CSCI E-33a

CS50's Web Programming
with Python and JavaScript

Spring 2020

Barbara Karakyriakou · Teaching Fellow

- Email: karakyriakou@fas.harvard.edu
- Phone: 617 820 6930

- Section Meetings: Wednesdays 8:30pm-10:00pm EST
- Office Hours: Saturdays 1:00pm - 2:30pm EST

Section 7: Scalability & Security

Agenda

- Scalability
  - Definition
  - Vertical vs Horizontal Scalability
  - Importance of Scalability
  - Django & Scalability
- Security
  - Vulnerability
  - Threats
  - Encryption
  - Hashing
  - Django & Security
- Useful Links
- Q&A

Scalability

Scalability is the ability of a system to efficiently handle a growing client demand by adapting and adding resources.

Vertical vs Horizontal Scaling

- Vertical scaling is adding more power to an existing machine
- Horizontal scaling is increasing resources by adding more machines

Vertical Scaling Example

Suppose that you have a database server with 10GB memory and it has exhausted. To handle more data, you can buy a server with memory of 2TB, which can handle larger amounts of data.

This is called Vertical Scaling.

The process involves adding more power such as CPU and disk power to enhance your storage process.

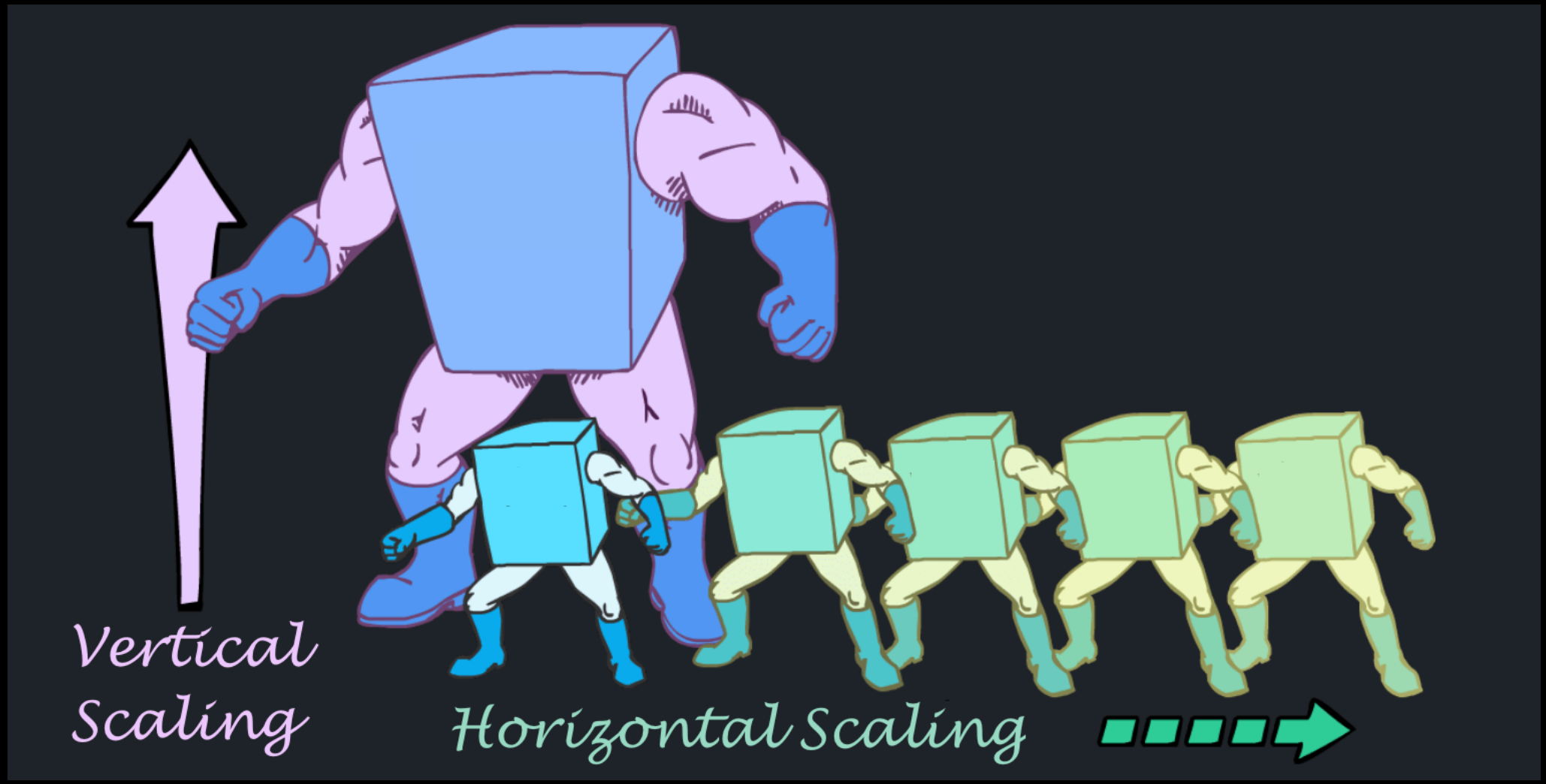
Horizontal Scaling Example

Instead of buying a single 2TB server, you are buying two hundred 10 GB servers.

This is called Horizontal Scaling.

It divides the data set and distributes the data over multiple servers, or *shards.* Each shard is an independent database.

Vertical vs Horizontal Scaling
(Scale Up or Scale Out)



Horizontal vs Vertical Scaling: Which is Better?

While both have advantages and disadvantages, horizontal scaling is more elastic, dynamic, and easier, because you can add more machines into an existing pool.

Vertical scaling on the contrary is often limited to the capacity of a single machine. Scaling beyond that capacity results in downtime and comes with an upper limit.

Horizontal vs Vertical Scaling: Which is Better?

Take for example, Facebook and Instagram. When they were was invented, they were addressed to a small crowd. Therefore, a single server would have been enough. However now, millions of people use these applications. It is impossible to have a single server which can store all the data.

Importance of Scalability

Why scalability should be a concern when building app?

You may start your application in a small scale, but with time it can grow larger, so you want to plan ahead and avoid having to rewrite your application.

Django & Scalability

According to their website, Django is “exceedingly scalable. Some of the busiest sites on the planet use Django’s ability to quickly and flexibly scale to meet the heaviest traffic demands.”

Some examples of websites that are running on Django are Instagram, Bitbucket, Quora, Pinterest, YouTube, Spotify.

https://www.djangoproject.com/start/overview/

Django & Scalability

How does Django scale? According to their website :

“Compared to development time, hardware is cheap, and so Django is designed to take advantage of as much hardware as you can throw at it.

Django uses a “shared-nothing” architecture, which means you can add hardware at any level – database servers, caching servers or Web/application servers.

The framework cleanly separates components such as its database layer and application layer. And it ships with a simple-yet-powerful cache framework.”

https://docs.djangoproject.com/en/3.0/faq/general/#does-django-scale

Security

Security vulnerability is a flaw or a weakness in software code or in a system which can be exploited by a an attacker (hacker) using a tool or a technique (threat) in order to perform unauthorized actions.

Common Security Vulnerabilities

- SQL Injection: when an attacker attempts to access or corrupt a database content to create, read, update, alter, or delete stored data.
- Broken Authentication: when an attacker hijacks active sessions to compromise authentication credentials and to assume the identity of a user.
- Cross-Site Scripting (XSS): when an attacker injects malicious code targeting website users, rather than the actual website itself, to redirect them(users) malicious sites.

Common Security Vulnerabilities

- Cross-Site Request Forgery (CSRF): when an attacker tricks an authenticated user into performing an action that she/he does not intend to do. A third-party website will send a request to a web application in which a user is already authenticated. The attacker can then access functionality via the victim's already authenticated browser.
- Security Misconfiguration: when an attacker access private data or features due to lack of maintenance or a lack of attention to a web application configuration.
- Insecure direct object reference: when a web application exposes a reference to an internal implementation object, such as files, database records, and database keys. An attacker can manipulate a reference to one of these objects to gain access to a user's personal data.

Common Security Threats

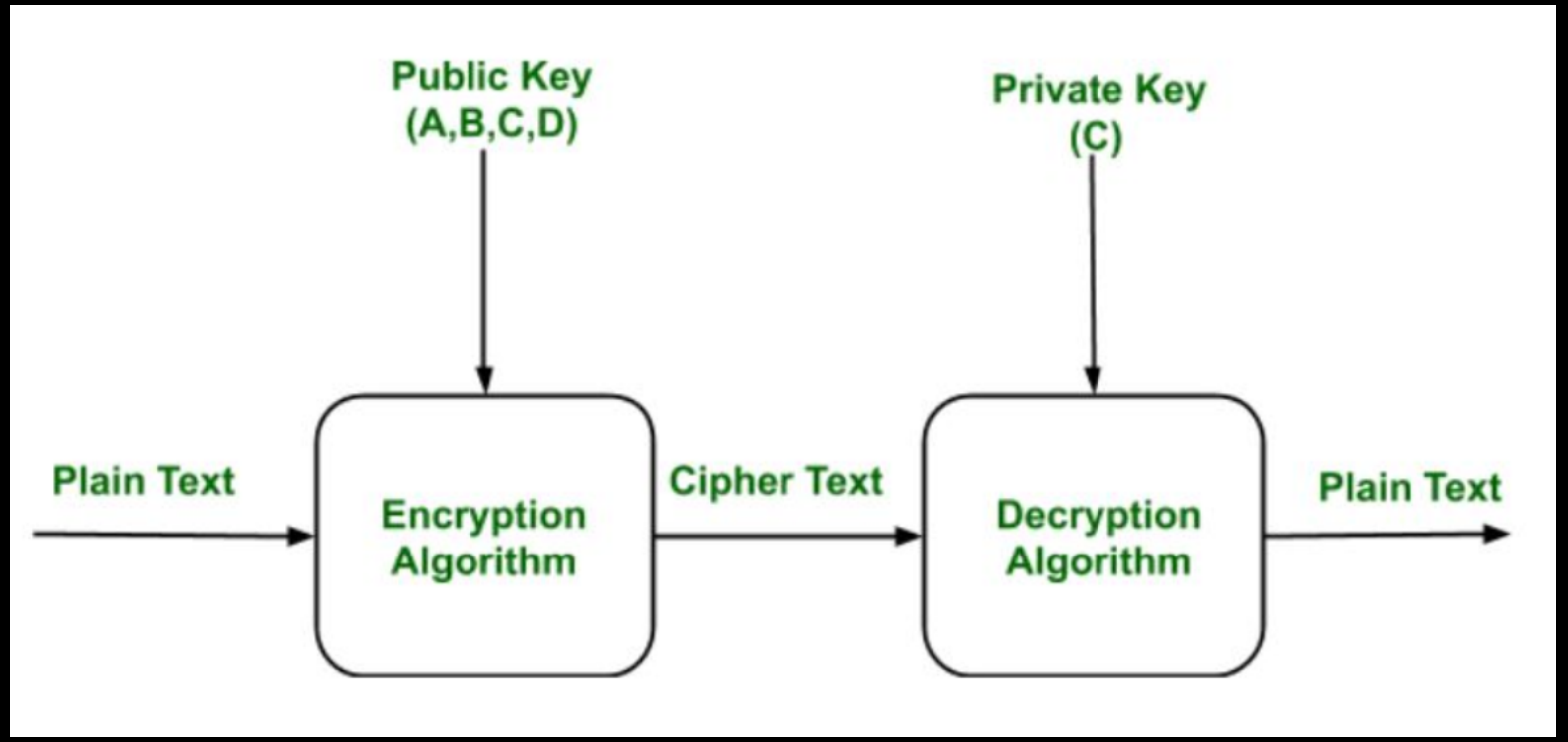
- Virus: replicates itself by modifying computer programs and inserting its own code
- Eavesdropping: a network layer attack
- Malware: a software purposely designed to cause damage
- Spyware: a type of malware aiming of stealing information
- Ransomware: a malware that threatens to publish victims' data or block their access for ransom
- Trojan: a type of malware that misleads users of its true intend

Common Security Threats

- Rootkit: a malicious set or collection of computer software
- Bootkit: a kernel mode rootkit
- Keylogger: a malicious action of recording the keys struck on a keyboard
- Phishing: a fraudulent attempt to obtain sensitive information
- Worm: a stand alone malicious program that replicates itself and spreads through networks
- Backdoor: a hidden method of bypassing normal authentication or encryption

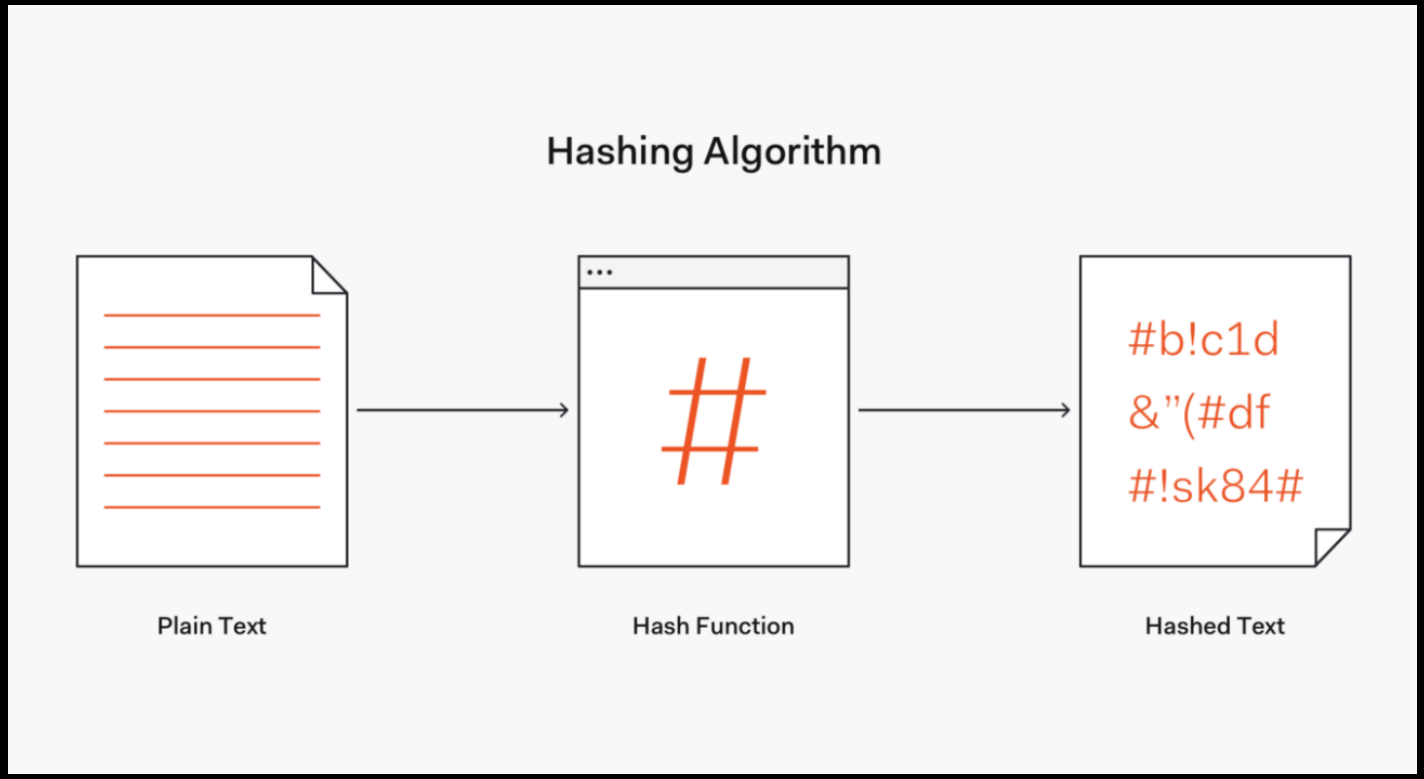
Security: Encryption

- Secret-Key Cryptography: when a secret or private key is the piece of information or parameter that is used to encrypt and decrypt messages in a symmetric encryption
- Public-Key Cryptography: an asymmetric encryption in which two separate keys are used. One is a public key (encrypts) and the other is a secret key (de-crypts)



Security: Hashing

Hashing is the creation of a unique hash value by using an algorithm to map data of any size to a fixed length.
The main difference between hashing and encryption is that hashing is one-way, in other words hashing is not reversible and cannot be de-crypted.



Django & Security

Django provides tools that help preventing common mistakes which can cause security vulnerabilities, such as CSRF, XSS, etc. However, security depends on the proper use of a tool and not the tool itself. Therefore, a developer should be able to learn how to use appropriate tools in order to secure her/his program.

For instance, in our projects this semester we used a CSRF token:

In settings.py:

```
MIDDLEWARE_CLASSES = [
'django.middleware.csrf.CsrfViewMiddleware’,
]
```

…and in our template:

```
<form method="post"> {% csrf_token %} ..... </form>
```

Useful Links

- Scalability

  https://powerfulpython.com/blog/scaling-python/

  https://blog.logrocket.com/12-tips-for-writing-clean-and-scalable-javascript-3ffe30abfe20/

  https://docs.djangoproject.com/en/3.0/topics/cache/#

  https://docs.djangoproject.com/en/3.0/faq/general/#does-django-scale

- Security

  https://docs.djangoproject.com/en/3.0/topics/security/

  https://docs.python-guide.org/scenarios/crypto/

  https://www.thepythoncode.com/article/encrypt-decrypt-files-symmetric-python

  https://blog.usejournal.com/javascript-web-application-security-guide-fdade350e373

Q&A