**《恶意代码分析》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **20级信息安全01班** | | | **姓名** | **蔡欣彤** |
| **实验题目** | **感染目标解析** | | | | | |
| **实验时间** | **2023.5.21** | | **实验地点** | **硬件实验室** | | |
| **实验成绩** |  | | **实验性质** | **☑验证性 □设计性 □综合性** | | |
| 教师评价：  ☑算法/实验过程正确； ☑源程序/实验内容提交； ☑程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； ☑报告规范；  其他：  评价教师签名： 杨吉云 | | | | | | |
| 1. 实验目的 2. 验证感染目标解析过程 3. 验证免疫过程 | | | | | | |
| 1. 实验项目内容 2. 利用工具软件实现PE文件节表免疫、节间免疫。 3. 编程实现：在实验三的基础上实现节表免疫、节间免疫。 | | | | | | |
| 三、实验过程或算法（源程序）  1.（1）用010editor打开.exe文件，将PE文件头以及节表项均往后挪，改变MZ头3C位置的值。  修改前：    修改后：      （2）将节表项往后移动，将optionalheader的大小变大  修改前：    修改后：      （3）将每个节表项的virtualsize置为sizeofrawdata  修改前：    修改后：    2.编程实现  （1）节表免疫，方法一  #include <windows.h>  #include <winioctl.h> //DDK驱动开发与控制  #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #include <stdint.h>  #include<iostream>  #include <string>  #include <sstream>  using namespace std;  //节表免疫，方法一  string decimal\_to\_hex\_str(int decimal) {  stringstream ss;  ss << hex << decimal;  return ss.str();  }  int main() {  // LPCWSTR path = "P:\\exp1.exe";  HANDLE hFile;  HANDLE hMapping;  LPVOID pMapping;  hFile = CreateFile("P:\\exp1.exe", GENERIC\_READ|GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING,  FILE\_ATTRIBUTE\_NORMAL, NULL);  if (INVALID\_HANDLE\_VALUE == hFile) {  return 0;  }  //将PE文件映射到内存  hMapping = CreateFileMapping(hFile, NULL, PAGE\_READONLY, 0, 0, 0);  if (!hMapping) {  return 0;  }  pMapping = MapViewOfFile(hMapping, FILE\_MAP\_READ, 0, 0, 0);//返回的是map的开始地址  if (!pMapping) {  return 0;  }  PIMAGE\_DOS\_HEADER dosheader;  dosheader = (PIMAGE\_DOS\_HEADER)pMapping;  if (dosheader->e\_magic != IMAGE\_DOS\_SIGNATURE) {  cout << "无效的PE文件" << endl;  return 0;  }  PIMAGE\_NT\_HEADERS nt\_header;  nt\_header = (PIMAGE\_NT\_HEADERS)((BYTE\*)pMapping + dosheader->e\_lfanew);  if (nt\_header->Signature != IMAGE\_NT\_SIGNATURE) {  cout << "无效的PE文件" << endl;  return 0;  }  PIMAGE\_SECTION\_HEADER section\_header;  section\_header = IMAGE\_FIRST\_SECTION(nt\_header);  for (int i = 0; i < nt\_header->FileHeader.NumberOfSections;  i++, section\_header++) {  cout << section\_header->Name << "\t" << section\_header->Misc.VirtualSize  << "\t\t" << section\_header->VirtualAddress << "\t\t" <<  section\_header->SizeOfRawData << "\t\t"  << section\_header->PointerToRawData << "\t\t\t" << section\_header->Characteristics << endl;  }  DWORD el\_anew = dosheader->e\_lfanew;  DWORD byteofthis = 24 + nt\_header->FileHeader.SizeOfOptionalHeader + 40 \* nt\_header->FileHeader.NumberOfSections;  // cout<<"SizeOfOptionalHeader:"<<nt\_header->FileHeader.SizeOfOptionalHeader<<endl;  // cout<<"sizeof(section\_header):"<<sizeof(section\_header)<<endl;  // cout<<"nt\_header->FileHeader.NumberOfSections:"<<nt\_header->FileHeader.NumberOfSections<<endl;  cout<<"byteofthis:"<<byteofthis<<endl;  UINT8\* Content = new UINT8[byteofthis];  memcpy(Content, (UINT8\*)pMapping + el\_anew, byteofthis);  int flag = 0;  for (size\_t i = 0; i < byteofthis; ++i)  {  unsigned char c = Content[i]; // must use unsigned char to print >128 value  flag++;  if (c < 16)  printf("0%x ", c);  else  printf("%x ", c);  if ((flag) % 16 == 0)  printf("\n");  }  //el\_anew的位置0x3C,大小为uint32\_t  //NT\_header+optionalHeader+sectionHeader的位置  // uint32\_t el\_anew = dosheader->e\_lfanew;  cout << "el\_fanew:" << el\_anew << endl;  //大小=18H+fileHeader.sizeofoptionalheader+sizeof(section\_header)\* nt\_header->FileHeader.NumberOfSections  // uint32\_t byteofthis=24+nt\_header->fileHeader.sizeofoptionalheader+sizeof(section\_header)\* nt\_header->FileHeader.NumberOfSections;  //转移的位置：1023，大小相同，  //59336704  OVERLAPPED overLap = { 0 };  overLap.Offset = 1024-byteofthis;  DWORD writeByte;  DWORD readsize = 0;  LARGE\_INTEGER offset;//long long signed  offset.QuadPart = (ULONGLONG)0 \* (ULONGLONG)512;//0  SetFilePointer(hFile, offset.LowPart, &offset.HighPart, FILE\_BEGIN);  if (!WriteFile(hFile, Content, byteofthis, &writeByte, &overLap)) {  cout << dec << "error code" << GetLastError() << endl;  }  //修改EL\_anew  UINT8 data[] = { 0x90, 0x02, 0x00, 0x00 };  DWORD el\_anewweizhi = 0x3c;  DWORD byteofthis1 = 4;  OVERLAPPED overLap1 = { 0 };  overLap1.Offset = 60;  // DWORD writeByte;  readsize = 0;  // LARGE\_INTEGER offset;//long long signed  offset.QuadPart = (ULONGLONG)0 \* (ULONGLONG)512;//0  SetFilePointer(hFile, offset.LowPart, &offset.HighPart, FILE\_BEGIN);  if (!WriteFile(hFile, data, byteofthis1, &writeByte, &overLap1)) {  cout << dec << "error code" << GetLastError() << endl;  }  CloseHandle(hFile);  }  （2）节表免疫，第二种  #include <windows.h>  #include <winioctl.h> //DDK驱动开发与控制  #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #include <stdint.h>  #include<iostream>  #include <string>  #include <sstream>  using namespace std;  //节表免疫，方法一  string decimal\_to\_hex\_str(int decimal) {  stringstream ss;  ss << hex << decimal;  return ss.str();  }  int main() {  // LPCWSTR path = "P:\\exp1.exe";  HANDLE hFile;  HANDLE hMapping;  LPVOID pMapping;  hFile = CreateFile("P:\\exp1.exe", GENERIC\_READ|GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING,  FILE\_ATTRIBUTE\_NORMAL, NULL);  if (INVALID\_HANDLE\_VALUE == hFile) {  return 0;  }  //将PE文件映射到内存  hMapping = CreateFileMapping(hFile, NULL, PAGE\_READONLY, 0, 0, 0);  if (!hMapping) {  return 0;  }  pMapping = MapViewOfFile(hMapping, FILE\_MAP\_READ, 0, 0, 0);//返回的是map的开始地址  if (!pMapping) {  return 0;  }  PIMAGE\_DOS\_HEADER dosheader;  dosheader = (PIMAGE\_DOS\_HEADER)pMapping;  if (dosheader->e\_magic != IMAGE\_DOS\_SIGNATURE) {  cout << "无效的PE文件" << endl;  return 0;  }  PIMAGE\_NT\_HEADERS nt\_header;  nt\_header = (PIMAGE\_NT\_HEADERS)((BYTE\*)pMapping + dosheader->e\_lfanew);  if (nt\_header->Signature != IMAGE\_NT\_SIGNATURE) {  cout << "无效的PE文件" << endl;  return 0;  }  PIMAGE\_SECTION\_HEADER section\_header;  section\_header = IMAGE\_FIRST\_SECTION(nt\_header);  for (int i = 0; i < nt\_header->FileHeader.NumberOfSections;  i++, section\_header++) {  cout << section\_header->Name << "\t" << section\_header->Misc.VirtualSize  << "\t\t" << section\_header->VirtualAddress << "\t\t" <<  section\_header->SizeOfRawData << "\t\t"  << section\_header->PointerToRawData << "\t\t\t" << section\_header->Characteristics << endl;  }  DWORD el\_anew = dosheader->e\_lfanew+24 + nt\_header->FileHeader.SizeOfOptionalHeader;  DWORD byteofthis = 40 \* nt\_header->FileHeader.NumberOfSections;  // cout<<"SizeOfOptionalHeader:"<<nt\_header->FileHeader.SizeOfOptionalHeader<<endl;  // cout<<"sizeof(section\_header):"<<sizeof(section\_header)<<endl;  // cout<<"nt\_header->FileHeader.NumberOfSections:"<<nt\_header->FileHeader.NumberOfSections<<endl;  cout<<"byteofthis:"<<byteofthis<<endl;  UINT8\* Content = new UINT8[byteofthis];  memcpy(Content, (UINT8\*)pMapping + el\_anew, byteofthis);  int flag = 0;  for (size\_t i = 0; i < byteofthis; ++i)  {  unsigned char c = Content[i]; // must use unsigned char to print >128 value  flag++;  if (c < 16)  printf("0%x ", c);  else  printf("%x ", c);  if ((flag) % 16 == 0)  printf("\n");  }  //el\_anew的位置0x3C,大小为uint32\_t  //NT\_header+optionalHeader+sectionHeader的位置  // uint32\_t el\_anew = dosheader->e\_lfanew;  cout << "el\_fanew:" << el\_anew << endl;  //大小=18H+fileHeader.sizeofoptionalheader+sizeof(section\_header)\* nt\_header->FileHeader.NumberOfSections  // uint32\_t byteofthis=24+nt\_header->fileHeader.sizeofoptionalheader+sizeof(section\_header)\* nt\_header->FileHeader.NumberOfSections;  //转移的位置：1023，大小相同，  //59336704  OVERLAPPED overLap = { 0 };  overLap.Offset = 1024-byteofthis;  DWORD writeByte;  DWORD readsize = 0;  LARGE\_INTEGER offset;//long long signed  offset.QuadPart = (ULONGLONG)0 \* (ULONGLONG)512;//0  SetFilePointer(hFile, offset.LowPart, &offset.HighPart, FILE\_BEGIN);  if (!WriteFile(hFile, Content, byteofthis, &writeByte, &overLap)) {  cout << dec << "error code" << GetLastError() << endl;  }  DWORD sizeofoptionalheader=1024-el\_anew-byteofthis;  sizeofoptionalheader=sizeofoptionalheader+nt\_header->FileHeader.SizeOfOptionalHeader;  cout<<sizeofoptionalheader<<endl;//2c0      //修改sizeofoptionalheader  UINT8 data[] = { 0xc0, 0x02};  DWORD el\_anewweizhi = 0xb0+0x14;  DWORD byteofthis1 = 2;  OVERLAPPED overLap1 = { 0 };  overLap1.Offset = 0xb0+0x14;  // DWORD writeByte;  readsize = 0;  // LARGE\_INTEGER offset;//long long signed  offset.QuadPart = (ULONGLONG)0 \* (ULONGLONG)512;//0  SetFilePointer(hFile, offset.LowPart, &offset.HighPart, FILE\_BEGIN);  if (!WriteFile(hFile, data, byteofthis1, &writeByte, &overLap1)) {  cout << dec << "error code" << GetLastError() << endl;  }  CloseHandle(hFile);  }  （3）节间免疫  #include <windows.h>  #include <winioctl.h> //DDK驱动开发与控制  #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #include <stdint.h>  #include<iostream>  #include <string>  #include <sstream>  using namespace std;  //节表免疫，方法一  string decimal\_to\_hex\_str(int decimal) {  stringstream ss;  ss << hex << decimal;  return ss.str();  }  int main() {  // LPCWSTR path = "P:\\exp1.exe";  HANDLE hFile;  HANDLE hMapping;  LPVOID pMapping;  hFile = CreateFile("P:\\exp1.exe", GENERIC\_READ|GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING,  FILE\_ATTRIBUTE\_NORMAL, NULL);  if (INVALID\_HANDLE\_VALUE == hFile) {  return 0;  }  //将PE文件映射到内存  hMapping = CreateFileMapping(hFile, NULL, PAGE\_READONLY, 0, 0, 0);  if (!hMapping) {  return 0;  }  pMapping = MapViewOfFile(hMapping, FILE\_MAP\_READ, 0, 0, 0);//返回的是map的开始地址  if (!pMapping) {  return 0;  }  PIMAGE\_DOS\_HEADER dosheader;  dosheader = (PIMAGE\_DOS\_HEADER)pMapping;  if (dosheader->e\_magic != IMAGE\_DOS\_SIGNATURE) {  cout << "无效的PE文件" << endl;  return 0;  }  PIMAGE\_NT\_HEADERS nt\_header;  nt\_header = (PIMAGE\_NT\_HEADERS)((BYTE\*)pMapping + dosheader->e\_lfanew);  if (nt\_header->Signature != IMAGE\_NT\_SIGNATURE) {  cout << "无效的PE文件" << endl;  return 0;  }  PIMAGE\_SECTION\_HEADER section\_header;  section\_header = IMAGE\_FIRST\_SECTION(nt\_header);  for (int i = 0; i < nt\_header->FileHeader.NumberOfSections;  i++, section\_header++) {  cout << section\_header->Name << "\t" << section\_header->Misc.VirtualSize  << "\t\t" << section\_header->VirtualAddress << "\t\t" <<  section\_header->SizeOfRawData << "\t\t"  << section\_header->PointerToRawData << "\t\t\t" << section\_header->Characteristics << endl;    }      for(int i=0;i<nt\_header->FileHeader.NumberOfSections;i++){  DWORD el\_anew =424+40\*i;  // DWORD el\_anew = 40+24 + nt\_header->FileHeader.SizeOfOptionalHeader+40\*i;  DWORD byteofthis=4;  UINT8\* Content = new UINT8[byteofthis];  memcpy(Content, (UINT8\*)pMapping + el\_anew+16, byteofthis);  int flag = 0;  for (size\_t i = 0; i < byteofthis; ++i)  {  unsigned char c = Content[i]; // must use unsigned char to print >128 value  flag++;  if (c < 16)  printf("0%x ", c);  else  printf("%x ", c);  if ((flag) % 16 == 0)  printf("\n");  }  cout<<endl;    OVERLAPPED overLap = { 0 };  overLap.Offset = 424+40\*i+8;  DWORD writeByte;  DWORD readsize = 0;  LARGE\_INTEGER offset;//long long signed  offset.QuadPart = (ULONGLONG)0 \* (ULONGLONG)512;//0  SetFilePointer(hFile, offset.LowPart, &offset.HighPart, FILE\_BEGIN);  if (!WriteFile(hFile, Content, byteofthis, &writeByte, &overLap)) {  cout << dec << "error code" << GetLastError() << endl;  }      }    // DWORD el\_anew = dosheader->e\_lfanew+24 + nt\_header->FileHeader.SizeOfOptionalHeader;  // DWORD byteofthis = 40 \* nt\_header->FileHeader.NumberOfSections;  //// cout<<"SizeOfOptionalHeader:"<<nt\_header->FileHeader.SizeOfOptionalHeader<<endl;  //// cout<<"sizeof(section\_header):"<<sizeof(section\_header)<<endl;  //// cout<<"nt\_header->FileHeader.NumberOfSections:"<<nt\_header->FileHeader.NumberOfSections<<endl;  // cout<<"byteofthis:"<<byteofthis<<endl;  // UINT8\* Content = new UINT8[byteofthis];  // memcpy(Content, (UINT8\*)pMapping + el\_anew, byteofthis);  // int flag = 0;  // for (size\_t i = 0; i < byteofthis; ++i)  // {  // unsigned char c = Content[i]; // must use unsigned char to print >128 value  // flag++;  // if (c < 16)  // printf("0%x ", c);  // else  // printf("%x ", c);  // if ((flag) % 16 == 0)  // printf("\n");  // }  //el\_anew的位置0x3C,大小为uint32\_t  //NT\_header+optionalHeader+sectionHeader的位置  // uint32\_t el\_anew = dosheader->e\_lfanew;  // cout << "el\_fanew:" << el\_anew << endl;  //大小=18H+fileHeader.sizeofoptionalheader+sizeof(section\_header)\* nt\_header->FileHeader.NumberOfSections  // uint32\_t byteofthis=24+nt\_header->fileHeader.sizeofoptionalheader+sizeof(section\_header)\* nt\_header->FileHeader.NumberOfSections;  //转移的位置：1023，大小相同，  //59336704  // OVERLAPPED overLap = { 0 };  // overLap.Offset = 1024-byteofthis;  // DWORD writeByte;  // DWORD readsize = 0;  // LARGE\_INTEGER offset;//long long signed  // offset.QuadPart = (ULONGLONG)0 \* (ULONGLONG)512;//0  // SetFilePointer(hFile, offset.LowPart, &offset.HighPart, FILE\_BEGIN);  // if (!WriteFile(hFile, Content, byteofthis, &writeByte, &overLap)) {  // cout << dec << "error code" << GetLastError() << endl;  // }  //  // DWORD sizeofoptionalheader=1024-el\_anew-byteofthis;  // sizeofoptionalheader=sizeofoptionalheader+nt\_header->FileHeader.SizeOfOptionalHeader;  // cout<<sizeofoptionalheader<<endl;//2c0  //  //  //  ////修改sizeofoptionalheader  //  // UINT8 data[] = { 0xc0, 0x02};  //  // DWORD el\_anewweizhi = 0xb0+0x14;  // DWORD byteofthis1 = 2;  //  // OVERLAPPED overLap1 = { 0 };  // overLap1.Offset = 0xb0+0x14;  //// DWORD writeByte;  // readsize = 0;  //// LARGE\_INTEGER offset;//long long signed  // offset.QuadPart = (ULONGLONG)0 \* (ULONGLONG)512;//0  // SetFilePointer(hFile, offset.LowPart, &offset.HighPart, FILE\_BEGIN);  // if (!WriteFile(hFile, data, byteofthis1, &writeByte, &overLap1)) {  // cout << dec << "error code" << GetLastError() << endl;  // }  CloseHandle(hFile);  } | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   2编程实现   1. 节表免疫，第一种结果：     用winhex查看得到：       1. 节表免疫，第二种结果：     Winhex：       1. 节间免疫     再次运行代码：    Winhex： | | | | | | |