Web Security I: Injection Attacks

Chengyu Song

Slides modified from
Dawn Song and John Mitchell

What is Web?

Web is a platform for deploying applications and sharing information,

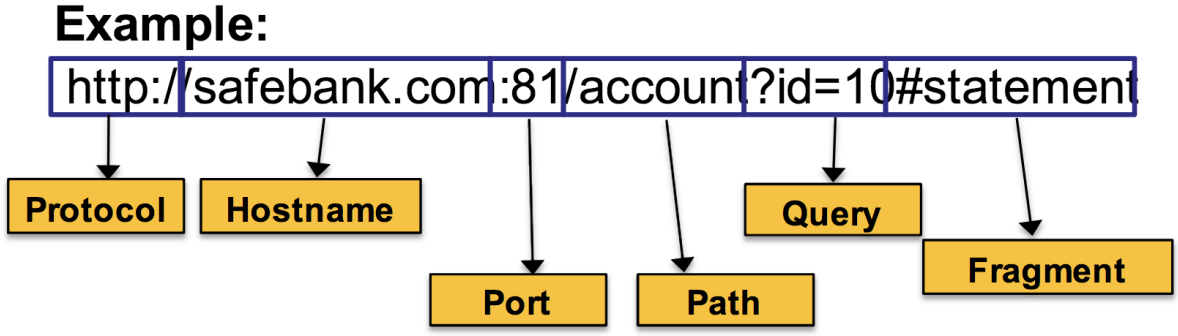
portably and *securely*

Hypertext Transfer Protocol

HTTP: a common data communication protocol on the web.

Uniform Resource Locator

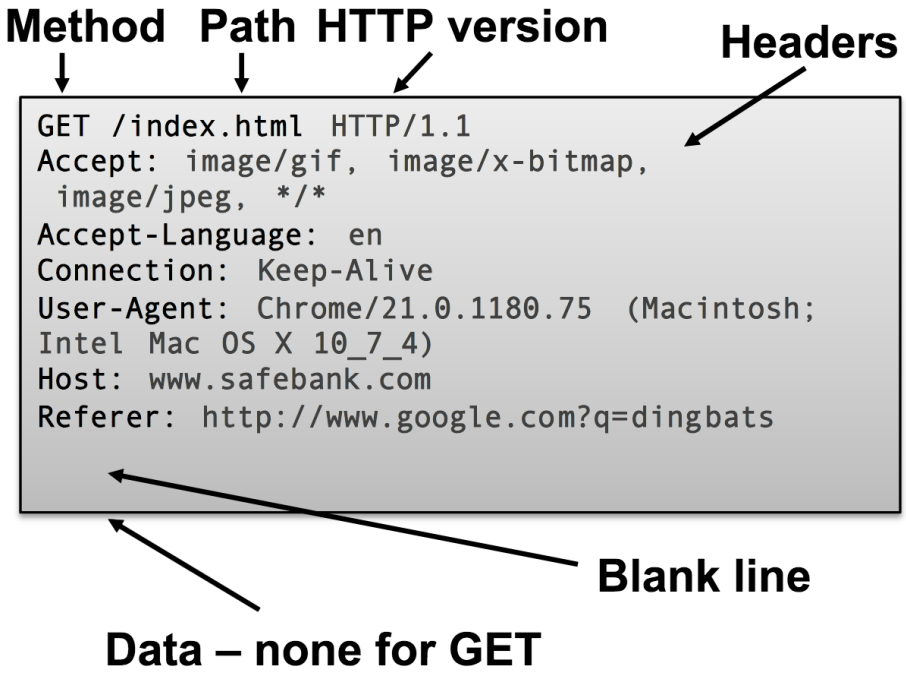
URL: global identifiers of network-retrievable resources



HTTP request

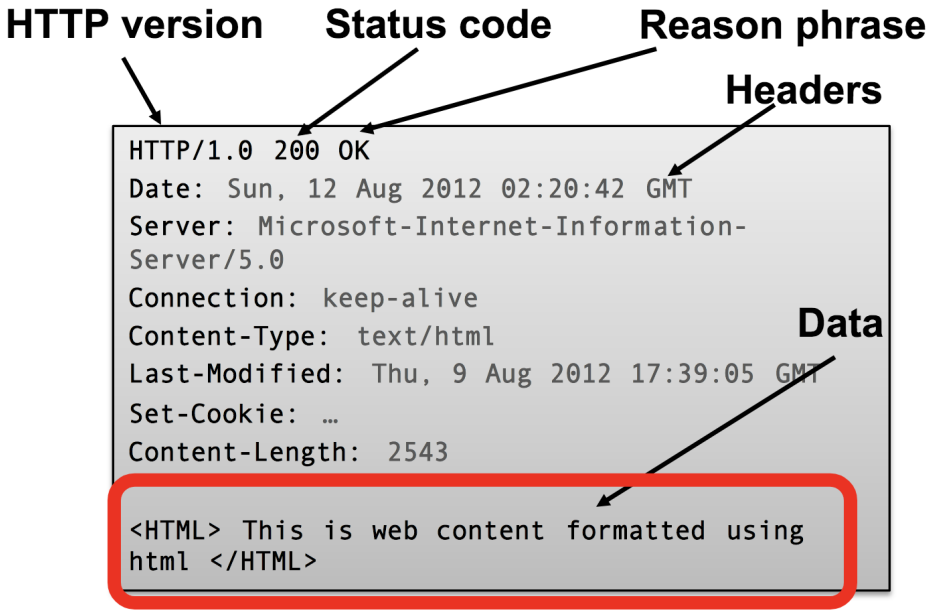
Sending commands to the sever side, like system call.

GET: no side effect
POST: possible side effect



HTTP response

Retrieving results from the server side, like system call returns.



HyperText Markup Language

HTML: a markup language to create structured documents that can embed

images, objects, create interactive forms, etc.

```
<html>
  <body>
    <div>foo <a href="http://google.com">Go to Google!</a></div>
    <form>
      <input type="text" /> <input type="radio" />
      <input type="checkbox" />
    </form>
  </body>
</html>
```

Web security: a historical perspective

- Similar to Internet, web is an example of "bolt-on security"
- Originally, the World Wide Web (www) was invented to allow physicists to share their research papers
  - Only textual web pages + links to other pages
  - No security model to speak of

Web security: nowadays

- The web became complex and adversarial quickly
- Web pages become very complex with embedded images, JavaScript, dynamic HTML, AJAX, CSS, frames, audio, video, sensors, VR, … from different servers
  - Today, a web site is a distributed application
- Web applications also become very diverse, news, shopping, videos, social network, banking, gaming, …
  - Attackers have various motivations

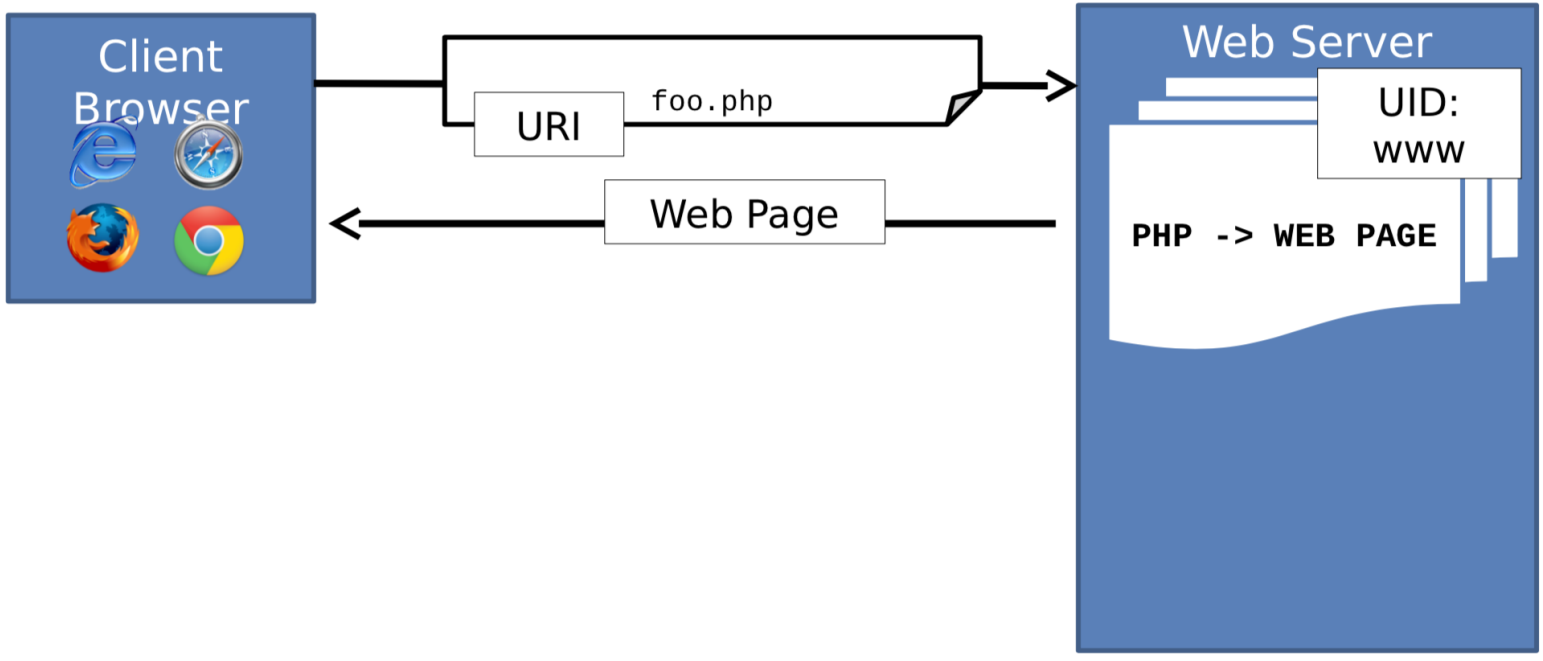
Desirable security goals

- **Integrity**: malicious websites should not be able to tamper with the integrity of my computer or my information on other web sites
- **Confidentiality**: malicious websites should not be able to learn confidential information from my computer or other web sites
- **Privacy**: malicious websites should not be able to spy on me or my activities online

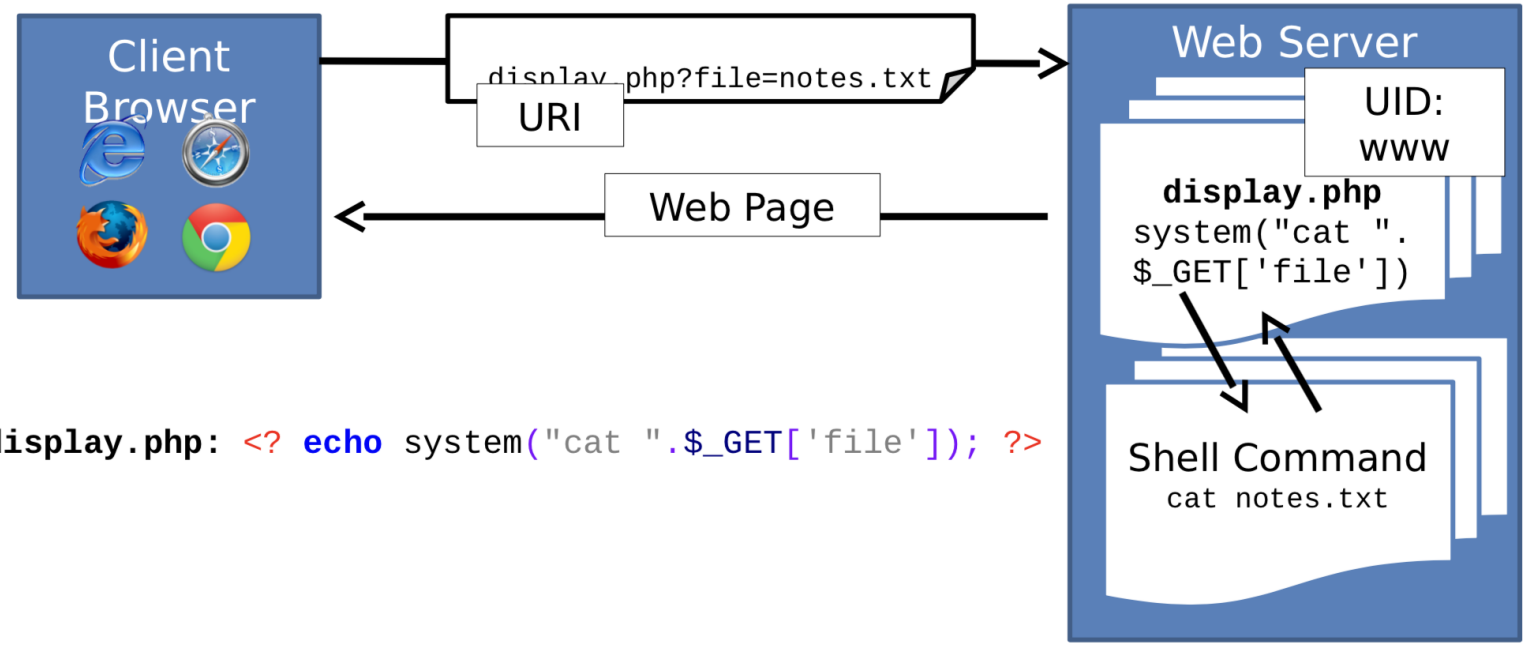
How these properties can be violated?

- Server side: injection attacks
- Client side: cross-site attacks

Background: how to server web requests



Background: a simple display web app




```
display.php: <? echo system("cat ".$_GET['file']); ?>
```

Background: how php works

```
<? echo system("cat ".$_GET['file']); ?>
```

<? *php-code* ?>	executes php-code at this point in the document
echo expr:	evaluates expr and embeds in doc
system(call, args)	performs a system call in the working directory
“ ….. ”, ‘ ….. ’	String literal. Double-quotes has more possible escaped characters.
.	(dot). Concatenates strings.
_GET[‘key’]	returns *value* corresponding to the *key/value* pair sent as extra data in the HTTP GET request

Command injection

```
<? echo system("cat ".$_GET['file']); ?>
```

Q: Assuming the script we've been dealing with (reproduced above) for http://www.example.net/display.php. Which one of the following URIs is an attack URI? (Hint: `%3B` -> ";" `%20` -> " " `%2F` -> "/")

a. http://www.example.net/display.php?file=rm
b. http://www.example.net/display.php?file=rm%20-rf%20%2F%3B
c. http://www.example.net/display.php?file=notes.txt%3B%20rm%20-rf%20%2F%3B%0A%0A
d. http://www.example.net/display.php?file=%20%20%20%20%20

Command injection

```
<? echo system("cat ".$_GET['file']); ?>
```

Q: Assuming the script we've been dealing with (reproduced above) for http://www.example.net/display.php. Which one of the following URIs is an attack URI?

a. http://www.example.net/display.php?file=rm
b. http://www.example.net/display.php?file=rm -rf /;
c. http://www.example.net/display.php?file=notes.txt; rm -rf /;
d. http://www.example.net/display.php?file=

Command injection

```
<? echo system("cat ".$_GET['file']); ?>
```

Q: Assuming the script we've been dealing with (reproduced above) for http://www.example.net/display.php. Which one of the following URIs is an attack URI?

a. `<? echo system("cat rm"); ?>`
b. `<? echo system("cat rm -rf /;"); ?>`
c. `<? echo system("cat notes.txt; rm -rf /;"); ?>`
d. `<? echo system("cat "); ?>`

Injection

- Injection is a general problem
  - Typically caused when data and code share the same *channel*
  - For example the code is `cat` and the filename the data
    - But ';' allows attacker to start a new command
- Q: does shellcode fall into the same category?

Input validation

- Two forms
  - **Blacklisting**: block known attack values
  - **Whitelisting**: only allow known-good values
- Blacklists are easily bypassed
  - Set of 'attack' inputs is potentially in nite
  - The set can change after you deploy your code
  - Only rely on blacklists as a part of a defense in depth strategy

Blacklist bypass

- Disallow semi-colons -> Use a pipe
- Disallow pipes and semi-colons -> Use the backtick operator to call commands in the arguments
- Disallow pipes, sem-coilons, and backtick -> Use the $ operator which works similar to backtick
- Disallow `rm` -> Use `unlink`
- Disallow `rm`, `unlink` -> Use cat to overwrite existing files

Whitelisting

```
<?
if(!preg_match("/^[a-z0-9A-Z.]*$/", $_GET['file'])) {
  echo “The file should be alphanumeric.";
  return;
}
echo system("cat ".$_GET['file']); ?>
```

- Can these input pass?

```
notes.txt
notes.txt; rm -rf /;
security notes.txt
```

Whitelisting

```
<?
if(!preg_match("/^[a-z0-9A-Z.]*$/", $_GET['file'])) {
  echo “The file should be alphanumeric.";
  return;
}
echo system("cat ".$_GET['file']); ?>
```

- Can these input pass?

```
notes.txt (YES)
notes.txt; rm -rf /; (NO)
security notes.txt (NO)
```

Input escaping

```
<?
#http://www.php.net/manual/en/function.escapeshellarg.php
echo system("cat ".escapeshellarg($_GET['file']));
?>
```

> ***escapeshellarg()*** *adds single quotes around a string and quotes/escapes any existing single quotes allowing you to pass a string directly to a shell function and having it be treated as a single safe argument --*
>
> *http://www.php.net/manual/en/function.escapeshellarg.php*

Input escaping

```php
<?
#http://www.php.net/manual/en/function.escapeshellarg.php
echo system("cat ".escapeshellarg($_GET['file']));
?>
```

```
+----------------------+----------------------------+
| GET input            | Command executed           |
+----------------------+----------------------------+
| notes.txt            | cat 'notes.txt'            |
| notes.txt; rm -rf /; | cat 'notes.txt rm -rf /;' |
| mary o'donnel        | cat 'mary o'\''donnel'     |
+----------------------+----------------------------+
```

Use less powerful API

- The `system` command is too powerful
  - Executes the string argument in a new shell
  - If only need to read a le and output it, use simpler API

```php
<? echo file_get_contents($_GET['file']); ?>
```

- Similarly, the `proc_open()` API
  - Executes commands and opens files for I/O
  - Can only execute one command at a time

Recap

- Command Injection: a case of injection, a general vulnerability
- Defenses against injection include input validation, input escaping, and use of a less powerful API

SQL injection: background

- SQL: A query language for database, e.g.,

```
SELECT statement WHERE clauses
```

- More information: http://en.wikipedia.org/wiki/SQL

Running example

Consider a web page that logs in a user by seeing if a user exists with the given username and password.

```
$result = pg_query("SELECT * from users WHERE
                    uid = '".$_GET['user']."' AND
                    pwd = '".$_GET['pwd']."';");
if (pg_query_num($result) > 0) {
  echo "Success";
  user_control_panel_redirect();
}
```

It sees if results exist and if so logs the user in and redirects them to their user control panel.

SQL injection

Q: Which one of the following queries will log you in as admin? (Hint: '--' starts comments)

a. `http://www.example.net/login.php?user=admin&pwd='`
b. `http://www.example.net/login.php?user=admin--&pwd=foo`
c. `http://www.example.net/login.php?user=admin'--&pwd=f`
d. It is not possible. (None of the above)

SQL injection

```
$result = pg_query("SELECT * from users WHERE
                uid = '".$_GET['user']."' AND
                pwd = '".$_GET['pwd']."';");
if (pg_query_num($result) > 0) {
  echo "Success";
  user_control_panel_redirect();
}
```

URI: `http://www.example.net/login.php?user=admin'--&pwd=f`

```
pg_query("SELECT * from users WHERE
          uid = 'admin'--'    AND  pwd = 'f';");
pg_query("SELECT * from users WHERE
          uid = 'admin';");
```

SQL injection

Q: Under the same premise as before, which URI can delete the users table in the database?

a. `www.example.net/login.php?user=;DROP TABLE users;--`
b. `www.example.net/login.php?user=admin%27%3B%20DROP%20TABLE%20users--%3B&pwd=f`
c. `www.example.net/login.php?user=admin;%20DROP%20TABLE%20users;%20--&pwd=f`
d. It is not possible. (None of the above)

SQL injection

- URI: `www.example.net/login.php?`

  `user=admin%27%3B%20DROP%20TABLE%20users--%3B&pwd=f`

- Decoded: `www.example.net/login.php?user=admin'; DROP TABLE users;--&pwd=f`

```
pg_query("SELECT * from users WHERE
          uid = 'admin'; DROP TABLE users;--' AND
          pwd = 'f';");
pg_query("SELECT * from users WHERE uid = 'admin';
          DROP TABLE users;");
```

SQL injection

- One of the most exploited vulnerabilities on the web
- Cause of massive data theft
  - 24% of all data stolen in 2010
  - 89% of all data stolen in 2009
- Like command injection, caused when attacker controlled data interpreted as a (SQL) command

Injection defenses

- Input validation
  - Whitelists untrusted inputs to a safe list
- Input escaping
  - Escape untrusted input so it will not be treated as a command
- Use less powerful API
  - Use an API that only does what you want
  - Prefer this over all other options

Input validation for SQL

```
<?
if(!preg_match("/^[a-z0-9A-Z.]*$/", $_GET[‘user'])) {
  echo "Username should be alphanumeric.";
  return;
}
// Continue to do login query
?>
```

- Can these input pass?

```
Pikachu
Pikachu'; DROP TABLE users--
O'Donnel
```

Input validation for SQL

```
<?
if(!preg_match("/^[a-z0-9A-Z.]*$/", $_GET[‘user'])) {
  echo "Username should be alphanumeric.";
  return;
}
// Continue to do login query
?>
```

- Can these input pass?

```
Pikachu (YES)
Pikachu'; DROP TABLE users-- (NO)
O'Donnel (NO, FALSE POSITIVE)
```

Input validation for SQL

```
pg_query("SELECT * from users WHERE
          uid = '".$_GET['user']."' AND
          pwd = '".$_GET['pwd']."';");
```

Q: Which of the following URIs would still allow you to login as admin?

a. http://www.example.net/login.php?user=admin&pwd=admin
b. http://www.example.net/login.php?user=admin&pwd='%20OR%201%3D1;--
c. http://www.example.net/login.php?user=admin'--&pwd=f
d. http://www.example.net/login.php?user=admin&pwd='--

Input validation for SQL

```
pg_query("SELECT * from users WHERE
          uid = '".$_GET['user']."' AND
          pwd = '".$_GET['pwd']."';");
```

URI (decoded): `http://www.example.net/login.php?user=admin&pwd=' OR 1=1;--`

```
pg_query("SELECT * from users WHERE uid = 'admin' AND
          pwd = '' OR 1 = 1;--';");
```

Input escaping

```
$_GET['user'] = pg_escape_string($_GET['user']);
$_GET['pwd'] = pg_escape_string($_GET['pwd']);
```

> ***pg_escape_string()*** *escapes a string for querying the PostgreSQL database. It returns an escaped literal in the PostgreSQL format.*

```
+-----------------------------+-------------------------------+
| GET input                   | Escaped output                |
+-----------------------------+-------------------------------+
| Bob                         | Bob                           |
| Bob'; DROP TABLE users; --  | Bob''; DROP TABLE users; --   |
| Bob' OR '1'='1              | Bob'' OR ''1''=''1            |
+-----------------------------+-------------------------------+
```

Use less powerful API

- Create a template for SQL Query, in which data values are substituted
- The database ensures untrusted value isn't interpreted as command
- **Always prefer over all other techniques**
- Less powerful:
    - Only allows queries set in templates.

Use less powerful API

```
<?
# The $1 and $2 are a ‘hole’ or place holder for what will
# be filled by the data
$result = pg_query_params('SELECT * FROM users WHERE
                           uid = $1 AND pwd = $2',
                           array($_GET['user'], $_GET['pwd']));

# Compare to
$result = pg_query("SELECT * FROM users WHERE
                    uid ='".$_GET['user']."' AND
                    pwd ='".$_GET[‘pwd'].”';");
?>
```

Recap

- SQL Injection: a case of injection, in database queries
- Extremely common, and pervasively exploited
- Use prepared statements to prevent SQL injection
    - **DO NOT** use escaping, despite what xkcd says