

how do i shot web

ISSS beginner talk

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name (
    { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
      | table_constraint
      | LIKE parent_table [ like_option ... ] }
    [, ... ]
)
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
```

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
    OF type_name [ (
        { column_name WITH OPTIONS [ column_constraint [ ... ] ]
          | table_constraint }
        [, ... ]
    ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
```

where *column_constraint* is:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

and *table_constraint* is:

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  EXCLUDE [ USING index_method ] ( exclude_element WITH operator [, ... ] ) index_parameters [ WHERE ( predicate ) ]
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

and *like_option* is:

```
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | ALL }
```

index_parameters in UNIQUE, PRIMARY KEY, and EXCLUDE constraints are:

```
[ WITH ( storage_parameter [= value] [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace ]
```

exclude_element in an EXCLUDE constraint is:

```
{ column | ( expression ) } { opclass } [ ASC | DESC ] [ NULLS { FIRST | LAST } ]
```

HTTP Status Codes

This page is created from HTTP status code information found at ietf.org and Wikipedia. Click on the **category** heading or the **status code** link to read more.

1xx Informational

100 Continue

2xx Success

★ 200 OK

203 Non-Authoritative Information

206 Partial Content

226 IM Used

3xx Redirection

300 Multiple Choices

303 See Other

306 (Unused)

4xx Client Error

★ 400 Bad Request

★ 403 Forbidden

406 Not Acceptable

★ 409 Conflict

412 Precondition Failed

415 Unsupported Media Type

418 I'm a teapot (RFC 2324)

423 Locked (WebDAV)

426 Upgrade Required

431 Request Header Fields Too Large

450 Blocked by Windows Parental Controls (Microsoft)

101 Switching Protocols

★ 201 Created

★ 204 No Content

207 Multi-Status (WebDAV)

301 Moved Permanently

★ 304 Not Modified

307 Temporary Redirect

102 Processing (WebDAV)

202 Accepted

205 Reset Content

208 Already Reported (WebDAV)

302 Found

305 Use Proxy

308 Permanent Redirect (experimental)

402 Payment Required

405 Method Not Allowed

408 Request Timeout

411 Length Required

414 Request-URI Too Long

417 Expectation Failed

422 Unprocessable Entity (WebDAV)

425 Reserved for WebDAV

429 Too Many Requests

449 Retry With (Microsoft)

499 Client Closed Request (Nginx)

★ 401 Unauthorized

★ 404 Not Found

407 Proxy Authentication Required

410 Gone

413 Request Entity Too Large

416 Requested Range Not Satisfiable

420 Enhance Your Calm (Twitter)

424 Failed Dependency (WebDAV)

428 Precondition Required

444 No Response (Nginx)

451 Unavailable For Legal Reasons

5xx Server Error

★ 500 Internal Server Error

503 Service Unavailable

506 Variant Also Negotiates (Experimental)

509 Bandwidth Limit Exceeded (Apache)

598 Network read timeout error

501 Not Implemented

504 Gateway Timeout

507 Insufficient Storage (WebDAV)

510 Not Extended

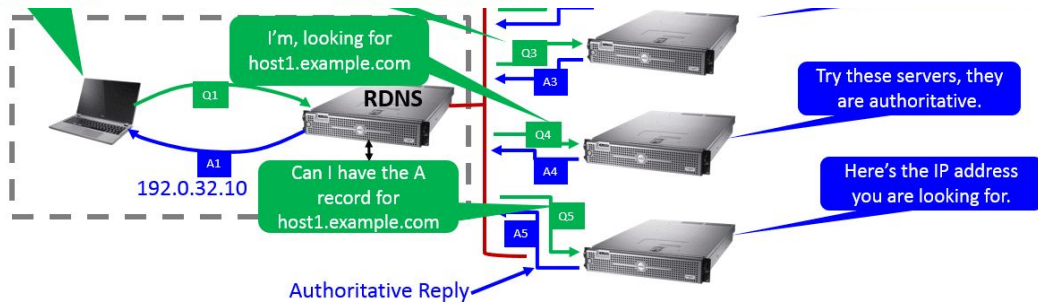
599 Network connect timeout error

502 Bad Gateway

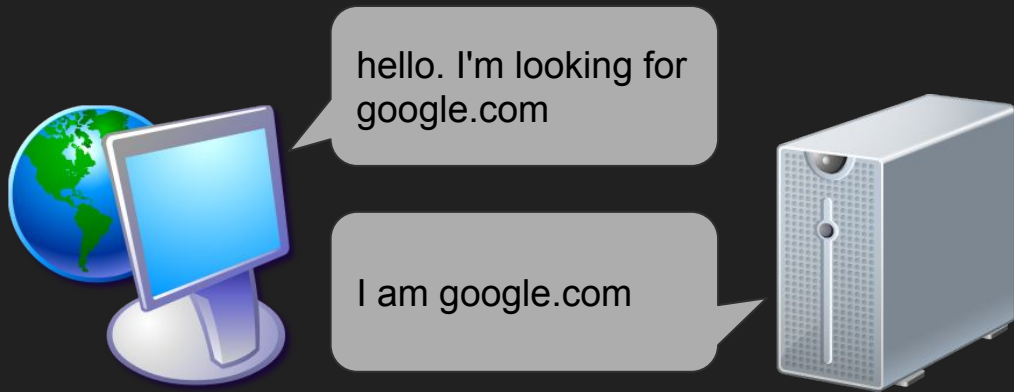
505 HTTP Version Not Supported

508 Loop Detected (WebDAV)

511 Network Authentication Required



for CTFs



- we can just think of the web as a way to talk to some other computer
- we use the web to send messages back and forth between computers
 - these are called *packets*

simplified server-client model



example



client

user navigates to example.com

```
GET /index.html HTTP/1.1  
Host: example.com
```



example.com server

```
HTTP/1.1 200 Success  
Server: example.com  
Content-Type: text/html  
<!DOCTYPE HTML>  
...  
<script src="/scripts/index.js">  
...
```

```
GET /scripts/index.js HTTP/1.1  
Host: example.com
```

example



client

HTTP/1.1 200 Success

```
...  
function submitForm() {  
    fetch("/login", {method: "post", body: {username: ..., password: ...}}).then(() => ...);  
}  
...
```

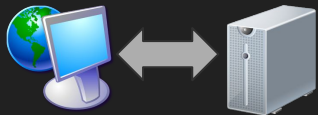


example.com server

...
user submits form/clicks button/presses enter, submitForm() runs in the browser
a login request is created by the JavaScript code

```
POST /login HTTP/1.1  
Host: example.com  
username=ggu&password=hunter2
```

example



```
SELECT * FROM User WHERE username = 'ggu'  
AND password = '769994529e9e073c3c661ccf25350b70';
```

username	email	password
ggu	gu@utexas.edu	75999...



underlying database

the server now knows the user exists and has the correct password
now it will log the user in...

example



client

HTTP/1.1 200 Success
...
Set-Cookie: token=eyJhbGciOiJIUzI1NiIsInR5cCI6I...

OK



example.com server

The browser now stores the cookie on disk. Every access to the server will now have the "token" cookie attached automatically.

GET /friends HTTP/1.1
Host: example.com
Cookie: token=eyJhbGciOiJIUzI1NiIsInR5cCI6I...

The server now knows who the client is because it can check the cookie.



Attacking cookies

- The server gives us a cookie to remind it who we are logged in as
- Since the cookie belongs to us, we can change it to whatever we want
 - Usually Inspect Element -> Application -> Storage -> Cookies
- A good server would "sign" the cookie to make sure it's not fake
 - JWT tokens are popular
 - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
 - "None" algorithm
- Log in as "thelegend27" here: <http://gggu.issc.io/cookies>

Bypassing client-side sanitization

- Sometimes the site sanitizes/checks a value entirely on the client-side
- e.g. "The username must not contain HTML characters", "Must not be empty"
- Tricking the server into accepting an invalid value can break things
- A few ways to do this:
 - Mess with the JavaScript in the page (rewrite a function)
 - Use Firefox "Edit and Resend" feature (don't forget to change "Content-Length" header)
 - Network -> right click on request -> "copy as cURL" -> copy into terminal
- Login as admin here: http://ggu.issc.io/client_side

Stored XSS

- Suppose a user posts a message, and the server simply places the message into the website
- **user123** says: **<i>Hello</i><script>fetch("/delete-account");</script> I am not a hacker**
- What can happen?
 - Cookies get leaked
 - XSS worm
 - Accounts get hijacked
- Gain admin status here: <http://ggu.iss.io/tickets>
 - Look up how to make a GET request in JavaScript (fetch is easiest)

Common Beginner CTF Problems

- robots.txt file
- .git directory
- jwt.io
- SQL injection
- OWASP top 10

Questions