1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт прикладной математики и механики
5. **Высшая школа кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА №5**

«**Исследование принципов разработки средств защиты»**

по дисциплине «Теория и системы управления информационной безопасностью**»**

1. Выполнили

студенты гр. 3651003/50801 Д.И. Кожушок

А.А. Корольков

Ю.В. Романюк

* 1. Руководитель Е.В. Жуковский

1. Санкт-Петербург
2. 2019

**1. Цель работы**

Изучение принципов разработки средств антивирусной защиты. Изучение технологий и механизмов ОС Windows, используемых для реализации функций защиты. Получение навыков проектирования и разработки многокомпонентного программного обеспечения.

**2. Формулировка задания**

Лабораторная работа выполняется в группах по 3 человека. В ходе выполнения лабораторной работы необходимо спроектировать и разработать программное обеспечение (ПО), реализующее основные функции, выполняемые современным антивирусным программном обеспечением.

В ходе выполнения лабораторной работы необходимо разработать DLL-библиотеку, которая путем перехвата вызовов функций в анализируемой программу реализует следующий функционал:

1. Отслеживает получаемые и передаваемые сетевые данные программ и осуществляет их сканирование на предмет наличия в них вредоносных сигнатур. Необходимо реализовать перехват функций из наиболее распространенных библиотек, используемых для сетевого взаимодействия (winhttp.dll, wininet.dll, ws32\_2.dll, libcurl.dll).
2. Осуществляет анализ информации о сетевых соединениях, создаваемых анализируемой программой и сравнивает с перечнем известных индикаторов компрометации (IoC).
3. Анализирует исходный код веб-страниц, включая Javascript и ссылки на другие ресурсы на предмет наличия потенциальных угроз. Осуществляется сравнение с известными индикаторами компрометации (IoC).
4. Отслеживает вводимые команды в консольные приложения и непосредственно в консоль. Осуществляется сравнение с известными командами, используемыми в целенаправленных атаках (MITRE ATT&CK).

Реализовать и протестировать п. 1-3 на примере известных веб-браузеров. Продемонстрировать возможность обнаружения вредоносной активности.

Реализовать и протестировать п. 4 на примере выявления опасных вредоносных действий, выполняемых через интерпретатор командной строки.

Интегрировать разработанный модуль в ранее реализованную защитную программу и реализовать его внедрение в запускаемые процессы через механизм APC.

**3. Результаты работы**

Microsoft Detours — проект, разрабатываемый в лабораториях Microsoft Research, для мониторинга и инструментирования вызовов API в Windows.

Для того чтобы использовать данный проект в лабораторной работе был создан проект в Visual Studio. Библиотека detours была прилинкована в проекту через настройки линкера.

Структура проекта:

BOOL(WINAPI \*pX) (…) = X; // сохраняем указатель на функцию X

BOOL WINAPI MyXHook ( …) {…} // поддельная функция

INT APIENTRY DllMain(HMODULE hDLL, DWORD Reason, LPVOID Reserved) {

switch (Reason) {

case DLL\_PROCESS\_ATTACH: {

DetourTransactionBegin();

DetourUpdateThread(GetCurrentThread()); // инициализирует установку хука для указанного потока

DetourAttach(&(PVOID&)pX, MyXHook); // установка хука

DetourTransactionCommit();

break;

}

case DLL\_PROCESS\_DETACH: { }

}

return TRUE;

}

1. Отслеживание получаемых и передаваемых сетевых данных. Перехват функций из наиболее распространенных библиотек, используемых для сетевого взаимодействия (winhttp.dll, wininet.dll, ws32\_2.dll, libcurl.dll).

Было проведено тестирование сетевых взаимодействий программ и было принято решение о расширении списка перехватываемых функций. Для отслеживания были выбраны следующие функции:

wininet:

**InternetCanonicalizeUrlA;**

**InternetCanonicalizeUrlW;**

**InternetCheckConnectionA;**

**InternetCheckConnectionW;**

**InternetCombineUrlA;**

**InternetCombineUrlW;**

**InternetConfirmZoneCrossingA;**

**InternetConfirmZoneCrossingW;**

**InternetCrackUrlA;**

**InternetCrackUrlW;**

**InternetCreateUrlA;**

**InternetCreateUrlW;**

**InternetConnectA;**

**InternetConnectW;**

**InternetGoOnlineA;**

**InternetGoOnlineW;**

**InternetOpenUrlA;**

**InternetOpenUrlW**

urlmon:

**URLDownloadToFileA;**

**URLDownloadToFileW;**

winhttp:

**WinHttpCrackUrl;**

**WinHttpCreateUrl;**

**WinHttpConnect;**

**WinHttpGetProxyForUrl;**

**WinHttpGetProxyForUrlEx;**

winsock2:

**connect;**

**gethostbyaddr;**

**gethostbyname;**

**inet\_addr;**

**WSAConnectByNameA;**

**WSAConnectByNameW;**

**WSARecvFrom;**

**WSASendTo;**

**recvfrom;**

**sendto;**

**getaddrinfo;**

windns:

**DnsGetProxyInformation;**

**DnsNameCompare\_A;**

**DnsNameCompare\_W;**

**DnsQuery\_A;**

**DnsQuery\_W;**

**DnsQueryEx.**

Из-за несовместимости функции библиотек urlmon и wininet были собраны в отдельный dll файл.

Для обработки данных потребовалось написать 2 функции по проверке IP адреса и URL.

Так же было организовано ведение логов с указанием процесса, использованной перехватываемой функции и данных о запросе. Лог хранится по пути C:\\log\\log.txt и C:\\log\\log2.txt.

2. Поиск индикаторов компрометации (IoC)

Индикаторы компрометации (IoC) — это отдельные известные вредоносные события, указывающие на то, что безопасность сети или компьютера уже нарушена.

В отличие от определений оповещений индикаторы считаются свидетельствами бреши в системе безопасности. Зачастую они появляются уже после совершения атаки и достижения ее цели (например, получение файлов в зараженной системе с их последующей отправкой на удаленный сервер).

К сетевым индикаторам компрометации относятся URL, совокупность IP-адресов и портов.

Для поиска IoC было использовано API сайта https://www.threatcrowd.org/ . оно представляет из себя запросы следующего формата:

https://www.threatcrowd.org/searchApi/v2/email/report/?**email**=william19770319@yahoo.com

https://www.threatcrowd.org/searchApi/v2/domain/report/?**domain**=aoldaily.com

https://www.threatcrowd.org/searchApi/v2/ip/report/?**ip**=188.40.75.132

https://www.threatcrowd.org/searchApi/v2/antivirus/report/?**antivirus**=plugx

https://www.threatcrowd.org/searchApi/v2/file/report/?**resource**=ec8c89aa5e521572c74e2dd02a4daf78

В ответ на запрос сервис отправляет страницу, содержащую информацию о запросе (поиск по совпадению, соответствующие IP адреса, связанные email и дату последнего обновления данной записи в базе). Также в ответе содержится строка вида **"votes":х,** где на месте х может стоят число от -1 до 1. -1 означает о положительном тесте на компрометацию, 0 о недостаточности информации или необходимости в более точном запросе, 1 о отрицательном тесте на компрометацию.

База данных в предоставленном сервисе обдирная , но недостаточная , так как применяя ее для записей из других открытых черных списков доменов (<http://www.covert.io/threat-intelligence/> ) не всегда получается добиться желаемого результата. Но доступное и удобное API позволяют значительно ускорить и упростить разработку, а также обеспечивает облачное обновление базы данных.

Для работы с API были использованы исходные коды программы curl, с их помощью совершались запросы к API.

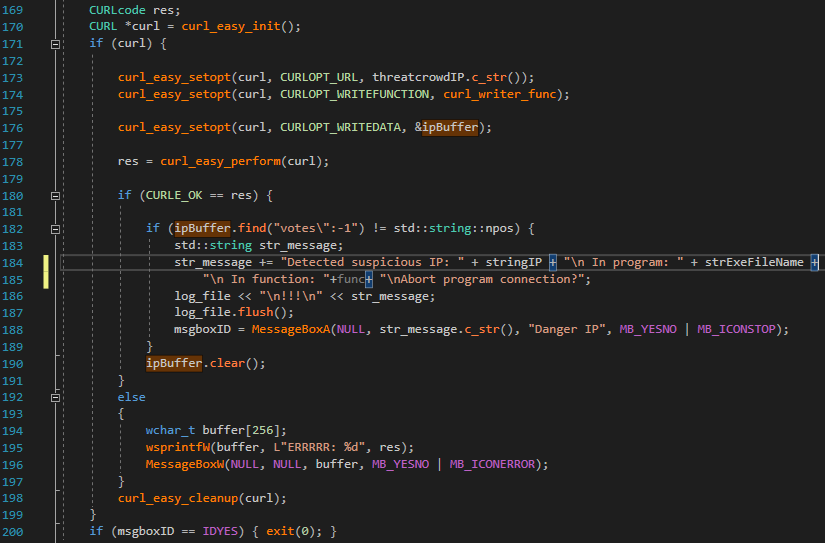


Рисунок 1 -запрос к базе IoC с помощью curl

Программа логирует проверяемые значения и функции, а случае обнаружения потенциально компрометирующего URL или IP адреса выводит информационное окно, предупреждающее и спрашивающее у пользователя разрешение на дальнейшее выполнение кода программы.

При совершении запроса API так же происходит срабатывание функции перехвата, поэтому для избежание рекурсивной проверки IP или URL, а также для избегания излишних проверок (так как они затратны по времени) URL localhost, threatcrowd.org, loopback и соответствующие IP адреса были добавлены в исключения.

Для внедрения в процессы было использовано приложение , поставляющееся вместе с Detours : withdll.exe. Было протестировано несколько сценариев использования.

Для первого была использована утилита curl, с ее помощью был произведен доступ к доверенному сайту(Рисунок 2), и к потенциально компрометирующему (Рисунок 3) .

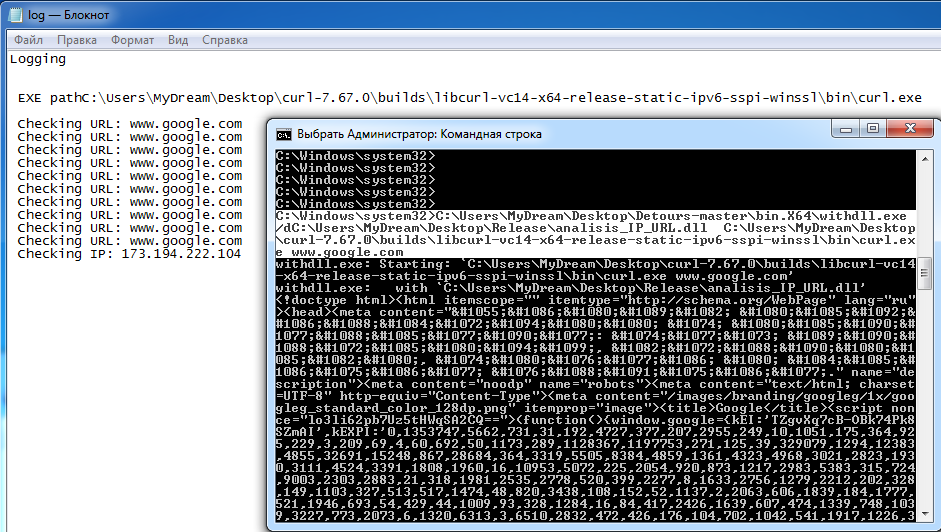


Рисунок 2 -Доступ к сайту gooogle.com через утилиту curl

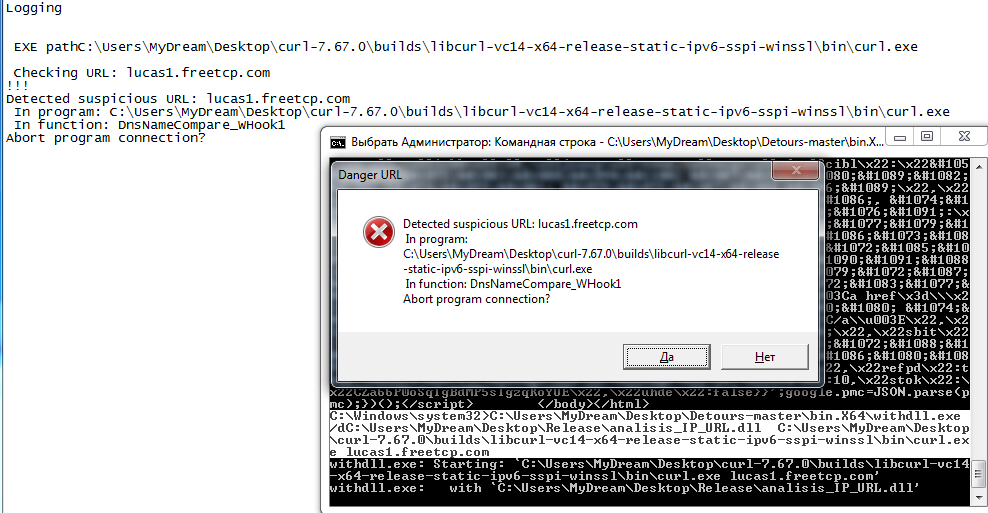


Рисунок 3- доступ к сайту lucas1.freetcp.com через утилиту curl.

Проверка обращений по IP адресу также проходит успешно. ( Рисунок 4)

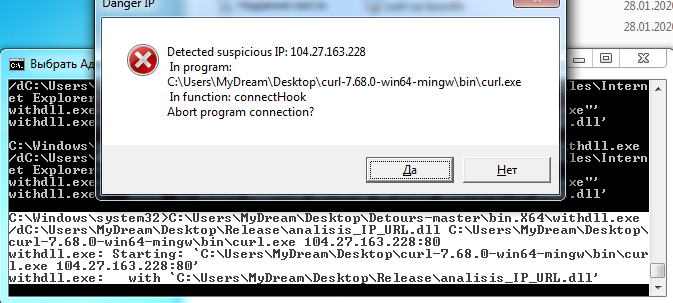


Рисунок 4 обращение к ресурсу по IP адресу

Далее были проведены тесты на примере браузера Internet Explore.

Для того чтобы протестировать необходимо было настроить Internet Explore. Дело в том, что в настроенном по умолчанию браузере каждая вкладка выполняется в своем процессе и не удавалось выполнить перехват информации. Таким образом, был добавлен ключ реестра:

HKCU\Software\Microsoft\Internet Explorer\Main\TabProcGrowth равный 0, что означает, что все вкладки буду выполняться в рамках одного процесса.

Результаты проведённых тестов можно увидеть на рисунке ниже.

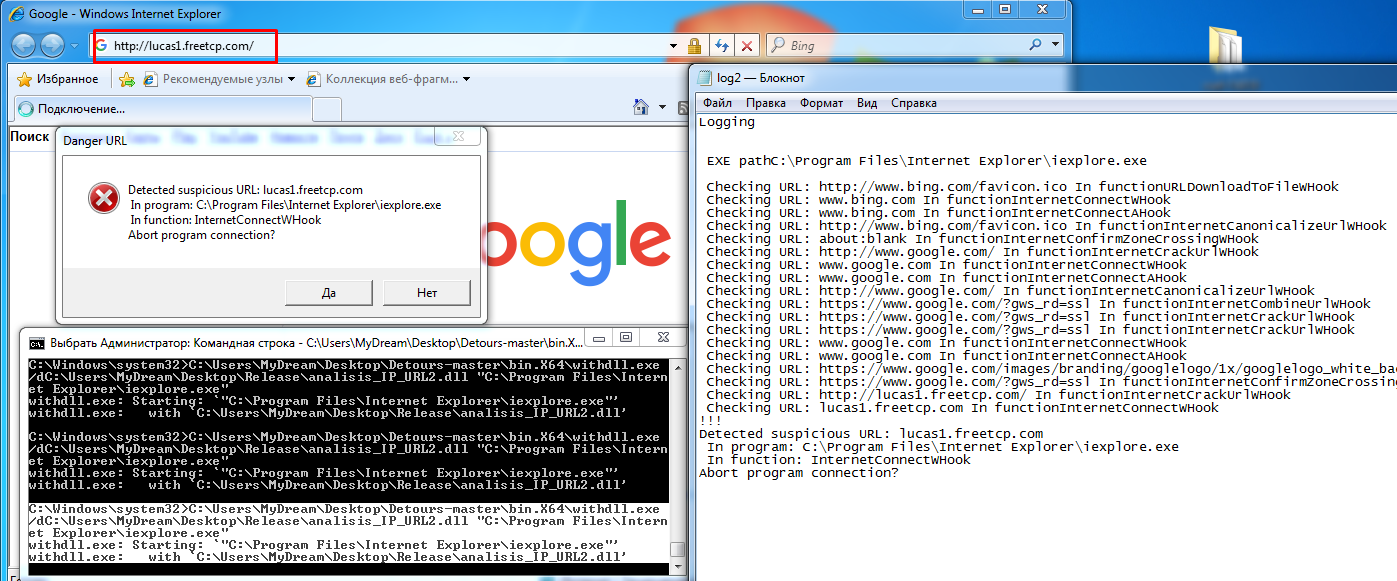


Рисунок 5 Изначальный переход на доверенный сайт google.com, c последующим переходом к lucas1.freetcp.com

Разработанные dll работают корректно, но выбранная база данных IoC часто дает ложноположительный результат, так как находит подстроки из скомпрометированных доменов в искомом домене. Также осуществление запросов очень затратно по времени , из-за чего становится невозможно использование данного способа проверки в качестве постоянного слежения за всеми запущенными процессами .

3. Анализ исходного кода веб-страниц, включая Javascript на предмет наличия потенциальных угроз.

Общий алгоритм чтения www страницы с сервера (<https://www.rsdn.org/article/inet/wininet.xml>):

InternetOpen

InternetConnect

HttpOpenRequest

HttpSendRequest

InternetReadFile

InternetCloseHandle

InternetCloseHandle

InternetCloseHandle

InternetOpen. Эта функция инициализирует WinInet и возвращает дескриптор, который необходим для вызова других функций WinInet.

InternetConnect. Эта функция открывает FTP или HTTP сессию для заданного сайта.

HttpOpenRequest. HTTP запрос выполняется в несколько этапов: открытие запроса, определение HTTP заголовка, собственно отправка запроса, чтение и обработка данных. Эта функция, как следует из её названия, открывает HTTP запрос.

HttpSendRequest. Отсылает запрос на сервер.

InternetReadFile. Эта функция позволяет читать данные результата запроса.

InternetQueryDataAvailable. Это функция запрашивает сервер, есть ли еще данные для чтения.

InternetCloseHandle. Эта функция закрывает любой из дескрипторов, созданных предыдущими функциями.

Таким образом, было реализовано 4 обработчика функций: MyHttpOpenRequestW, MyInternetReadFile, MyInternetQueryDataAvailable, MyInternetCloseHandle.

Для хранения исходного кода веб-страницы используется структура:

struct site {

HINTERNET hash; //дескриптор сессии

std::string info; //исходный код страницы

}

site\_list[SIZE];

При открытии HTTP запроса в структуру записывается дескриптор сессии и обнуляется исходный код:

HINTERNET MyHttpOpenRequestW(HINTERNET hConnect, LPCWSTR lpszVerb, LPCWSTR lpszObjectName, LPCWSTR lpszVersion, LPCWSTR lpszReferrer, LPCWSTR \*lplpszAcceptTypes, DWORD dwFlags, DWORD\_PTR dwContext)

{

HINTERNET res = pHttpOpenRequestW(hConnect, lpszVerb, lpszObjectName, lpszVersion, lpszReferrer, lplpszAcceptTypes, dwFlags, dwContext);

if (res != NULL)

{

site\_list[cur].hash = res;

site\_list[cur].info = "";

cur++;

}

return res;

}

При считывании данных информация записывается в структуру по соответствующему дескриптору:

BOOL MyInternetReadFile(HINTERNET hFile, LPVOID lpBuffer, DWORD dwNumberOfBytesToRead, LPDWORD lpdwNumberOfBytesRead)

{

BOOL res = pInternetReadFile(hFile, lpBuffer, dwNumberOfBytesToRead, lpdwNumberOfBytesRead);

if (res != 0 && html\_js(hFile) != FALSE)

{

for (int i = 0; i < cur; i++)

{

if (site\_list[i].hash == hFile)

{

site\_list[i].info += (LPSTR)lpBuffer;

}

}

}

return res;

}

После того, как прочитаны все данные, исходный код целиком загружен, он отправляется на проверку:

BOOL MyInternetQueryDataAvailable(HINTERNET hFile, LPDWORD lpdwNumberOfBytesAvailable, DWORD dwFlags, DWORD\_PTR dwContext)

{

int msgboxID = IDNO;

DWORD NumberOfBytes = \*lpdwNumberOfBytesAvailable;

BOOL res = pInternetQueryDataAvailable(hFile, lpdwNumberOfBytesAvailable, dwFlags, dwContext);

if (res != 0 && NumberOfBytes != 0 && (\*lpdwNumberOfBytesAvailable) == 0)

{

string data;

for (int i = 0; i < cur; i++)

{

if (site\_list[i].hash == hFile)

{

data = site\_list[i].info;

}

}

if (data.size() != 0)

{

std::ofstream fout("C:\\Users\\Julia\\Desktop\\5\\data.tmp");

fout << data;

fout.close();

if (analysisInfo(data) != 0)

{

msgboxID = MessageBoxW(NULL, L"FOUND A POTENTIAL THREAT!", L"DANGER", MB\_YESNO | MB\_ICONERROR););

}

}

}

if (msgboxID == IDYES)

{

exit(0);

}

return res;

}

При закрытии дескриптора обнуляется структура:

BOOL MyInternetCloseHandle(HINTERNET hInternet)

{

try

{

for (int i = 0; i < cur; i++)

{

if (site\_list[i].hash == hInternet)

{

site\_list[i].hash = NULL;

site\_list[i].info = "";

}

}

}

catch (...) {};

return pInternetCloseHandle(hInternet);

}

Анализ исходного кода страницы заключается в запуске классификатора, обученного в курсе МБКС:

WCHAR cmd[32] = L"C:\\Windows\\System32\\cmd.exe";

WCHAR args[90] = L"/C python C:\\Users\\Julia\\Desktop\\5\\classifier.py C:\\Users\\Julia\\Desktop\\5\\data.tmp";

CreateProcessW(cms, args, NULL, NULL, FALSE, 0, NULL, L"C:\\Users\\Julia\\Desktop\\5", &si, &pi)

Для тестирования был создан сайт на XAMPP. Примеры вредоносного JavaScript были взяты: <https://github.com/HynekPetrak/javascript-malware-collection/>

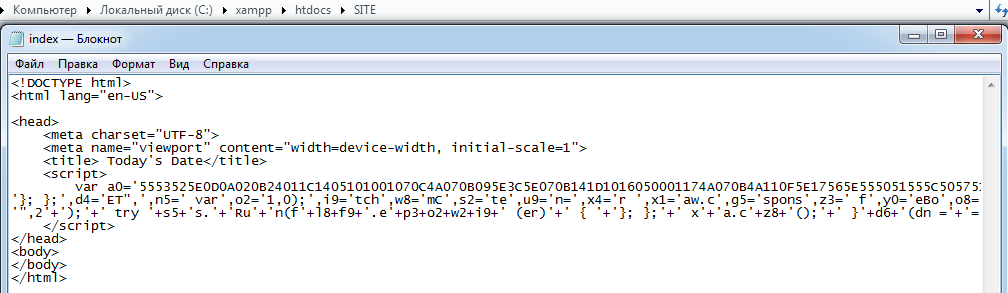


Рисунок – Вредоносная страница

При посещении данной страницы появляется следующее сообщение:

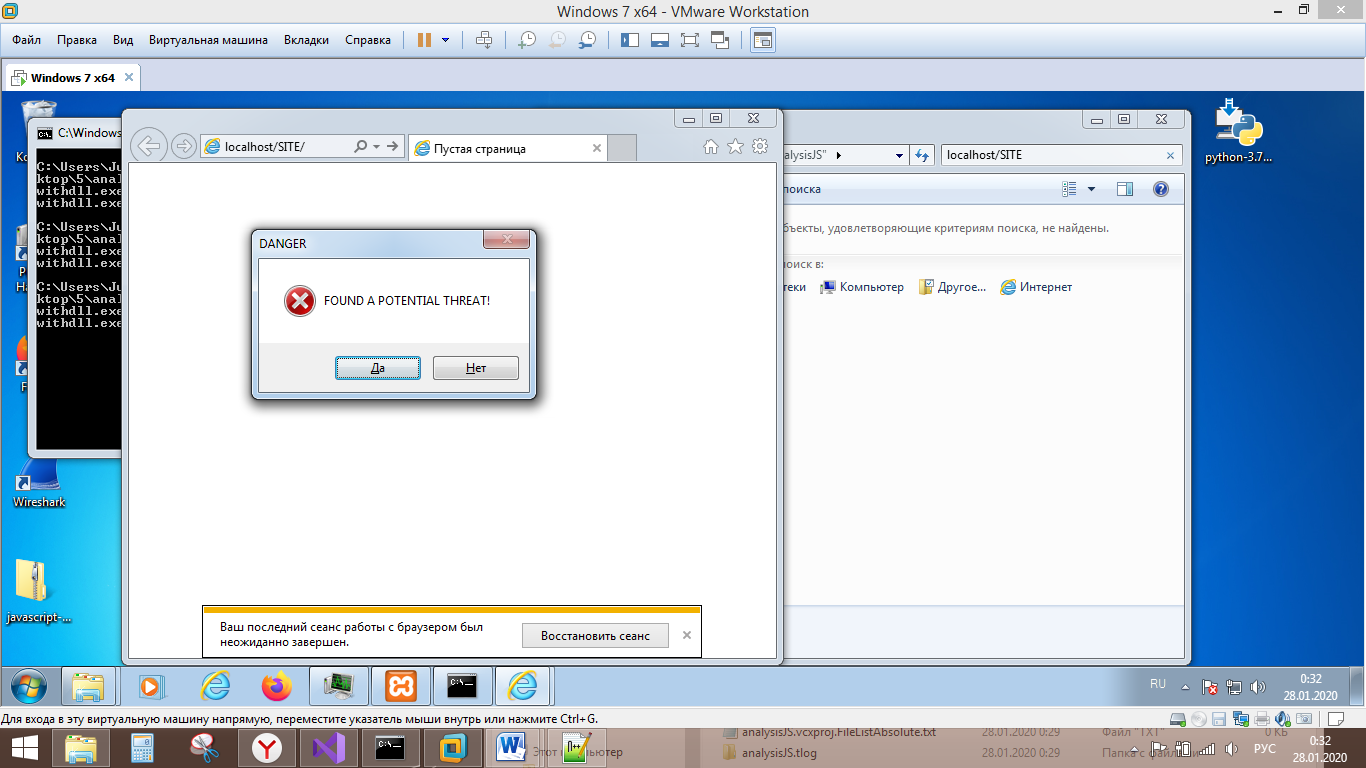


Рисунок – Обнаружение потенциальной угрозы

4. Отслеживание вводимых команд в консоль.

Функция ReadConsole читает символьный ввод данных из консольного буфера ввода и удаляет их из буфера. Таким образом, необходимо перехватит данную функцию и проанализировать буфер.

При вызове функции буфер отправляется на проверку:

BOOL WINAPI MyReadConsoleA(\_In\_ HANDLE hConsoleInput, \_Out\_ LPVOID lpBuffer, \_In\_ DWORD nNumberOfCharsToRead, \_Out\_ LPDWORD lpNumberOfCharsRead, \_In\_opt\_ PCONSOLE\_READCONSOLE\_CONTROL pInputControl)

{

BOOL ret = pReadConsoleA(hConsoleInput, lpBuffer, nNumberOfCharsToRead, lpNumberOfCharsRead, pInputControl);

console(lpBuffer, A);

return ret;

}

Анализ вводимых команд осуществляется поиском подстроки (опасных команд) в строке (перехваченном буфере):

for (int i = 0; i < 27; i++)

{

LPWSTR command = (LPWSTR)lpBuffer;

if (wcsstr(\_wcslwr(command), commandReadConsoleW[i]) != NULL)

tmp = i;

}

if (tmp >= 0)

{

wchar\_t msg[500] = L"Command: \n";

wcscat\_s(msg, commandReadConsoleW[tmp]);

MessageBoxW(NULL, msg, L"DANGER", MB\_OK | MB\_ICONSTOP);

exit(0);

}

Список уязвимых команд был сформирован в процессе изучения основных техник, применяемых злоумышленниками на различных этапах осуществления хакерских атак, представленных корпорацией MITRE.

Список опасных консольных команд:

const char\* commanReadConsoleA[27] = {

"tasklist",

"ver",

"nslookup",

"ipconfig /all",

"net time",

"systeminfo",

"netstat -an",

"qprocess",

"query user",

"whoami",

"net start",

"time /t",

"dir",

"net view",

"net use",

"ping",

"type",

"net user",

"net localgroup",

"net group",

"net config",

"net share",

"dsquery",

"nbtstat -a",

"net session",

"reg",

"iexplore"

};

Пример работы:

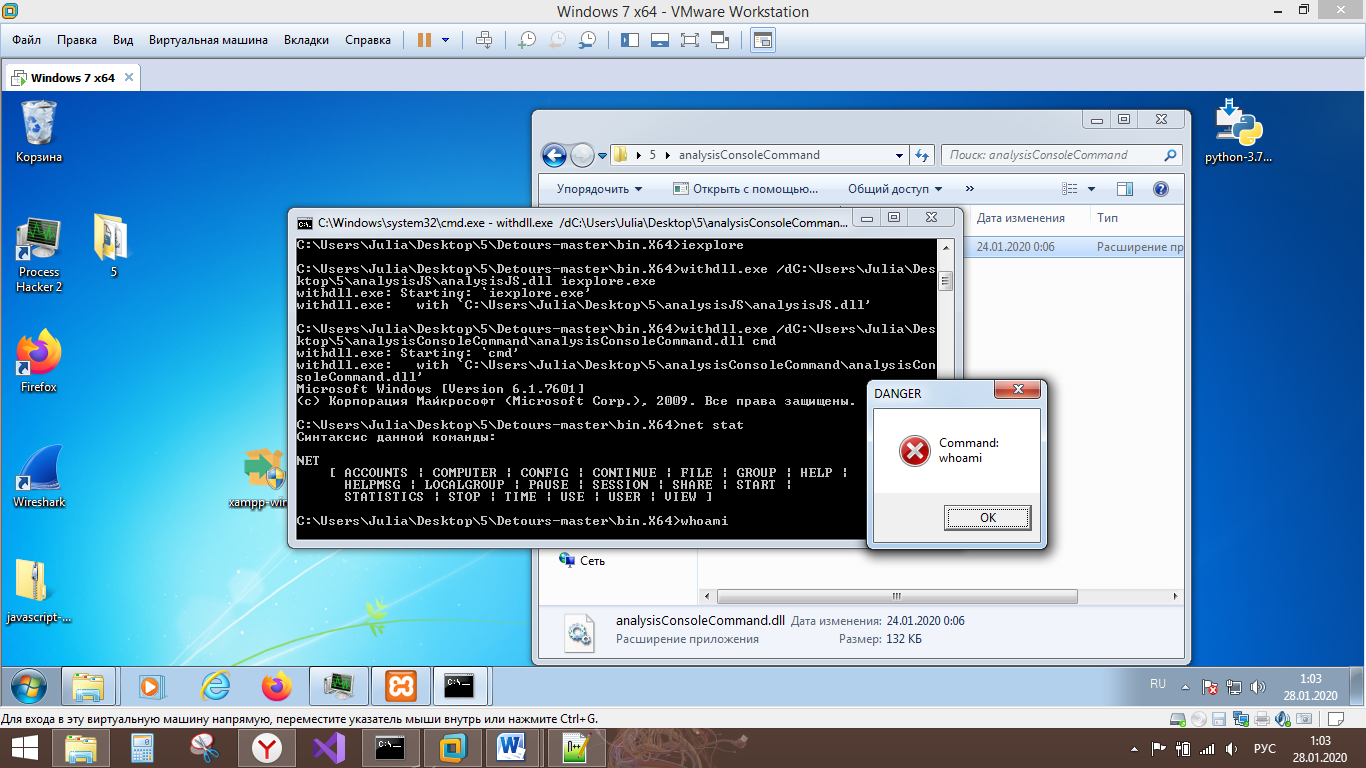


Рисунок – Обнаружение опасной команды

**4. Выводы**

В ходе выполнения лабораторной работы были разработаны DLL-библиотеки, которые путём перехвата вызовов функций в анализируемой программе осуществляют:

1. анализ информации о сетевых соединениях, создаваемых анализируемой программой;
2. анализ исходного кода веб-страниц, включая Javascript;
3. анализ команд, вводимых в консольные приложения.

Работоспособность разработанных библиотек была проверена на примере браузера Internet Explore.