Статья Ресёрч. Детектируем какие АВ стоят на ПК юзера, если он просто перешёл по ссылке

xss.is/threads/38955

Оглавление (для быстрого поиска по CTRL+F)
- Предисловие
- Как работает скрипт
- Как исследовать АВ
- Тестирование АВ
- Выводы

Предисловие

Я задался вопросом, можно ли детектировать какой именно Антивирус стоит у юзера на машине если он просто перейдёт по моей ссылке? Хорошо обдумав этот вопрос мне на ум пришли 4 гипотизы как это потанциально можно провернуть.

1) Так как ав всегда добавляют свои SSL серты в браузер для снифинга HTTPS трафика, то в теории можно это детектить. Но я сразу откинул эту идею так как точно знаю что браузеры не предоставляют каких либо API для доступа к инфе какие у юзера стоят сертификаты в браузере. Такое API есть вроде как только для браузерных расширений, но это вообще не вариант.
2) Некоторые АВ при своей установке ещё автоматически устанавливают своё расширение в браузер. Я предположил что эти расширения могут инжектить что-то в DOM всех HTML страниц и таким образом их наличие в браузере можно детектировать с любой старницы через JS. Гипотиза могла потанциально выстрелить, но после некоторых тестов на которые я убил сутки, эта гипотиза оказалась не удачной. Причина в том что только несколько АВ уставливают свои расширения в браузер при их установке на сам ПК. Но все они ничего не инжектят в DOM страницы.
3) В теории можно детектировать наличие определённого АВ путём создание 10-ти фишинговых доменов, повесить на них фишинги и каждый из доменов отправить в АВ лабу и таким образом у нас будет 10 доменов каждый из которых детектится разными АВ. Но в этой гипотизе есть два узких места. Первый - если мы будем добавлять эти домены в айфрейме на нашем основном домене, то как АВ поведут себя, будут ли они сразу лочить мейн домен по которому перешёл юзер, или только айфрейм в коде HTML? А если с этих доменов подгружать ресурсы (CSS/JS/картинки) будет ли такая-же проблема? Просто если АВ будет лочить не какой-то линк в самом HTML мейн страницы, а саму страницу - то в теории на первый взгляд это не обойти и задетектить так не удастся. Второе узкое место - наверняка я бы столкнулся с проблемой что АВ шарят свои базы чёрных списков доменов другим АВ и потанциально отдав 1 фиш домен условному Авасту, он может отдать его-же всем остальным АВ, или к примеру

Google Safe Browsing, а от него уже этот детект получат другие АВ. Из-за этих сложностей я откинул этот вариант, так как реализовать такой концепт само по себе труднозатратно, так ещё и вероятнее всего любая из этих потанциальных проблем с слишком большой вероятностью безповоротно зафейлит сам концепт.
4) И последнее что мне пришло в голову - это детектить АВ по наличию открытых на прослушивание портов. То есть мы из JS можем чекать определённые порты и понимать открыты ли они на прослушивание или нет. Такая техника используется в различных анти-фрод скриптах которые проверяют прослушиваются ли на машине популярные порты, такие как VNC, что косвенно может говорить о том что машина протроянена всяким троянским софтом на подобии HVNC. Ну и собственно мне и пришла мысль делать такие-же проверки, но детектя сами антивирусы. Эта гипотизу я и начал ресёрчить и добился некоторых положительных результатов, о чём поделюсь с вами в этой статье.

Я точно не знал как именно происходит детект открытых портов из JS из-за этого начал гуглить уже рабочие скрипты для детекта, но ничего рабочего не нагуглил, что меня на самом деле удивило и отсюда пришлось веселиться с деобфускацией и отладкой антифрод скриптов где это реализовано. Что и как я деобфусцировал говорить не буду так как по мне это не нужная инфа. Переписав скрипт под себя я получил на выходе такой рабочий код, в него сразу добавлены порты для детекта Аваста/AVG и Касперского.

Как работает скрипт
1) Инициализируется соединение WebSocket Secure (wss) через JS по адресу 127.0.0.1 по списку портов который находится в массиве “ports”, а так же запоминается время, когда был инициализирован WebSocket. Обязательно нужно использовать wss (то есть шифрованное соединение с сокетом), иначе тайминги будут непредсказуемы. Подозреваю что это как раз и связано с TLS-ом.
2) Скрипт ожидает вызова CallBack функций WebSocket, а именно: onopen, onerror, onclose
3) Скрипт получает CallBack на одну из данных функций и подсчитывает время формулой (данное время минус время инициализирования вебсокета)
4) Закрывает WebSocket запоминая время, которое было подсчитано в предыдущем пункте
5) Если коллбеки не были вызваны, вызывается функция, переданная в setTimeout с интервалом в 1500, который принудительно закрывает сокет, логируя это
6) После того как скрипт отработал, он в синхронном режиме переходит на следующий порт

Открытыми портами считаются те, у которых тайминг соединения меньше 500 мс. Это значение было мной выявлено как оптимальное при тестах.

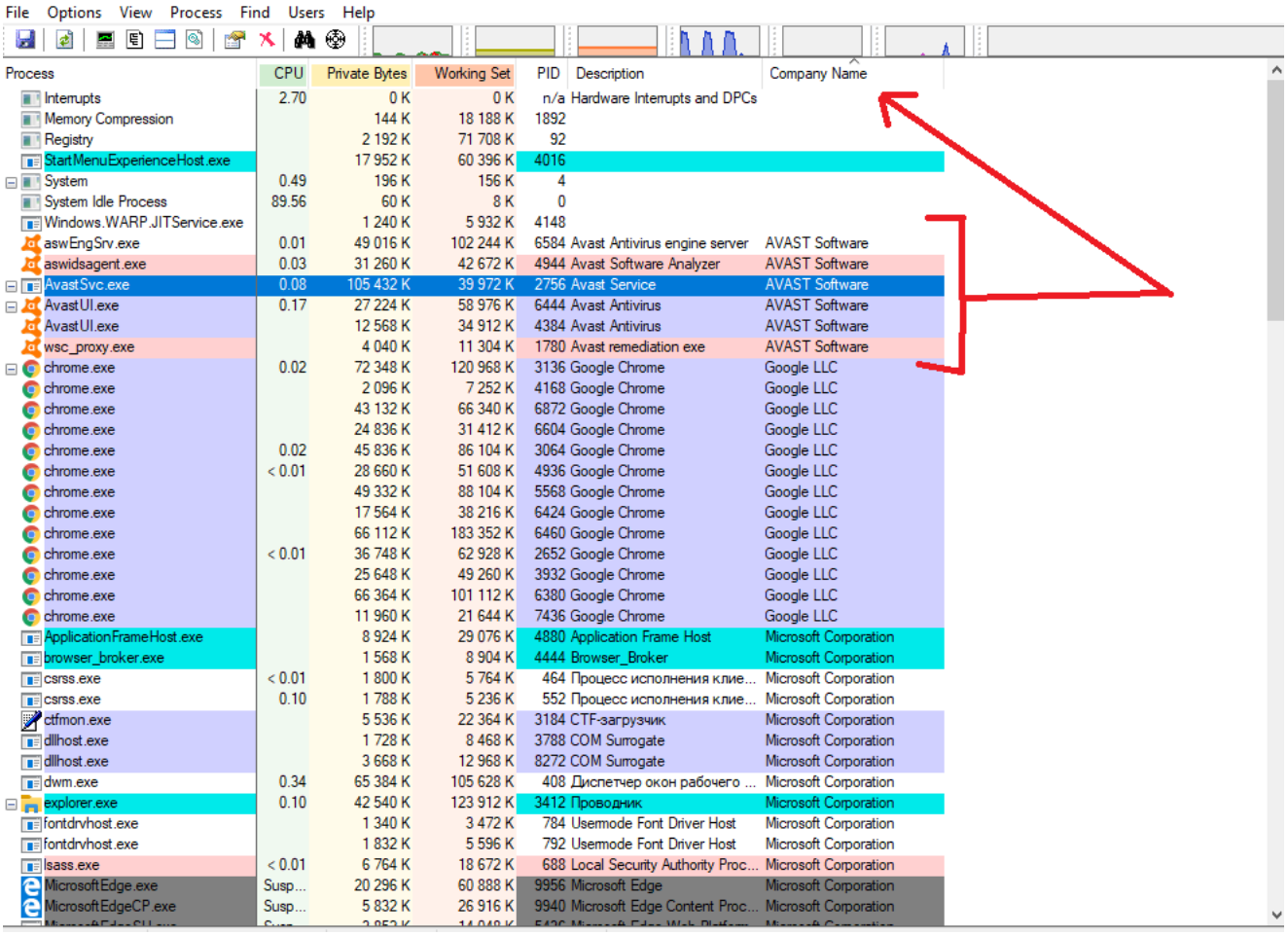
Code:

```
// Массив портов которые детектим
var ports = [
    12025, 12110, 12119, 12143, // AVG + AVAST
    49676 // KASPERSKY
];

// Сюда складируются тайминги запросов
window.ports_timings = {}

// Функция получения таймингов портов
function checkPorts(ports) {
    self.addr = "wss://127.0.0.1";
    self.timeout = 1500;
    self.i = 0;
    function onopen(_) {
        console.log("Open:", self.port);
    }
    function onerror(_) {
        var timeAfterStart = Date.now() - self.startTime;
        console.log("Error:", self.port, timeAfterStart)
    }
    function onclose(_) {
        var timeAfterStart = Date.now() - self.startTime;
        console.log("Close:", self.port, timeAfterStart);
        window.ports_timings[self.port] = timeAfterStart;
        if (i !== ports.length)
            check(ports[i++]);
    }
    function ontimeout() {
        var timeAfterStart = Date.now() - self.startTime;
        if (timeAfterStart > self.timeout) {
            console.debug("Timeout: ", self.port)
            close();
        } else {
            setTimeout(ontimeout, 10);
        }
    }
    function close() {
        self.isDone = true;
        if (self.ws !== null) {
            self.ws.close();
            self.ws = null;
        }
    }
    function check(port) {
        console.debug("Check port: " + port);
        try {
            self.ws = new WebSocket(self.addr + ":" + port);
            self.ws.onopen = onopen;
            self.ws.onerror = onerror;
            self.ws.onclose = onclose;
```

```
            self.port = port;
            self.startTime = Date.now();
            setTimeout(ontimeout, 5);
        } catch (exc) {
            console.error(exc.message);
        }
    }

    check(ports[i++]);
};

// Функция получения открытых портов
function getOpenPorts() {
    var opened = [];
    for (i in window.ports_timings)
        if (window.ports_timings[i] < 500) opened.push(i);
    return opened;
}

// Запуск сканирования портов
checkPorts(ports);
```

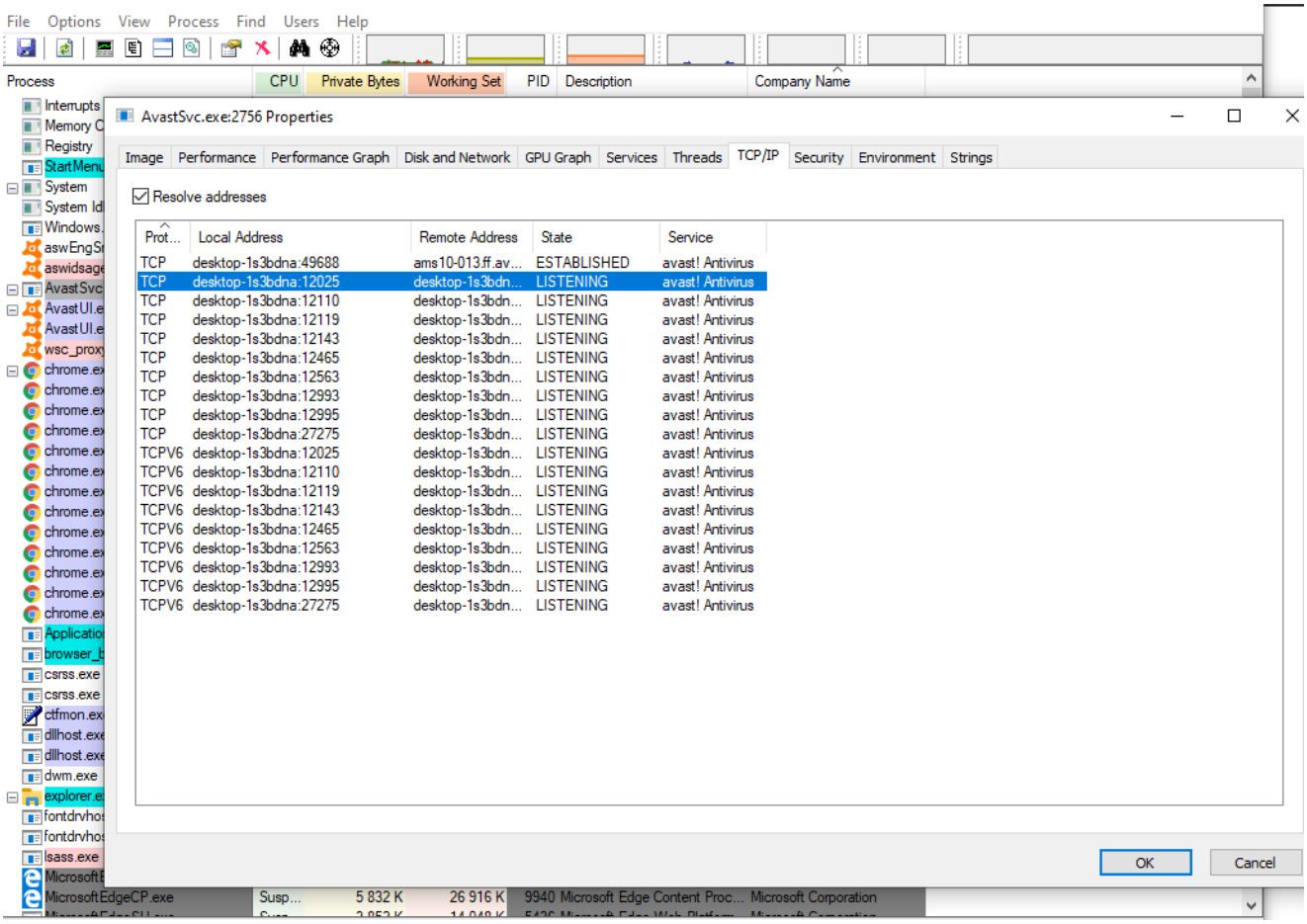
P.s. - я пытался реализовать скрипт асинхронным, чтобы можно было одновременно детектить сразу несколько детектов, а не по очереди как это работает сейчас в синхронном режиме, но на практике если делать детекты асинхронными, то можно словить несколько ложных срабатываний.

Как исследовать АВ (или любой другой софт) на наличие детекта на Windows
• Качаем софт Process Explorer.
• Запускаем Process Explorer с правами администратора (ПКМ > запуск от имени администратора)
• Находим все процессы программы которую исследуем на этот детект. В колонке Company Name видим название компании к которой принадлежит запущенный процесс.



• Дважды кликаем по интересующему нас процессу, открывается окно с свойствами данного процесса, открываем вкладку TCP/IP

• Ищем порты, что имеет статус LISTENING как на скрине ниже с примером на главном процессе аваста. LISTENING - означает что аваст открывает порт на самом ПК и прослушивает его, именно такие порты мы и можем детектить.



• Далее просто берём и записываем в моём JS скрипте в массиве Ports нужный нам порт для детекта.

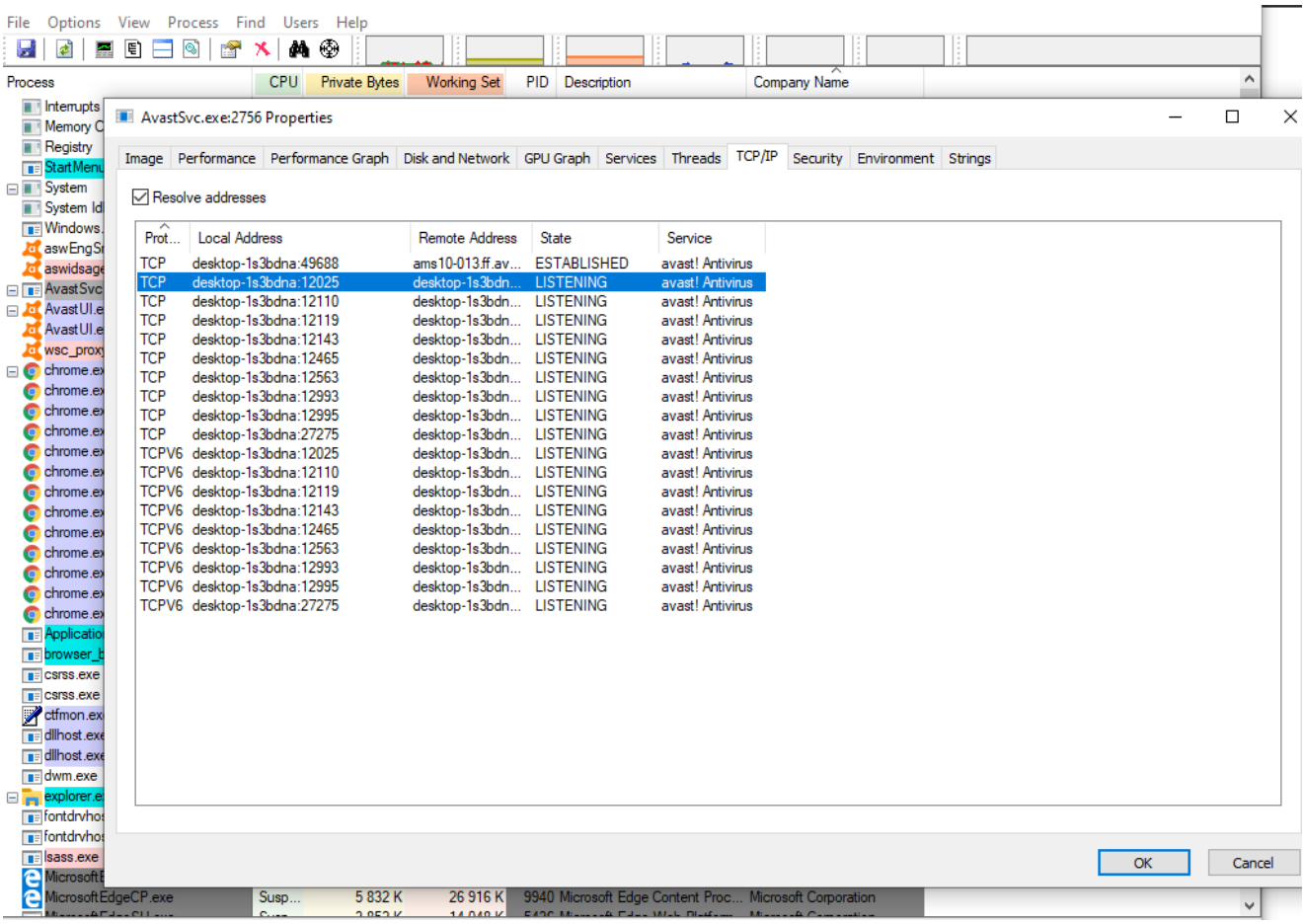
Тестирование AB

Я взял десяток популярных AB для теста, раскатил их на изолированных виртуалках и начал тестить

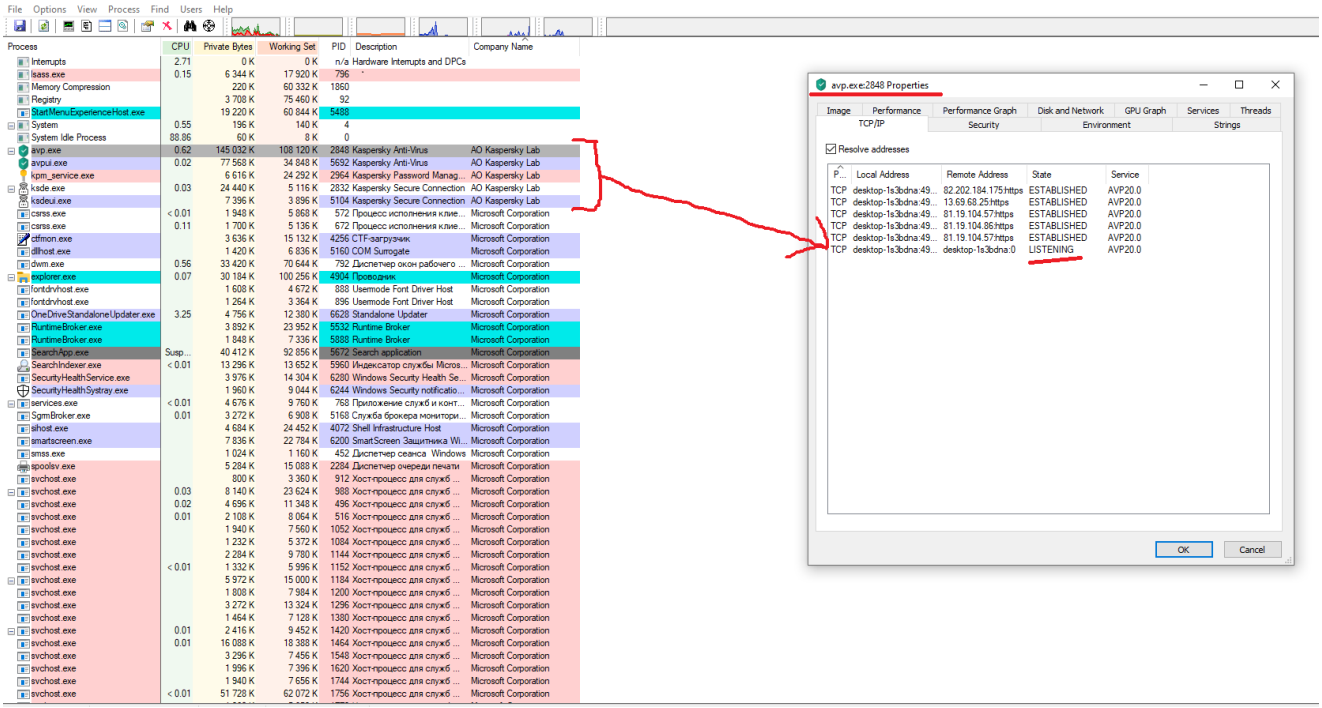
Список AB учавствующих в моих тестах - Kaspersky, Avast, AVG, Node32, Qihoo360, MacAfee, Comodo, Avira, DrWeb. (оттестированы были самые базовые редакции ав)

Результаты: из протестированых AB открытые порты на прослушивание устанавливают Kaspersky, Avast и AVG. Причём с последними двумя интересная картина произошла т.к. у них одинаковые процессы и они оба открывают одинаковые порты. Сначало меня это привело в ступор, но потом я вспомнил что Аваст когда-то покупал AVG и видимо они как-то унифицировали свои интерфейсы. Таким образом мы хоть и можем задетектить Аваста и АВГ, но кто именно из них был задетекчен понять нельзя. Все остальные AB не открывали порты на прослушивание.

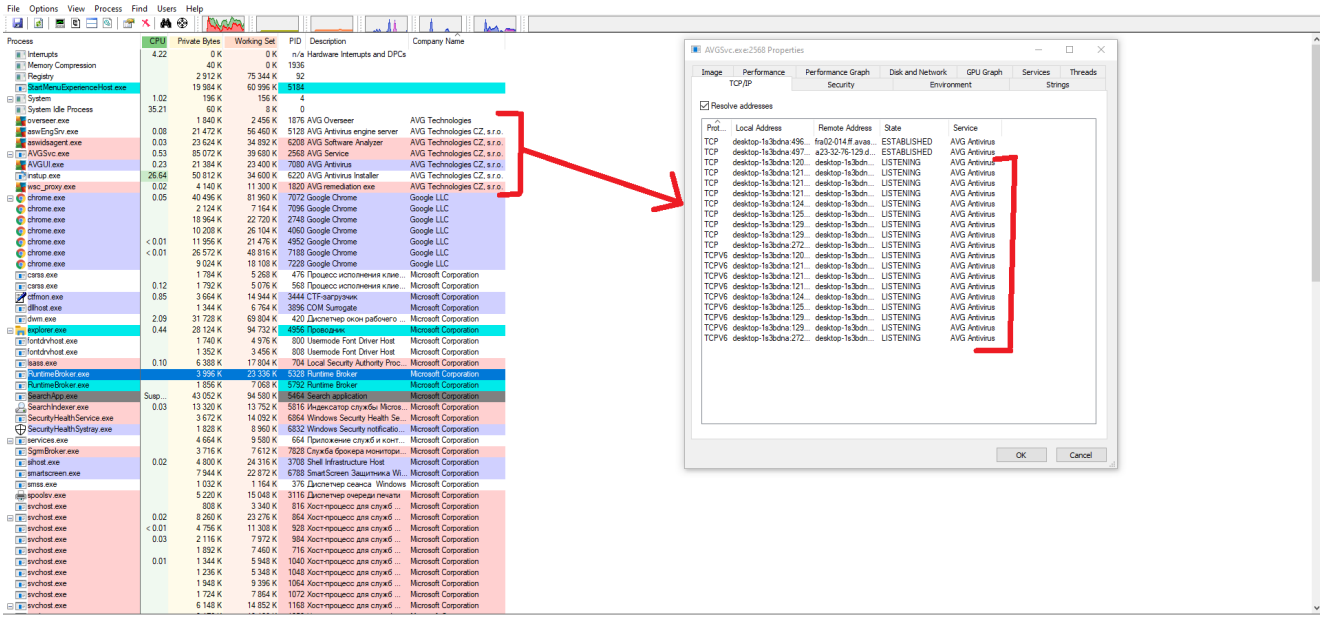
Мейн процесс Аваста (AvastSvc.exe) открывает на прослушивает много разных портов, я взял четыре из них (12025, 12110, 12119, 12143) и добавил в скрипт



Касперский открывает один порт на прослушивание (49676)



Процессы АВГ аналогично авасту, разницы между ними нету (12025, 12110, 12119, 12143)



Выводы

В целом ресёрч выдался удачным, так как удалось реализовать надёжный детект касперского и аваста/авг. Это два самых популярных АВ в СНГ, а Аваст на сколько я знаю вообще самый популярный ав во всём мире. Так-же я тестировал только самые базовые редакции антивирусов, к примеру у касперского я взял из AntiVirus Free, я не

тестировал Kaspersky Internet security/Premium и т.п. из-за этого потанциально другие AB открывают порты, но в расширеных своих редакциях, со всякими там фаерволами и т.п. Я тестировал именно базовые редакции с мыслью о том, что если в базе есть детект то он будет и в расширеных версиях AB. Расширеные версии детектить я не стал так как это заняло бы гораздо больше времени.
Так-же стоит не забывать что с помощью этого готового кода и мануала можно детектить любой другой софт установленный на компе юзера который открывает порт на прослушивание. К примеру вы хотите атаковать таргет эксплуатируя какой-то не по умолчанию устанавливаемый на всех машинах софт, предположем торрент. То при переходе таргетом по вашей ссылке у человека детектится наличие utorrent-а на компе и ему отдаётся *.torrent, вместо *.exe, а если детекта торрента не произошло то можно сразу отдавать *.exe. Тем самым не будет потерян в пустую ценный момент атаки в случае если у юзера нету торрент клиента, а мы ему отдаём торрент файл. То есть можно придумать много полезных применений такой технике детекта.

P.s. - если кому-то в голову придёт ещё какой-то интересный концепт детекта или мысли как улучшить то что я написал тут в статье, но вы к примеру сами не можете реализовать их и не хотите выносить это на публику, то дайте знать в лс, я бы мог проверить ваши идеи лично и взамен поделиться с вами результатами.

UPD - эта методика была проверена на Google Chrome, там она отрабатывает надёжно.
Так-же всё должно работать на Chromium based браузерах (Opera, Yandex, Edge)
Так-же надёжно работает в старом Edge не на движке хрома
Не работает в Firefox (там не предсказуемые тайминги)
IE не тестил.
Last edited: Jun 27, 2020

Первая ссылка на которую ты ссылаешься использует Флеш. Флеш давно мёртв, это даже в теории работать не будет. Помянем

Вторая ссылка на которую ты ссылаешься (https://vah13.github.io/AVDetection/) Я её гуглил год назад и уже год назад почти все детекты что там были не работали. А теперь по подробнее насчёт этого:

Сейчас развернул опять тестовую среду и затестил 3 из 5 AB (+Аваст) которые якобы детектит эта либа
1- Касперский, вместе с ним устанавливается расширение в хром (https://chrome.google.com/webstore/detail/kaspersky-protection/elhpdacimkjpccooodognopfhbdgnpbk), само расширение ничего на страницу не инжектит, а значит детект с этой гитхаб либы вида
Code:

```
ka = frm.contentDocument.getElementsByTagName('html')[0].outerHTML;
if (ka.indexOf("kasperskylab_antibanner") !== -1)
{
        alert("AV name is Kaspersky");
        return;
}
```

Не отработает. Стоит отметить что раньше он детектил инжект в тег HTML о чём я писал выше в одном из комментариев к статье, когда я тестил пару лет назад этот детект ещё существовал, сейчас возможности детектить каспер через расширения нету.

2- Аваст, вместе с ав устанавливается расширение (https://chrome.google.com/webstore/detail/avast-online-security/gomekmidlodglbbmalcneegieacbdmki), само расширение ничего на страницу не инжектит, а значит детект через расширения невозможен.

3- Dr web, вместе с ав не устанавливаются какие либо расширения. А значит детект вида
Code:

```
if (document.getElementsByClassName('drweb_btn').length > 0)
{
        alert("AV name is DrWeb");
        return;
}
```

у этой либы если даже и работает то полностью бессмыслен.

4- Avira, вместе с ав устанавливаются 2 расширения. Я вчера тестируя Авиру не увидел что она ставит расширения потому что она поставила их после перезагрузки ОС, из-за этого эти расширения пролетели мимо меня, это очень странно на самом деле ну да ладно.
Какой детект у этой либы?
Code:

```
var AV = document.getElementById("abs-top-frame")
if (AV!==null)
{
if ( AV.outerHTML.indexOf('/html/top.html')>=0 & AV.outerHTML.indexOf('chrome-
extension://')>=0 )
{
        alert("AV name is Avira");
  return;
}
}
```

И он тоже не отработает, потому что расширения авиры уже не инжектят элемент с id "abs-top-frame" в страницу.
НО они теперь инжектят в DOM свой узел с css классом abineContentPanel в котором есть айфрейм с классом abileContentFrame, по этому её можно детектировать по расширениям. НО учитывая что её расширение установило всего 3млн людей, это на фоне всего мира как то не очень. Но это в любом случае лучше чем ничего. Линки на расширения авиры - https://chrome.google.com/webstore/detail/avira-password-manager/caljgklbbfbcjjanaijlacgncafpegll и https://chrome.google.com/webstore/detail/avira-safe-shopping/ccbpbkebodcjkknkfkpmfeciinhidaeh

В либе остались детекты Битдефендера и Нортона, но их я затестить не могу, эти 2 ав никак не хотят устанавливаться через тор, увы.

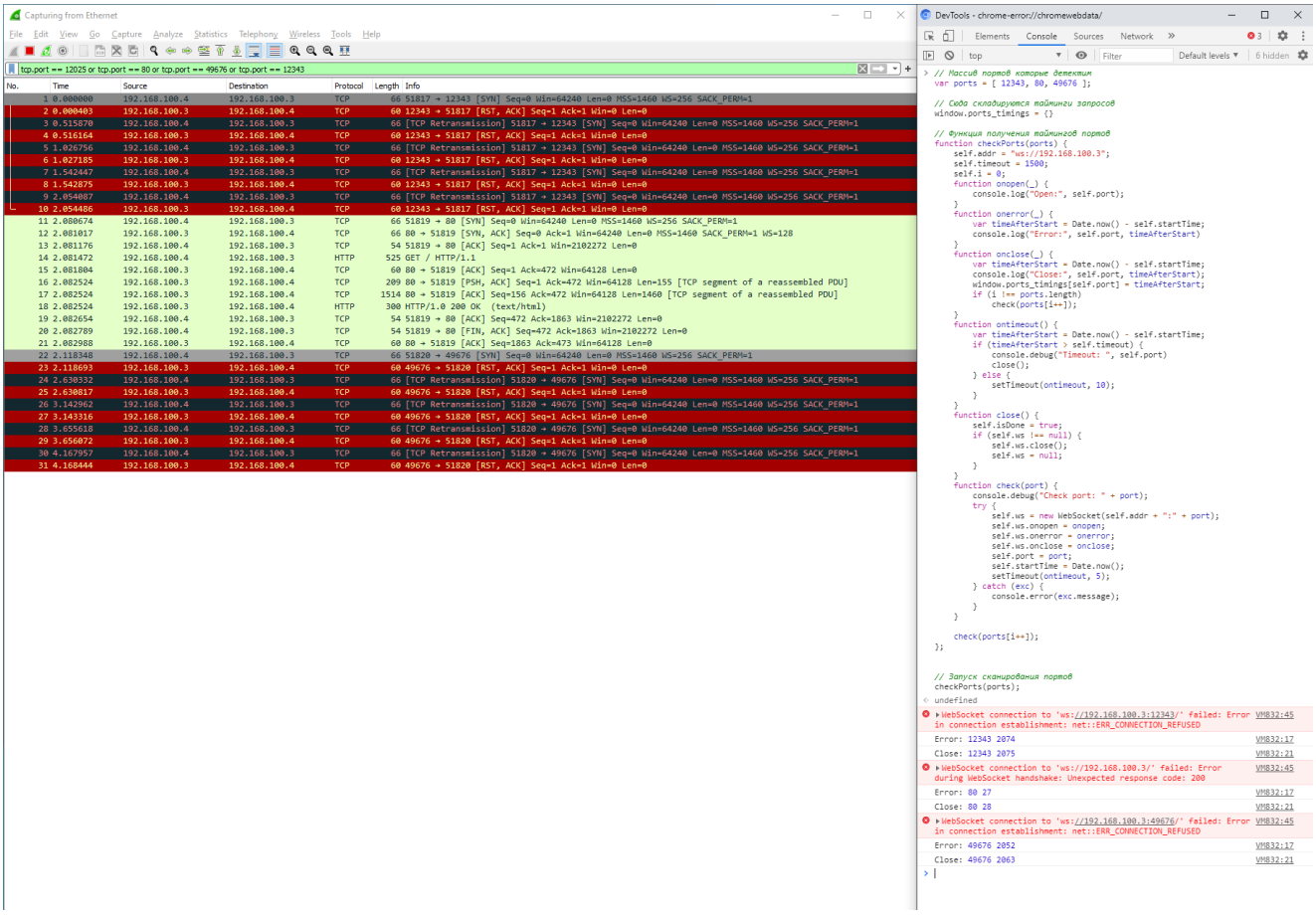
И последнее - ты привёл в пример детект с помощью Бифа в видео на ютубе, но Beef то юзает именно эту **не** рабочую либу с гитхаба, а значит этот модуль бифа тоже на данный момент не рабочий. Пруф: переходим по линку (https://github.com/beefproject/beef...194/modules/host/detect_antivirus/config.yaml) и жмём CTRL+F вбиваем "authors: ["phosphore","vah13","nbblrr"]" И мы видим что этот модуль за авторством vah13 которому и принадлежит гитхаб этой либы с нерабочими детектами https://vah13.github.io/AVDetection/
Все остальные АВ что я тестил для статьи, точно (я уверен на 99%, с авирой подстава получилась) не добавляют в браузер свои расширения, а значит их методикой #2 которую я описываю в статье не задетектить.
Единственное что стоит проверить Битдефендера и Нортона кому это нужно, вполне возможно что эти ав до сих пор могут идти с расширениями из коробки которые что-то инжектят в DOM веб страниц. Я так-же припоминаю что когда я тестил эту либу год назад, в ней отработал только Битдефендер.

P.s. надеюсь этим дополнительным мини ресёрчем я закрыл все вопросы связанные с детектом АВ через их браузерные расширения.
Так-же стоит отметить что судя по всему детекты Касперского и Аваста/АВГ на данный момент возможны только с помощью моей методики что я описал в статье, то есть фактически это уникальная разработка.

Чтобы ответить на эти вопросы мне пришлось сделать ещё один “мини ресёрч” который отнял у меня не один час, из-за этого собственно и не ответил сразу. Вопросы которые закрыл этот дополнительный ресёрч.
1- Почему по факту если порт открыт, то задержка создания и закрытия соединения меньше в разы чем если порт закрыт? Тут возникает диссонанс т.к. по логике если порт закрыт то соединение должно максимально быстро обрываться и тем самым

закрытые порты должны иметь меньшую задержку чем открытые, но происходит ровно наоборот.
2- Можно ли этой методикой сканить локальную сеть юзера через его браузер? То есть не сканить на открытые порты комп юзера (пример - 127.0.0.1:4343), а порты на других IP в пределах локальной сети юзера (пример - 192.168.100.3:445).

И так, я расскатил 2 виртуалки на Windows 10 x64, первая имеет локальный IP - 192.168.100.4, вторая 192.168.100.3. На первой виртуалке мы запускаем браузеры и в них наш JS скрипт, который будет обращаться через сокеты на IP второй виртуалки. На второй виртуалке открыт на прослушивание только 80 порт. Обе виртуалки хоть и запущены на одном ПК, но объединены в локальную сеть. Далее начал снифать запросы ваершарком. Вся нужная инфа на одном скрине (если он у вас здесь не отображается, откройте в новой вкладке прикреплённый к теме PNG с именем "Capture_ws.PNG")



Как видно из скрина я запустил скрипт который делает 3 вебсокет запроса, с нашей первой виртуалки (192.168.100.4) на вторую (192.168.100.3), 3 последовательных запроса на 3 разных порта. На 192.168.100.3:12343, 192.168.100.3:80 и 192.168.100.3:49676. Из этих портов открыт только 80, на нём висит обычный веб-сервер который не умеет обрабатывать сокеты. В ваершарке видно что запросы на

порты (первый и третий) 12343 и 49676 выполнялись по 5 раз. Как видно на скрине первым запросом (номера запросов видны в колонке No.) браузер создаёт TCP соединение и инициализирует TCP рукопожатие, отправляя TCP пакет с флагом SYN на IP 192.168.100.3:12343, на что в следующем запросе сервер дропает пакет и отправляет клиенту ответ с TCP флагом RST (reset) то есть сброс соединения, далее клиент делает вывод что возможно что-то пошло не так и делает ещё несколько попыток повторяя в точности запрос, в случае хрома - ещё 4 попытки. Сервер каждый раз обрывает соединение отправляя пакет TCP ответ с флагом RST. В совокупности все эти 5 запросов и ответы на них длятся 2054 миллисекунды. Так-же в консоли браузера мы видем что JS вернул ошибку соединения “Error: 12343 2074”, где 12343 порт на который мы пытались создать вебсокет соединение, а 2074 миллисекунды сколько прошло он начала инициализации вебсокета до полного закрытия соединения (после 5ти попыток создать TCP соединения) + задержка в обработке ошибки самим браузером (20 миллисекунд). Стоит отметить что в данном случае клиент прежде чем создавать повторный TCP SYN запрос на каждую попытку вновь создать соединение, выжидает где-то 0.4-5 сек и только после этого делает новый запрос, сервер же ему каждый раз отвечает ресетом за 0.001 сек. Именно из-за задежки которую выжидает клиент перед каждой попыткой создать TCP соединение и создаётся возможность детектировать открытые/закрытый порты по задержке которую использует эта методика. Стоит так-же уточнить что все другие браузеры имеют такое же поведение, но к примеру IE с Edge (не на движке хрома) делают 3 повторных запроса, вместо 5ти из-за этого средняя задержка в этих браузерах при закрытом порту 1сек вместо 2сек. В Firefox точно не помню, во время тестов забыл это заскринить, но там вроде бы было тоже 5 запросов.

Теперь посмотрим на запросы которые “зелёные”, это запросы на 80 порт. Если в кратце - клиент с сервером успешно создают TCP соединение в котором клиент отправляет стандартный GET где указывает заголовок апгрейда протокола до вебсокетов (11-14 запросы). Сервер же просто отвечает 200 HTTP кодом и отдаёт html страницу (15-18 запросы), что в протоколе вебсокетов является не корректным ответов из-за чего клиент завершает TCP соединение с сервером (19-20 запросы) и сервер отвечает что принял завершение TCP соединения (21 запрос). Далее клиент сразу выдаёт ошибку в консоли хрома “Error: 80 27”. На все запросы в данном случае ушло 27 миллисекунд что так-же подтверждается ваершарком, на столько быстро отработали все запросы потому что хоть я и эмулировал локалку создав две виртуалки и обьединив их в локальную сеть, но все запросы всё равно крутятся в пределах одной машины, из-за этого нету дополнительных задержек которые нужны скорости света для того чтобы в реальной локальной сети пролететь от одного пк к другому по езернету или вафле, но они в любом случае всегда будут в разы меньше при запросах на открытый порт, нежели при закрытый.

Если посмотреть следующие запросы тоже на закрытый порт 49676 то видем полностью аналогичную картину как при первых запросах на закрытый порт 12343, которые я обсудил выше.

Далее ответ на второй вопрос. Можно ли сканировать локалку такой методикой? Да, можно, нету никакой причины почему это могло бы не работать в локалке. НО, я считаю попытки засканить локальный IP адреса через браузер юзера - юзлесс. На то есть две причины
1- Судя по всему эти запросы нельзя делать ассинхронными, иначе будут сбиваться тайминги и мы не сможем детектить из-за чего мы вынуждены тратить на каждый запрос в среднем 1.5-2 секунды. Учитывая что юзер на нашем сайте в большинстве случаев будет находиться не более 1 минуты (что очень оптимистично), то мы сможем создать всего 30 вебсокет соединений с разными IP или портами. Что является очень малым количеством запросов для скана локалки.
2- Даже если как-то умудриться сделать скрипт ассинхронным (в чём я сильно сомневаюсь), то мы сможем увеличить скорость только в 8 раз потому что браузеры ограничивают одновременное кол-во запросов с браузера юзера в одном домене. И вот это уже никак по идее не обойти, потому что даже различные ресурсы (CSS, картинки и т.п) имеют то-же самое ограничение (я не уверен насчёт восьми, но где-то около того). По итогу увеличение скорости скана даже в 8 раз ничего нам не даёт, так как скорость всё равно остаётся очень низкой.

Отсюда вывод: пытаться сканить локальные IP через браузер юзера есть смысл только в одном случае, если вы точно знаете какие IP и порты к ним нужно сканить. Из-за этого я не вижу смысла в таком применении этой методики. Но вот мой JS код который для теста скана локалки, в нём вы можете задать один IP и массив портов к нему которые нужно просканить. Я не стал писать скрипт в который можно задавать срару список IP, потому что не вижу в этом практического применения и смысла тоже, те кому надо могут немного переделать скрипт под эту задачу.

Code:

```
// IP который сканим
var ip2scan = "192.168.100.3";

// Массив портов которые детектим на указанном IP
var ports = [
    20
];

// Сюда складируются тайминги запросов
window.ports_timings = {}

// Функция получения таймингов портов
function checkPorts(ip, ports) {
        var obj = {};
        obj.ip = ip;
    obj.proto = "ws";
    obj.timeout = 1500;
    obj.pi = 0;
    function onopen(_) {
        console.log("Open:", obj.port);
    }
    function onerror(_) {
        var timeAfterStart = Date.now() - obj.startTime;
        console.log("Error:", obj.port, timeAfterStart)
    }
    function onclose(_) {
        var timeAfterStart = Date.now() - obj.startTime;
        console.log("Close:", obj.port, timeAfterStart);
                if (!(obj.ip in window.ports_timings))
                        window.ports_timings[obj.ip] = {}
        window.ports_timings[obj.ip][obj.port] = timeAfterStart;
        if (obj.pi !== ports.length)
            check(ports[obj.pi++]);
    }
    function ontimeout() {
        var timeAfterStart = Date.now() - obj.startTime;
        if (timeAfterStart > obj.timeout) {
            console.debug("Timeout: ", obj.port)
            close();
        } else {
            setTimeout(ontimeout, 10);
        }
    }
    function close() {
        obj.isDone = true;
        if (obj.ws !== null) {
            obj.ws.close();
            obj.ws = null;
        }
    }
    function check(port) {
```

```
        console.debug("Check port: " + port);
        try {
            obj.ws = new WebSocket(obj.proto + "://" + obj.ip + ":" + port);
            obj.ws.onopen = onopen;
            obj.ws.onerror = onerror;
            obj.ws.onclose = onclose;
            obj.port = port;
            obj.startTime = Date.now();
            setTimeout(ontimeout, 5);
        } catch (exc) {
            console.error(exc.message);
        }
    }

    check(ports[obj.pi++]);
};


// Запуск сканирования портов
checkPorts(ip2scan, ports);
```

Отдельно я отресёрчил ещё один вопрос, который просто забыл отресёрчить в изначальном ресёрче.
Суть в том что в браузерах (Хроме и Firefox) есть black-листы, кхм, извиняюсь, block-листы портов. То есть эти браузеры имеют список портов на которые делая запрос - они не выполнят запрос в принципе и выдадут ошибку. Видимо это сделано в целях защиты и в этих списках есть дефолтные порты популярных протоколов/сервисов (SSH, Telnet, RDP и т.п). То есть обращаться к этим портам не имеет никакого смысла. Пример: запрос на 127.0.0.1:22 выдаст всегда ошибку и запрос даже не попытается выполниться, в том числе если обратиться на 22 порт на любой IP, в том числе и не локальные.

Списки портов в блок листах браузеров
1- Chromium based браузерах (Chrome, Yandex, Opera и вероятно новый Edge на движке хрома) - список найден в исходнике Chromium (https://github.com/chromium/chromiu...f722090104e0e491d6bf071/net/base/port_util.cc) виден весь список блок портов.
2- Firefox - список портов в офф доке https://developer.mozilla.org/en-US/docs/Mozilla/Mozilla_Port_Blocking
3- IE 11 - не лочит порты
4- Edge (не на движке хрома) - аналогично IE, не лочит порты.

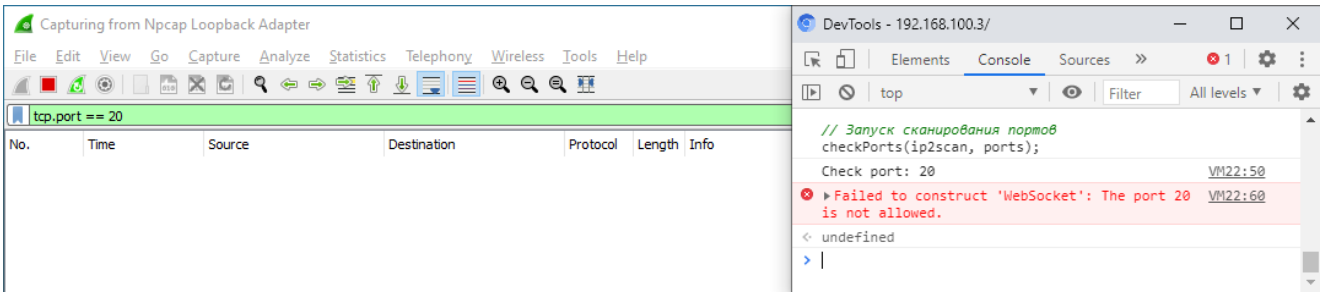
Я хотел убедиться в этом лично делая два запроса из каждого браузера, первый на 127.0.0.1:20, второй на виртуалку в локалке 192.168.100.3:20. В итоге Chrome и Firefox даже не делали сетевых запросов, а просто выдавали ошибку. А IE с Edge делали

запросы как обычно и выдавали ошибку уже создания вебсокет сессии. Всё это видно в скринах ниже. Отсюда следует вывод, что делать запросы в Firefox и Chromium based браузерах на заблокированные порты - нету никакого смысла, на любой IP адрес, будь он локалхостом, в локальной сети или в интернете.

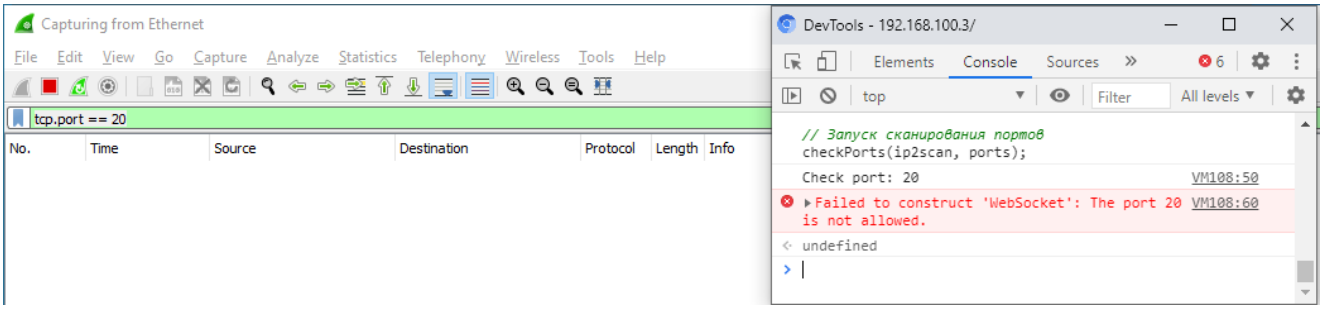
Если у вас ниже не отображаются скрины этого теста, смотрите их в закреплённых файлах, приставка Local на скрине - это запрос на 127.0.0.1:20 а приставка LAN это запрос на 192.168.0.3:20.

Google Chrome

Запрос на 127.0.0.1:20. Выдало ошибку, сетевых запросов сделано не было.

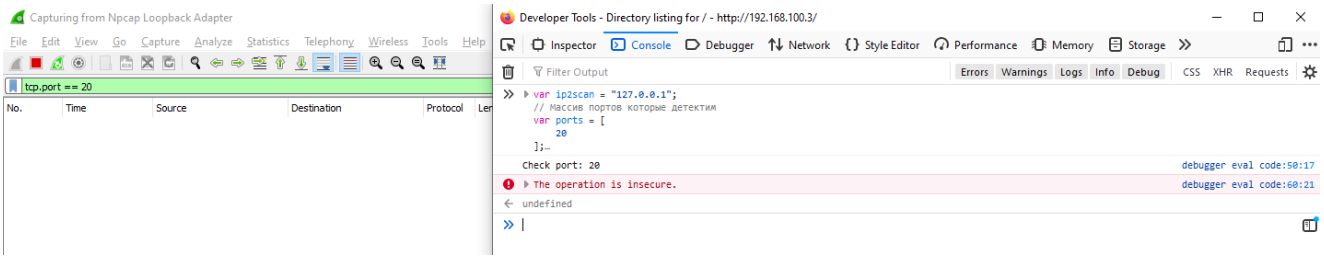


Запрос на 192.168.100.3:20. Выдало ошибку, сетевых запросов сделано не было.

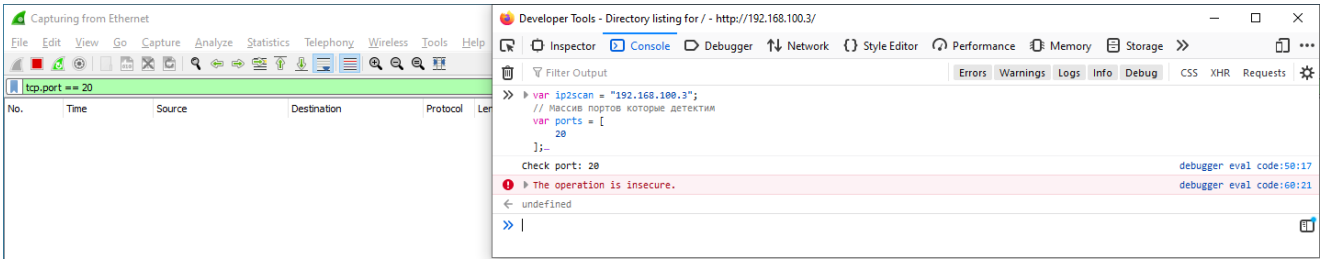


Firefox

Запрос на 127.0.0.1:20. Выдало ошибку, сетевых запросов сделано не было.

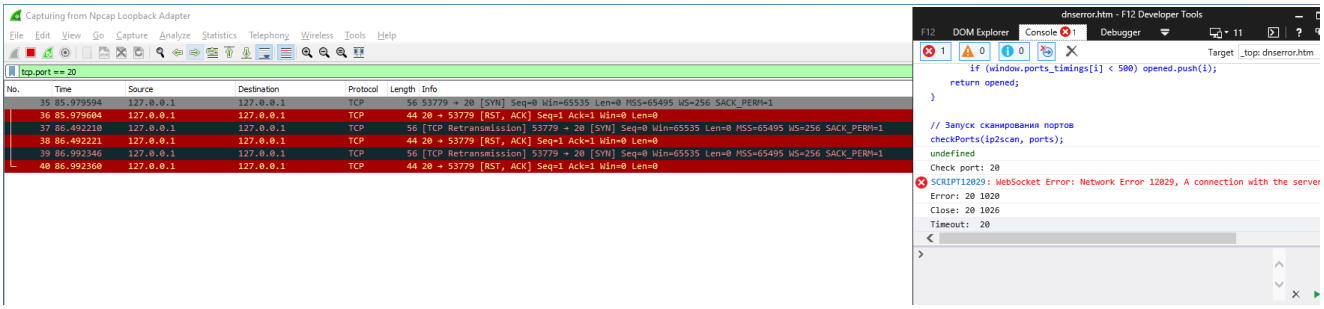


Запрос на 192.168.100.3:20. Выдало ошибку, сетевых запросов сделано не было.

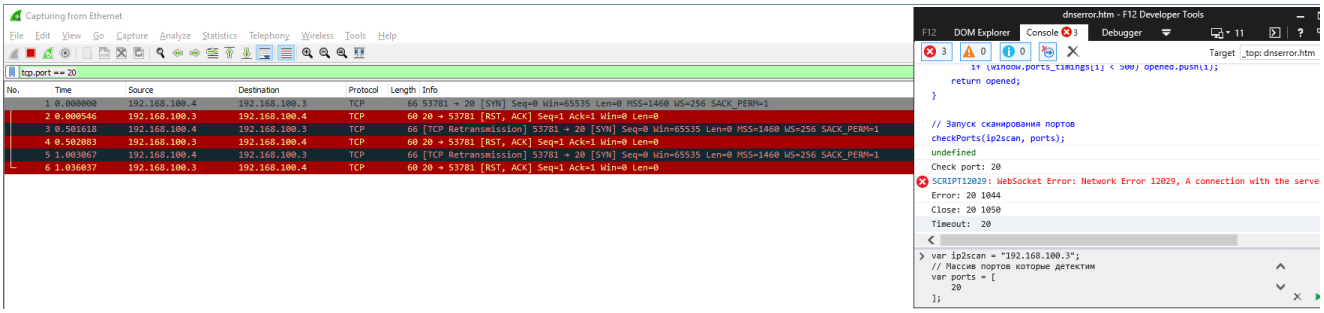


IE 11

Запрос на 127.0.0.1:20. Выдало ошибку создания веб-сокет соединения, в ваершарке видны попытки создать TCP соединение, порт не блокируется браузером.

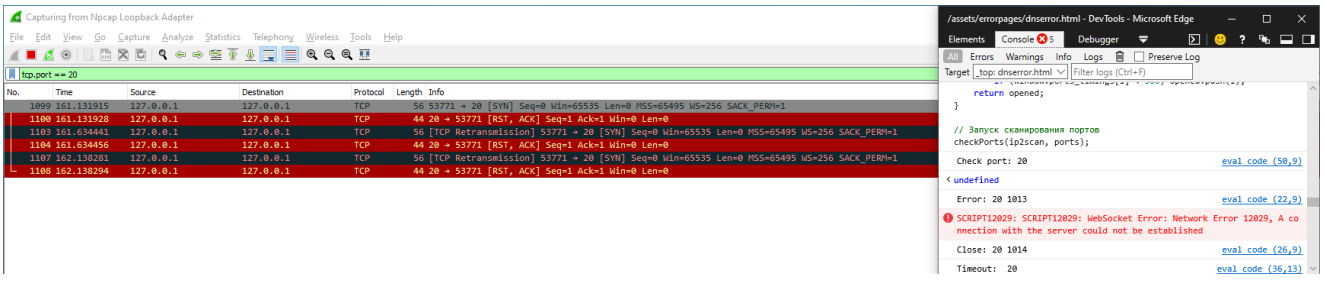


Запрос на 192.168.100.3:20. Выдало ошибку создания веб-сокет соединения, в ваершарке видны попытки создать TCP соединение, порт не блокируется браузером.

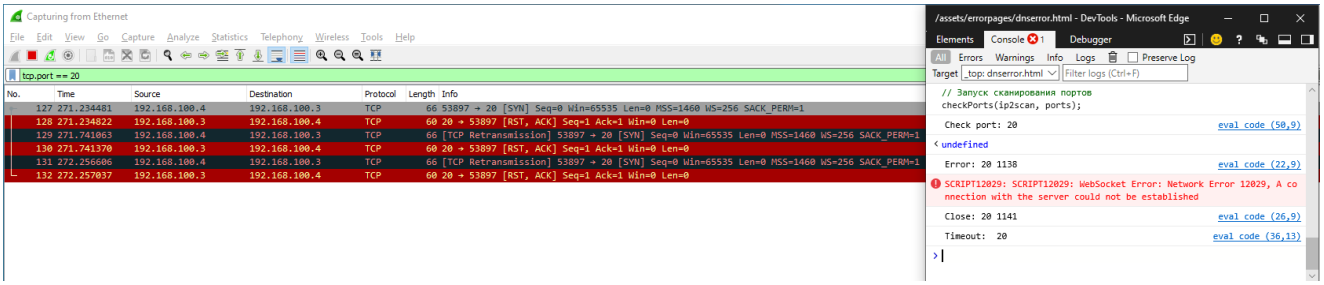


Edge (не на движке хрома)

Запрос на 127.0.0.1:20. Выдало ошибку создания веб-сокет соединения, в ваершарке видны попытки создать TCP соединение, порт не блокируется браузером.



Запрос на 192.168.100.3:20. Выдало ошибку создания веб-сокет соединения, в ваершарке видны попытки создать TCP соединение, порт не блокируется браузером.



ВАЖНО!
Методика работает во всех браузерах, в том числе и Firefox, так-же всё работает если делать запросы по обычному сокету (WS), а не только по защищённому (WSS).
Почему-то при моих первичных тестах я сделал не корректные выводы основанные на моём старом тестовом коде который видимо не корректно работал в Firefox и при использовании обычных WS сокетов, а когда я его уже отрефакторил и немного изменил для того чтобы выложить в главном топике этой темы - он пофиксился. Ну это классика чё тут скажешь. Я невероятно ахринел когда мой-же код из моей же темы корректно отработал на Firefox и с обычными WS сокетами. Из-за чего в главном топике темы где я написал что методика работает только на WSS и то что она не работает в Firefox - ошибка.

Вот теперь можно полноценно сказать что ресёрч этого вопроса полностью закончен.
Last edited by a moderator: Aug 3, 2020