Security

BUILDING FOR THE "*bahd*" GUYS

NEO IGHODARO / @NEOIGHODARO

hotels.ng

WHAT WE WILL *Consider*

QUICK *Overview*

- Web Security
- Server Security

THE *no-brainers.*

- If you use third-party libraries, keep them up to date.
- Don't just use third-party libraries.
- Do not trust anything coming from outside into your app.
- Do not trust anything coming from inside your app.
- Encrypt sensitive data.
- Use complex passwords. https://www.creativitykills.co/create-secure-password-can-remember/

DIVING *Deeper*

SQL Injection

SQL injection attacks are when an attacker uses a web form field or URL parameter to gain access to or manipulate your database.

```
// -- Unsafe -----
"SELECT * FROM table WHERE column = '" + parameter + "';"
// "SELECT * FROM table WHERE column = '' OR '1'='1'

// -- Safe OR use a good DBAL -----
$stmt = $pdo->prepare('SELECT * FROM table WHERE column = :value');
$stmt->execute(array('value' => $parameter));
```

XSS (<Cross Site Scripting />)

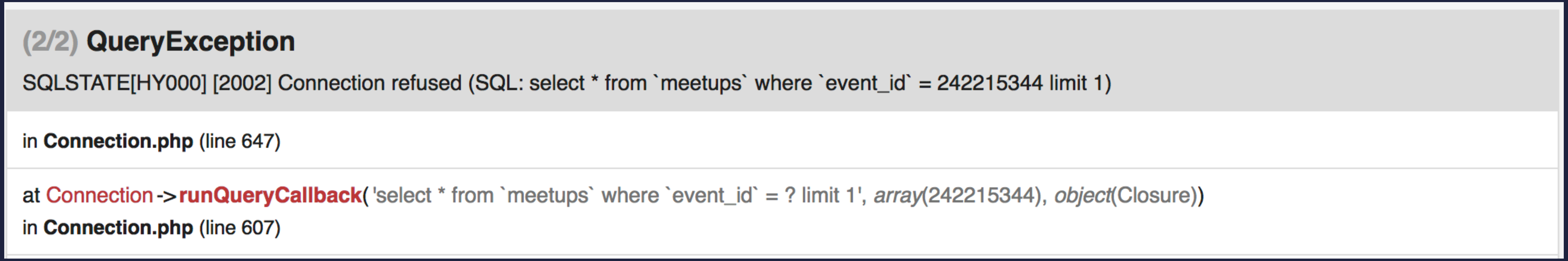
XSS attacks inject malicious JavaScript into your pages, which then runs in the browsers of your users, and can change page content, or steal information to send back to the attacker.

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.

https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

ERROR MESSAGES

Provide only minimal errors to your users, so they don't leak secrets (e.g. API keys or database passwords). Don't provide full exception details either, as these can make SQL injection easier. Keep detailed errors in your server logs.



```
(2/2) QueryException
SQLSTATE[HY000] [2002] Connection refused (SQL: select * from `meetups` where `event_id` = 242215344 limit 1)

in Connection.php (line 647)

at Connection->runQueryCallback('select * from `meetups` where `event_id` = ? limit 1', array(242215344), object(Closure))
in Connection.php (line 607)
```

VALIDATION

Validation should be done on both the client and server side. For simple errors like required fields, use the browser validation but always validate server side as it can be bypassed.

FILE UPLOADS

Allowing users to upload files to your website poses a big security risk. The risk is that any file uploaded, though seemingly innocent, can contain a script that when executed on your server completely opens up your website. Treat all files with suspicion. You can serve all assets from a server without server execution access.

```
deny from all
<Files ~ "^\w+\.(gif|jpe?g|png)$">
order deny,allow
allow from all
</Files>
```

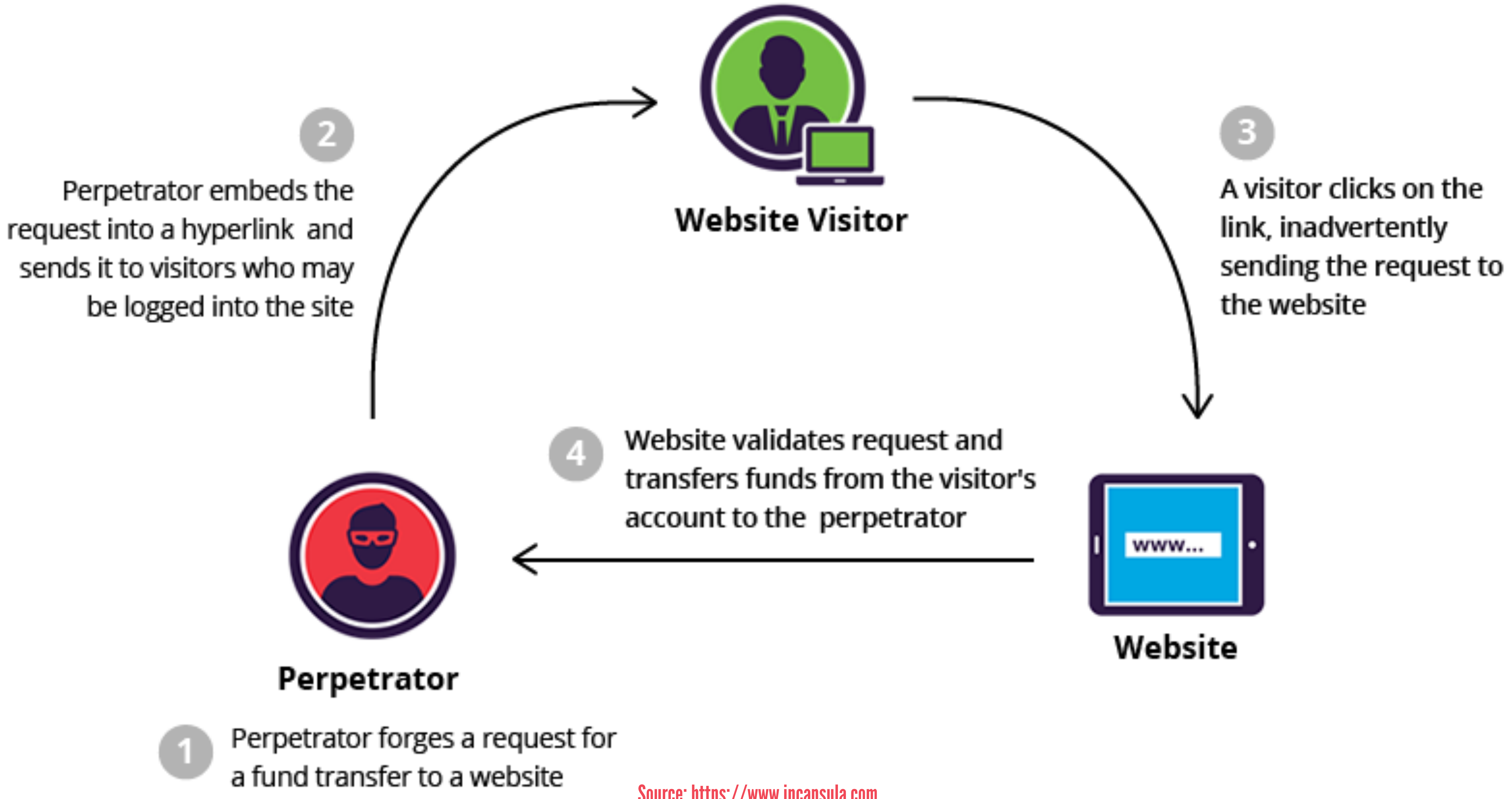
HTTPS *everywhere!*

HTTPS provides security over the Internet. HTTPS guarantees to users that they're talking to the server they expect, and that nobody else can intercept or change the content they're seeing in transit.

For sensitive data, it's highly advisable to use only HTTPS to deliver it. Credit cards, login pages, and typically far more of your site too. Cookies set using HTTPS arr encrypted. Hence, an attacker stealing this would not be able to decrypt or use the cookies.

http://letsencrypt.org/

CSRF (Cross Site Request Forgery)


2
Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
Website Visitor
3
A visitor clicks on the link, inadvertently sending the request to the website
4
Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Perpetrator
Website
1
Perpetrator forges a request for a fund transfer to a website

Source: https://www.incapsula.com

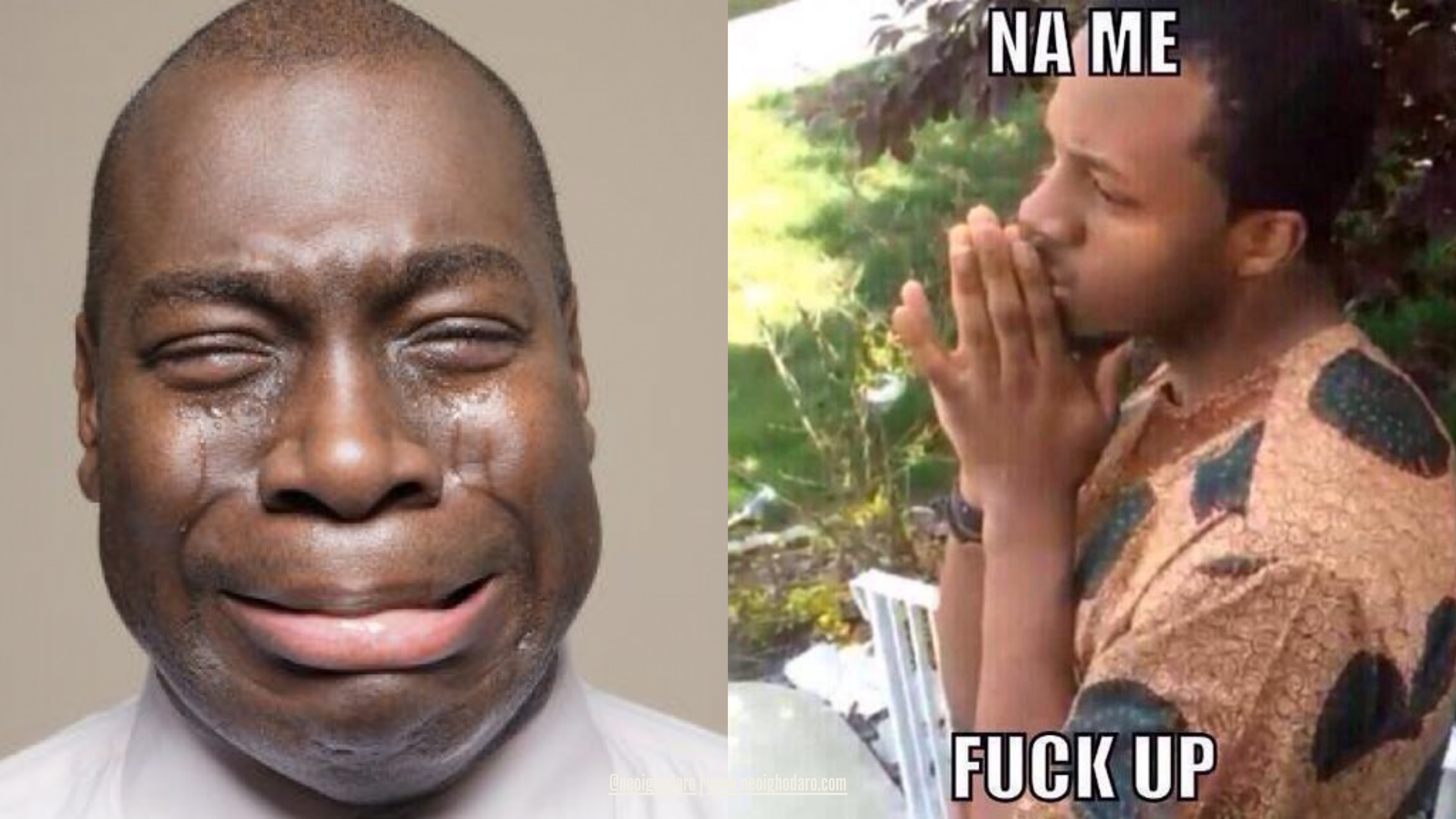
This is something you may not know about the way browser handles sessions...

```php
// deleteuser.php
<?php
// ...
if ($_SESSION['logged_in'] === false) {
    die("You are not logged in!");
}

// ...
deleteUser($_GET['user_id']);
```

and then the attacker does this to a logged in user from his own site...

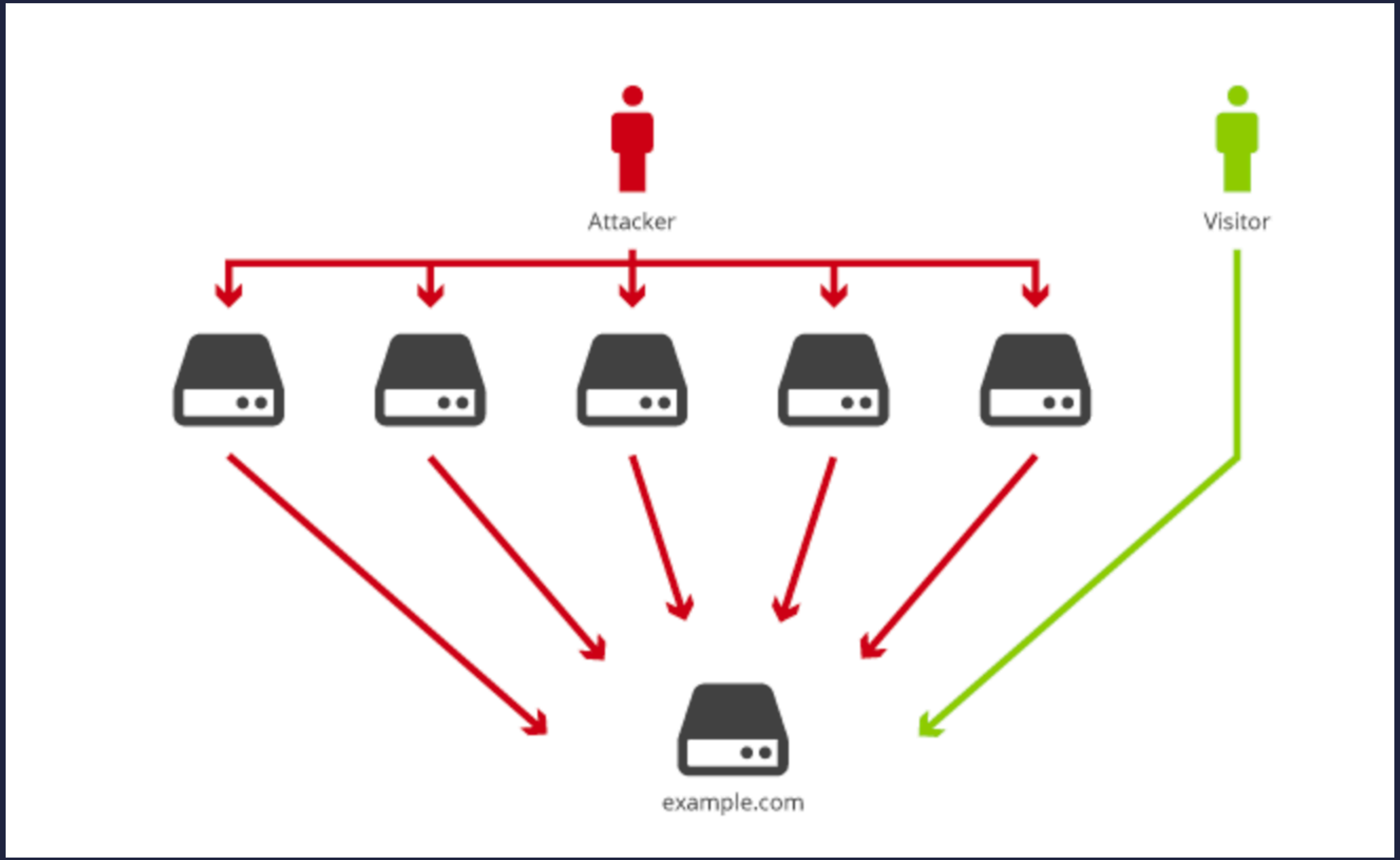
```html
<img src="http://penalty-to-throwing.com/deleteuser.php?user_id=1" title="Oops!" />
```


NA ME
FUCK UP

DDOS (*Distributed Denial of Service*)

In a DDoS attack, the incoming traffic flooding the victim originates from many different sources – potentially hundreds of thousands or more. This effectively makes it impossible to stop the attack simply by blocking a single IP address; plus, it is very difficult to distinguish legitimate user traffic from attack traffic when spread across so many points of origin.

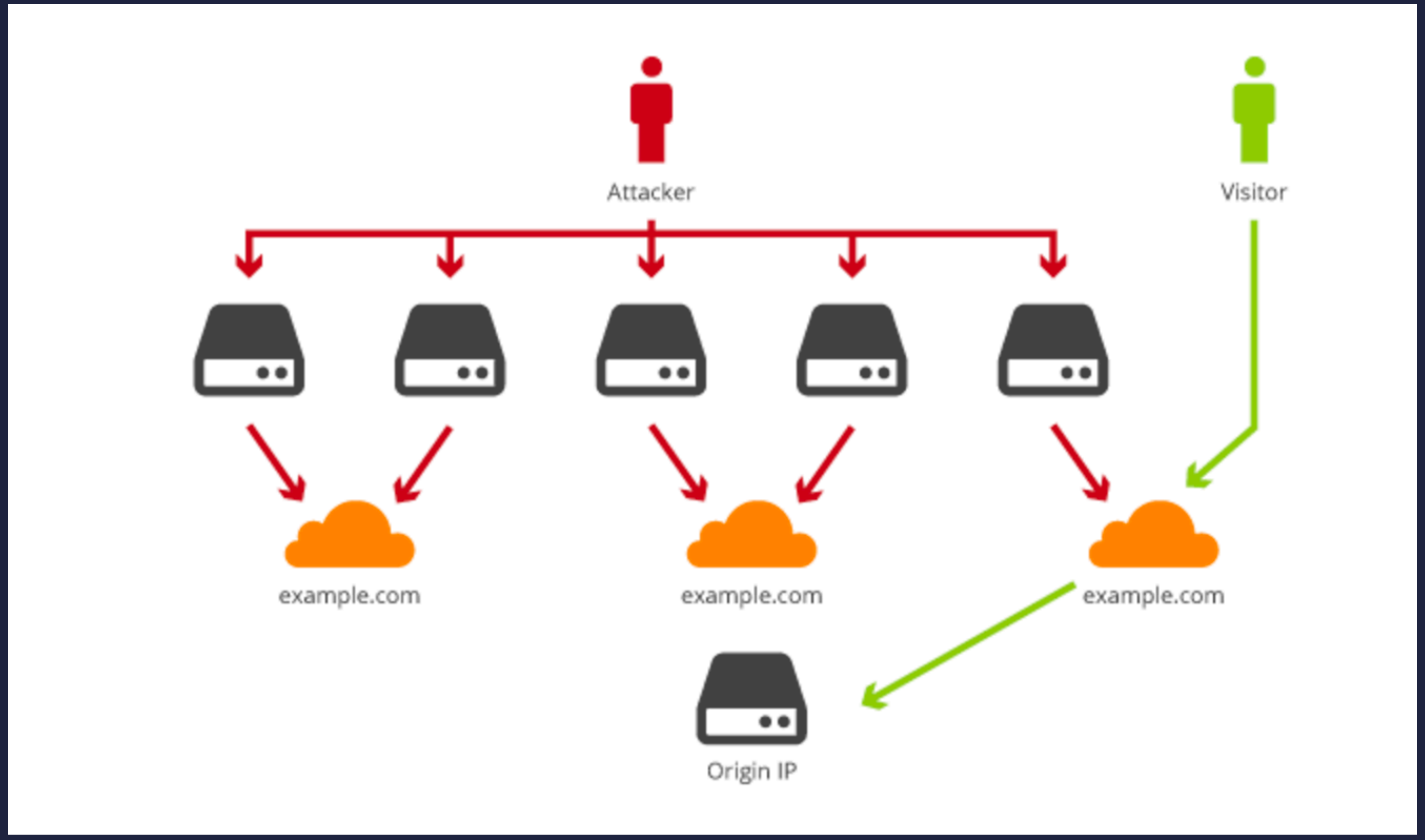
WHAT HAPPENS IN A DDOS ATTACK



Source: https://blog.cloudflare.com/ddos-prevention-protecting-the-origin/

FIGHTING A DDOS ATTACK

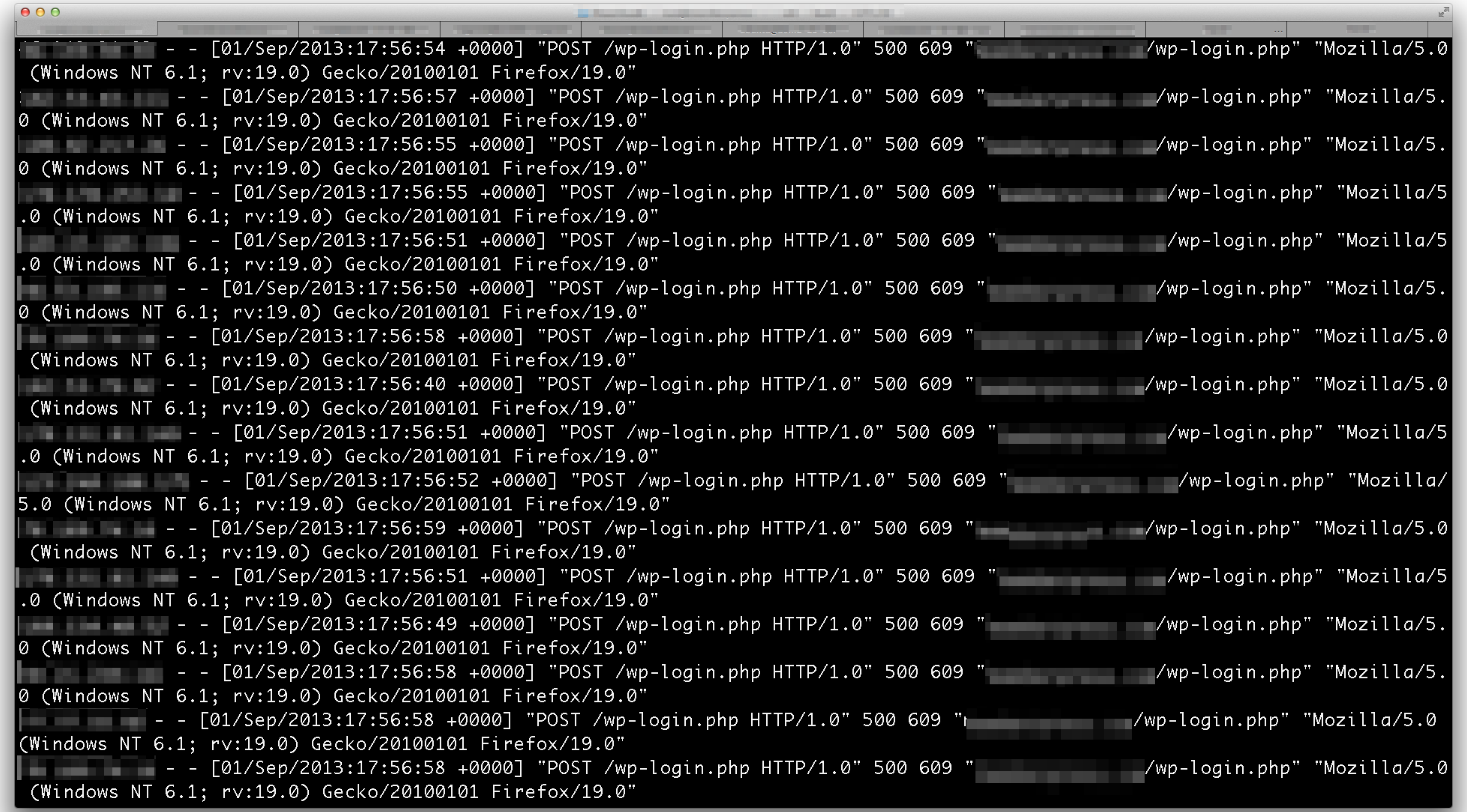
Fighting a DDoS attack is a very complex thing to do but an easy way is to use a service like CloudFlare.



Source: https://blog.cloudflare.com/ddos-prevention-protecting-the-origin/

BRUTE FORCE (*gra-gra*)

Brute force is a trial and error method used by application programs to decode encrypted data such as passwords, through exhaustive effort.



```
[redacted] - - [01/Sep/2013:17:56:54 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.0
 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:57 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.
0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:55 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.
0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:55 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5
.0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:51 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5
.0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:50 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.
0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:58 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.0
 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:40 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.0
 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:51 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5
.0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:52 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/
5.0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:59 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.0
 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:51 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5
.0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:49 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.
0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:58 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.
0 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:58 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.0
(Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
[redacted] - - [01/Sep/2013:17:56:58 +0000] "POST /wp-login.php HTTP/1.0" 500 609 "[redacted]/wp-login.php" "Mozilla/5.0
 (Windows NT 6.1; rv:19.0) Gecko/20100101 Firefox/19.0"
```

TARGET="_blank"

When you use target='_blank' the page you're linking to gains partial access to the linking page via the `window.opener` object.

The newly opened tab can change the `window.opener.location` to some phishing page. Or even execute some JavaScript on the opener-page on your behalf... Users trust the page that is already opened, they won't get suspicious.

```
rel="noopener noreferrer"
```

In JavaScript, `window.open` is also vulnerable to this, so always nullify if you are using it.

```
var newWindow = window.open();
newWindow.opener = null;
```

THANK YOU!

SLIDES AVAILABLE AT: HTTPS://BIT.LY/BUILDING-FOR-THE-BAHD-GUYS