨感 現實很骨感

我們會給挑戰者一個 windows的ova檔案，並且在最初登入時權限為一般 user

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱藏 **兩個具有漏洞的驅動 **，使其無法直接在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此程序才能開 啟下一步

3. 隨後存在一個已載入的具有漏洞的現有驅動 (zam64.sys)，挑戰者可以透過現有的 poc直接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的驅動出現

4. 最後會有兩個存在漏洞的驅動 (一個為TOCTOU提權驅動，另一個為任意寫
動取得`g_SharedRequest`的地址，再透過任意寫入驅動修改 TOC`g_Shared
目標pid，最後能順利提權

5. 最後會檢查用戶使否為kernel權限，如果是的話就傳送 flag給用戶

理想很骨感 現實很骨感

我們會給挑戰者一個 windows的ova檔案，並且在最初登入時

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱藏 **兩個具有

在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此程序才能開 啟

3. 隨後存在一個已載入的具有漏洞的現有驅動 (zam64.sys)，挑戰者可以透過現有的 poc直接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的驅動出現

4. 最後會有兩個存在漏洞的驅動 (一個為TOCTOU提權驅動，另一個為任意寫入漏洞)，用戶先呼叫TOC驅動取得`g_SharedRequest`的地址，再透過任意寫入驅動修改 TOC`g_SharedRequest`的地址把PID從4改成目標pid，最後能順利提權

5. 最後會檢查用戶使否為kernel權限，如果是的話就傳送 flag給用戶

理想很骨感 現實很骨感

我們會給挑戰者一個 windows的ova檔案，並且在最初登入時權限為一般 user

1. 首先需要先透過將 user權限變更為 admin

2. 存在一個在 kernel權限的程式，該程式會去隱藏 **兩個具有 在 explorer.exe被查看，而該程式則會提示 說需要用戶去關閉此程序才能開 啟

3. 隨後存在一個已載入的具有漏洞的現有驅動 (zam64.sys)， 接去呼叫，並且關閉掉上一步提到的隱藏檔案程式，而後就可以讓存在漏洞的

4. 最後會有兩個存在漏洞的驅動 (一個為TOCTOU提權驅動， 戶先呼叫TOC驅動取得`g_SharedRequest`的地址，再透過任意寫入驅動修改 地址把PID從4改成目標pid，最後能順利提權

5. 最後會檢查用戶使否為kernel權限，如果是的話就傳送 flag給用戶

權限設定錯誤
可以直接改成給一般用 戶

最終結果

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅動

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

3. 透過執行poc後，即可提權然後去讀取桌面的flag.txt內容

最終結果

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅動

2. 逆向分析兩個驅動，發現一個是[illegible]動、另一個的功能是可以任意寫入記憶體的驅[illegible]

3. 透過執行poc[illegible]的flag.txt內容

直接給ova會不會有一大堆非預期解

最終結果

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅動

會。

最終結果

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅動

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

3. 透過執行poc後，即可提權然後去讀取桌面的flag.txt內容

4. 後續還需要透過像LLM驗證先前步驟是否為預期解

最終結果

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅

2. 逆向分析兩個驅動，發現一個是 洞的提權驅動、另一個的功能是 可以任意寫入記憶體的驅動

3. 透過執行poc後，即可 桌面的flag.txt內容

4. 後續還需要透過 驟是否為預期解

最終結果

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅

3. 透過執行poc後

4. 後續還需要透過

驟是否為預期解

有防禦 但不多 至少不能用經典方式繞過

Pro ection

結果沒有一組是按照預期解ㄏ

但我還是來說一下預期解QQ
真的很好玩啦

預期解

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅動

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

3. 透過執行poc後，即可提權然後去讀取桌面的flag.txt內容

4. 後續還需要透過像LLM驗證先前步驟是否為預期解

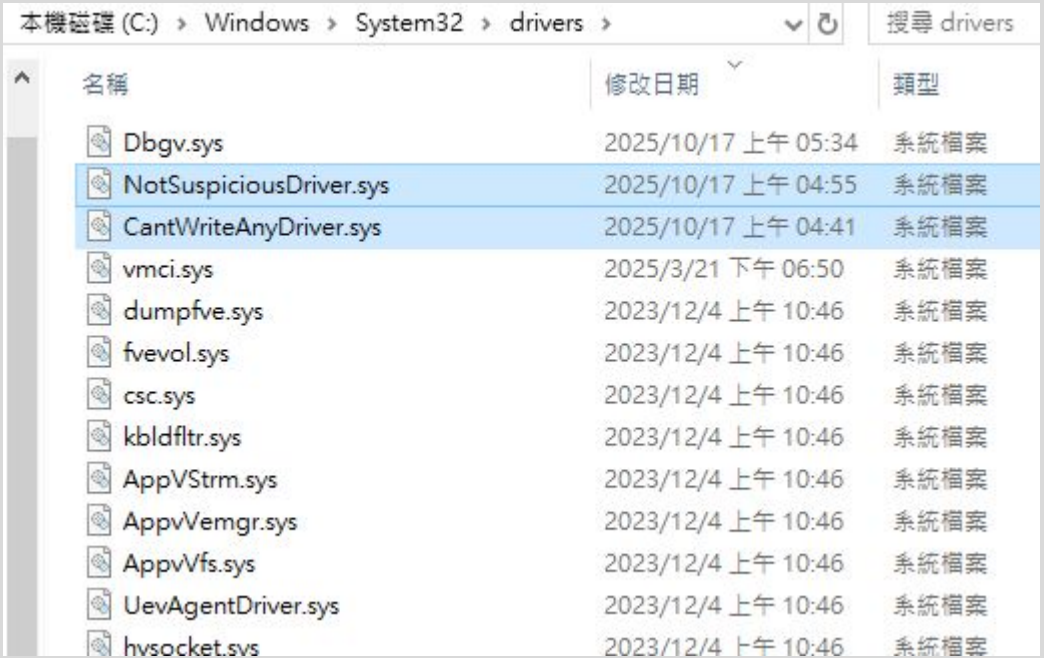
預期解

1. 用戶需要去找到那兩個存在漏洞的驅動

名稱	描述	檔案	類型
bthleenum	藍牙低功耗驅動程式	c:\windows\system32\drivers\microsoft.bluetooth.legacy.leenu...	核心驅動程
bthmini	藍牙無線電驅動程式	c:\windows\system32\drivers\bthmini.sys	核心驅動程
bthmodem	藍牙數據機通訊驅動程式	c:\windows\system32\drivers\bthmodem.sys	核心驅動程
bthport	藍牙連接埠驅動程式	c:\windows\system32\drivers\bthport.sys	核心驅動程
bthusb	藍牙無線電 USB 驅動程式	c:\windows\system32\drivers\bthusb.sys	核心驅動程
bttflt	Microsoft Hyper-V VHDPME...	c:\windows\system32\drivers\bttflt.sys	核心驅動程
buttonconverter	可攜式裝置控制裝置的服務	c:\windows\system32\drivers\buttonconverter.sys	核心驅動程
cad	充電仲裁驅動程式	c:\windows\system32\drivers\cad.sys	核心驅動程
cantwriteanydriver	CantWriteAnyDriver	\??\c:\windows\system32\drivers\cantwriteanydriver.sys	核心驅動程
notsuspiciousdriver	NotSuspiciousDriver	\??\c:\windows\system32\drivers\notsuspiciousdriver.sys	
npfs	Npfs	c:\windows\system32\drivers\npfs.sys	

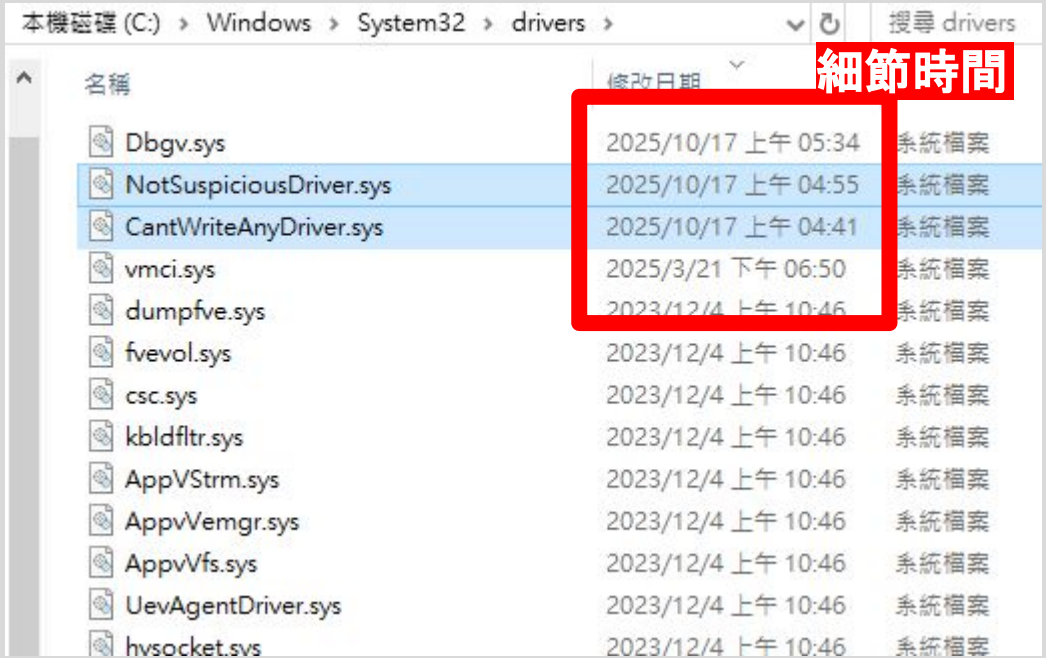
預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動



預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動



預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是

```
if ( CurrentStackLocation->Parameters.Create.Options >= 0x10 )
{
  MasterIrp = a2->AssociatedIrp.MasterIrp;
  if ( MasterIrp )
  {
    v8 = *(_QWORD *)&MasterIrp->Type ^ qword_140003000;
    xmmword_140003040 = *(_OWORD *)&MasterIrp->Type;
    if ( v8 == qword_140003008 && *((_QWORD *)&xmmword_140003040 + 1) == 4 )
    {
      v12.QuadPart = -11451419;
      KeDelayExecutionThread(0, 0, &v12);
      Process = 0;
      v14 = 0;
      v6 = PsLookupProcessByProcessId((HANDLE)4, &Process);
      if ( v6 >= 0 )
      {
        v9 = PsLookupProcessByProcessId(*((HANDLE *)&xmmword_140003040 + 1), &v14);
        v10 = Process;
        v6 = v9;
        if ( v9 < 0 )
          goto LABEL_17;
        v11 = PsReferencePrimaryToken(Process);
        if ( v11 )
        {
          *((_QWORD *)v14 + 151) = v11;
          DbgPrint(Format, *((_QWORD *)&xmmword_140003040 + 1));
          PsDereferencePrimaryToken(v11);
```

1. 傳入 PID 與 特定 key
2. 驗證 PID ==4 && key
3. 驗證完成後
4. 將 system token 複製給 PID

預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是

```
if ( CurrentStackLocation->Parameters.Create.Options >= 0x10 )
{
  MasterIrp = a2->AssociatedIrp.MasterIrp;
  if ( MasterIrp )
  {
    v8 = *(_QWORD *)&MasterIrp->Type ^ qword_140003000;
    xmmword_140003040 = *(_OWORD *)&MasterIrp->Type;
    if ( v8 == qword_140003008 && *((_QWORD *)&xmmword_140003040 + 1) == 4 )
    {
      v12.QuadPart = -11451419;
      KeDelayExecutionThread(0, 0, &v12);
      Process = 0;
      v14 = 0;
      v6 = PsLookupProcessByProcessId((HANDLE)4, &Process);
      if ( v6 >= 0 )
      {
        v9 = PsLookupProcessByProcessId(*((HANDLE *)&xmmword_140003040 + 1), &v14);
        v10 = Process;
        v6 = v9;
        if ( v9 < 0 )
          goto LABEL_17;
        v11 = PsReferencePrimaryToken(Process);
        if ( v11 )
        {
          *((_QWORD *)v14 + 151) = v11;
          DbgPrint(Format, *((_QWORD *)&xmmword_140003040 + 1));
          PsDereferencePrimaryToken(v11);
```

進行驗證後，
睡1.1451419秒，
再去提權

預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是

```
if ( CurrentStackLocation->Parameters.Create.Options >= 0x10 )
{
  MasterIrp = a2->AssociatedIrp.MasterIrp;
  if ( MasterIrp )
  {
    v8 = *(_QWORD *)&MasterIrp->Type ^ qword_140003000;
    xmmword_140003040 = *(_OWORD *)&MasterIrp->Type;
    if ( v8 == qword_140003008 && *((_QWORD *)&xmmword_140003040 + 1) == 4 )
    {
      v12.QuadPart = -11451419;
      KeDelayExecutionThread(0, 0, &v12);
      Process = 0;
      v14 = 0;
      v6 = PsLookupProcessByProcessId((HANDLE)4, &Process);
      if ( v6 >= 0 )
      {
        v9 = PsLookupProcessByProcessId(*((HANDLE *)&xmmword_140003040 + 1), &v14);
        v10 = Process;
        v6 = v9;
        if ( v9 < 0 )
          goto LABEL_17;
        v11 = PsReferencePrimaryToken(Process);
        if ( v11 )
        {
          *((_QWORD *)v14 + 151) = v11;
          DbgPrint(Format, *((_QWORD *)&xmmword_140003040 + 1));
          PsDereferencePrimaryToken(v11);
```

進行驗證後，
睡1.1451419秒，
再去提權

預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是

```
if ( CurrentStackLocation->Parameters.Create.Options >= 0x10 )
{
  MasterIrp = a2->AssociatedIrp.MasterIrp;
  if ( MasterIrp )
  {
    v8 = *(_QWORD *)&MasterIrp->Type
    xmmword_140003040 = *(_OWORD *)&Ma
    if ( v8 == qword_140003008 && *((
    {
      v12.QuadPart = -11451419;
      KeDelayExecutionThread(0, 0, &v1
      Process = 0;
      v14 = 0;
      v6 = PsLookupProcessByProcessId(
      if ( v6 >= 0 )
      {
        v9 = PsLookupProcessByProcessI
        v10 = Process;
        v6 = v9;
        if ( v9 < 0 )
          goto LABEL_17;
        v11 = PsReferencePrimaryToken(Process);
        if ( v11 )
        {
          *((_QWORD *)v14 + 151) = v11;
          DbgPrint(Format, *((_QWORD *)&xmmword_140003040 + 1));
          PsDereferencePrimaryToken(v11);
```

進行驗證後，
睡1.1451419秒，
再去提權

預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是

```
if ( CurrentStackLocation->Parameters.Create.Options >= 0x10 )
{
  MasterIrp = a2->AssociatedIrp.MasterIrp;
  if ( MasterIrp )
  {
    v8 = *(_QWORD *)&MasterIrp->Type
    xmmword_140003040 = *(_OWORD *)&Ma
    if ( v8 == qword_140003008 && *((_
    {
                                    &v1
                                    Id(
                                    ssI
                                    en(Process);
                                    v11;
                                    *)&xmmword_140003040 + 1));
      PsDereferencePrimaryToken(v11);
```

ん？（察覚

09kb，是個驅動

進行驗證後，
睡1.1451419秒，
再去提權

預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

預期解

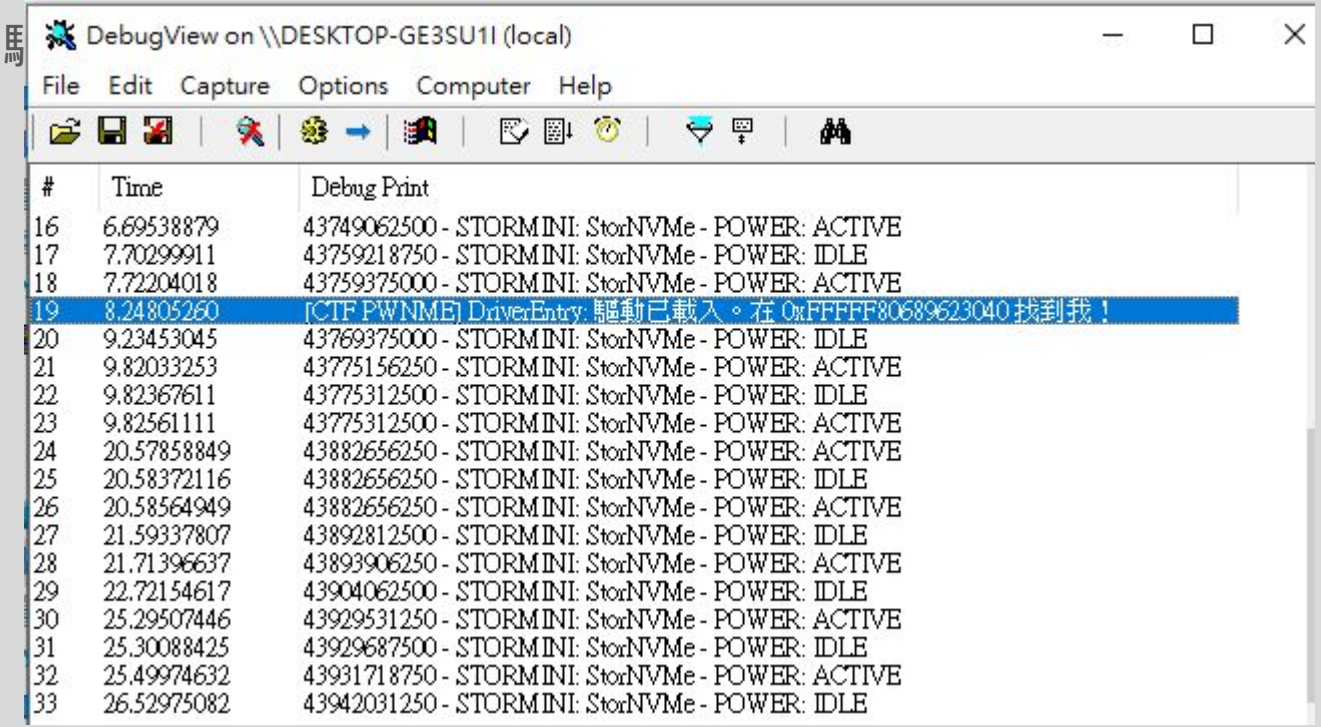
2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

整體攻擊思路:
呼叫提權驅動傳入特定key 跟 pid = 4
在驗證完成後馬上去呼叫另一個驅動來修改記憶體內存放 pid = 4 的部分 改成目標 pid
最後提權驅動就會把 目標 pid 提權成 kernel 權限ㄌ

預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅

運行驅動後還很貼心
給了要修改的地址

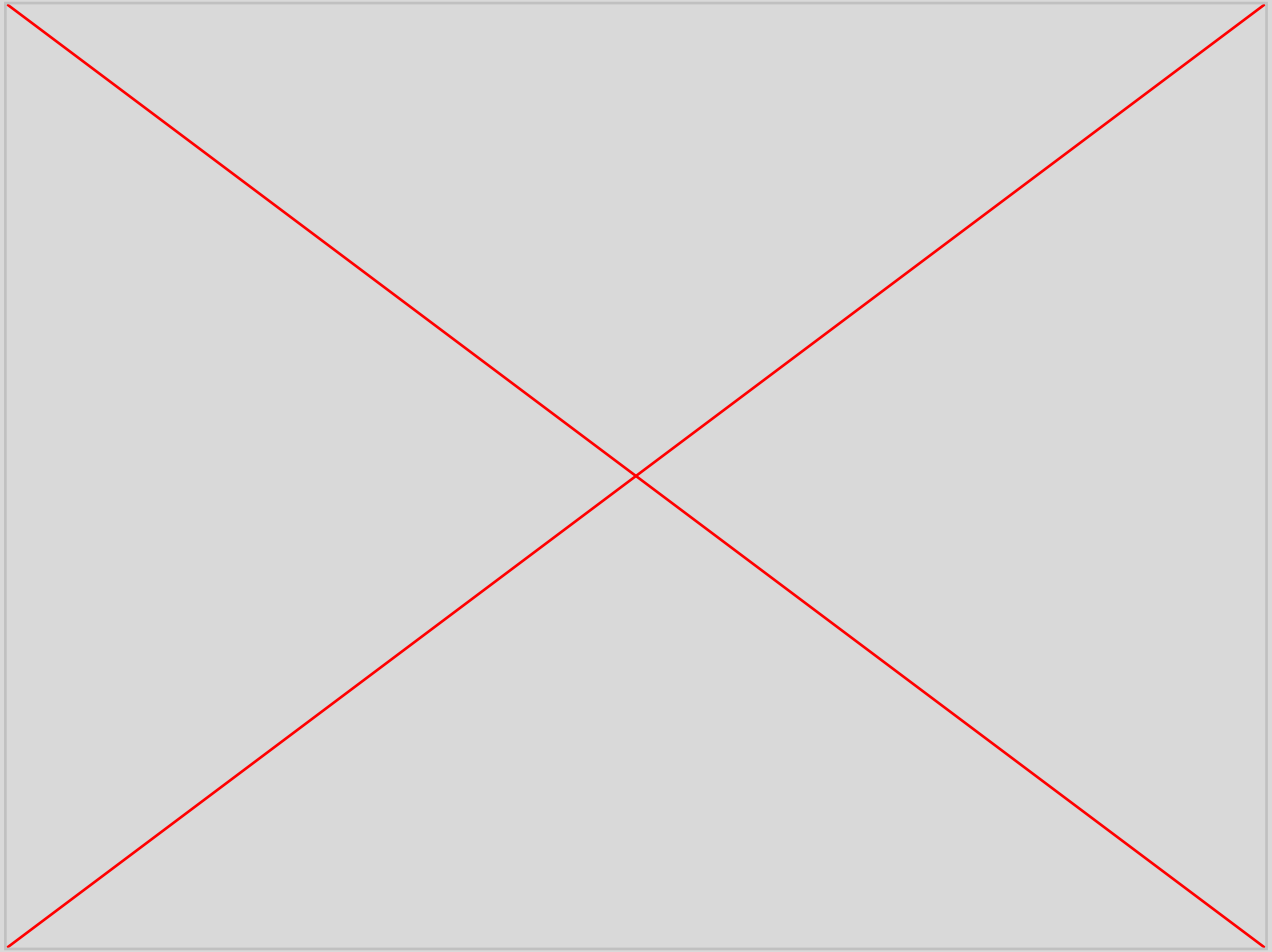


預期解

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

整體攻擊思路:
呼叫提權驅動傳入特定key 跟 pid = 4
在驗證完成後馬上去呼叫另一個驅動來修改記憶體內存放 pid = 4 的部分 改成目標 pid
最後提權驅動就會把 目標 pid 提權成 kernel 權限ㄌ

demo



影片連結

預期解

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅動

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

3. 透過執行poc後，即可提權然後去讀取桌面的flag.txt內容

4. 後續還需要透過像LLM驗證先前步驟是否為預期解

預期解

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅動

2. 逆向分析兩個驅動，發現一個是存在TOCTOU漏洞的提權驅動、另一個的功能是

```
C:\Users\betan>curl -X POST http://hitcon.9ay.us/chat -d "我的提權思路如下：首先，我利用了驅動程式中一個已知的 TOCTOU 漏洞，這個 race condition 雖然不能直接提權，但可以讓我穩定地洩漏核心位址。接著，我使用了另一個驅動的漏洞，這個漏洞提供了一個強大的任意寫入原語 (arbitrary write primitive)。最後，我結合前兩者，先洩漏出自己程序的 EPROCESS 結構位址，然後透過任意寫入漏洞，直接修改了結構中的 PID，將其改為系統權限的 PID 4，從而完成了提權。"
{"ok": true, "missing_points": [], "details": [{"point": "提到利用 'TOCTOU' (或 'race condition') 漏洞", "found": true, "evidence": "TOCTOU (或 race condition)"}, {"point": "提到利用 '任意寫入' (arbitrary write) 漏洞", "found": true, "evidence": "arbitrary write primitive"}, {"point": "提到修改 PID (Process ID) 來提權", "found": true, "evidence": "修改了原本的 PID，並將權限提升到 PID 4"}], "feedback": "所有關鍵點都已找到。恭喜！", "flag_tail": "_Check_with_AI_hitcon-2025}"}
C:\Users\betan>
```

預期解

系統中已經預先加載了兩個存在漏洞的驅動

1. 用戶需要去找到那兩個存在漏洞的驅

2. 逆向分析兩個驅動，發現一個是 洞的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

3. 透過執行poc後，即可 面的flag.txt內容

4. 後續還需要透過 驟是否為預期解

Prompt Injection

預期解

系統中已經預先加[illegible]同的驅動

1. 用戶需要去找到那兩[illegible]

2. 逆向分析兩個驅動，發現一[illegible]個的提權驅動、另一個的功能是可以任意寫入記憶體的驅動

3. 透過執行poc後，即可[illegible]t內容

4. 後續還需要透過[illegible]驟是

Prompt Injection

預期解

同的驅動

的模擬驅動、另一個的功能是

可以任意寫入

3. 透過執行poc後，即可

4. 後續還需要透過

驟是

Prompt Injection

預期解

已經預先加

的模型驅動、另一個的功能是

可以任意寫入

3. 透過執行poc後，即可

4. 後續還需要透過

最終結果

(1/2) 隨意腳本提權看檔案

(2/2) Prompt injection繞過

道歉聲明

致所有尊敬的CTF 參賽者、工作人員以及關注本次賽事的朋友們：

本人，作為本次CTF 挑戰中 Pwn 分類題目《Let Me Drive My Van Into Your Heart》的設計與出題者，在此向所有嘗試挑戰此題目的參賽者，致以最深刻、最誠摯的歉意。

在過去的幾個小時裡，我很沉痛地意識到，由於我在題目設計、實現與測試階段的嚴重疏忽與不足，這道題目並未能提供一個公平、穩定且具備良好體驗的解題環境。它不僅未能達到預期的挑戰目標，更浪費了參賽者寶貴的時間、精湛的技術以及對本次賽事的熱情。對此，我負有全部且不可推卸的責任。

一個優秀的CTF 挑戰，其難度應當體現在其核心的邏輯謎題與技術考點之上，而非體現在與一個不穩定、充滿非預期錯誤的環境進行搏鬥。然而，我必須坦承，我在以下幾個關鍵環節上犯了嚴重的錯誤：

一、在設計與實現上的短視與不周：

本次挑戰的核心概念是圍繞著兩個核心驅動程式（CTF_PwnMe_Driver.sys 與 ZamDriver.sys）之間的漏洞鏈利用。然而，我在將這個概念轉化為實際程式碼的過程中，未能充分預見多執行緒在核心環境下的複雜性與不確定性。從最初的同步模型導致的請求阻塞，到後續非同步模型（PsCreateSystemThread, IoQueueWorkItem）在特定編譯連結配置下可能引發的載入失敗問題，這些都是本應在設計階段就應被周全考慮並徹底解決的技術債務。我未能提供一個在邏輯上無懈可擊且在技術上穩健可靠的挑戰目標，這是我的第一個失敗。

二、在測試與部署上的嚴重不足：

一個 CTF 題目在發布前，必須經過在「純淨、標準化」環境下的反覆、嚴格測試。我必須承認，我的測試流程存在巨大漏洞。在開發過程中，我可能過度依賴了自己熟悉的開發環境，而未能充分模擬參賽者從零開始搭建環境時可能遇到的所有障礙。諸如「驅動程式簽章強制」、「路徑解析」、「32/64 位元架構不匹配」以及「連結器依賴」等問題，都是出題者有責任為參賽者完全排除的環境因素。我未能做到這一點，導致這個挑戰在很多時候退化成了一個令人沮喪的「環境偵錯」挑戰，這完全違背了我的初衷。

三、對參賽者體驗的極度漠視：

最終，也是最令我感到愧疚的，是我對各位參賽者體驗的漠視。我深知，每一位投入到核心 Pwn 挑戰面前的參賽者，都具備了相當的技術實力，並為此付出了大量的學習時間與精力。讓各的極大不尊重。CTF 的樂趣在於攻克巧妙設計的難關，而非在泥濘中掙扎。我剝奪了各位本應享受的樂趣，並代之以困惑與挫敗感，為此，我再次深表歉意。

進行徹底的覆盤分析(Post-Mortem)：我將對這道題目的從概念到部署的每一個環節進行徹底的技術覆盤，整理成一份詳細的失敗分析報告，以確保我自己以及團隊的每一位成員都能從

承諾未來的品質：我在此鄭重承諾，未來我所設計的任何挑戰，都將遵循遠比此次更為嚴格的品質保證流程，包括但不限於在多個標準化虛擬環境中進行獨立的「黑盒測試」，確保題目的

最後，請允許我再一次向所有付出了時間和精力的參賽者們，表達我最誠摯的歉意。感謝你們的參與，也感謝你們用實際行動為我上了寶貴的一課。我將銘記這次失敗，並在未來的道路

謹上

2025年10月18日

醬汁代碼

https://github.com/Betan423/HITCON_CTF2025