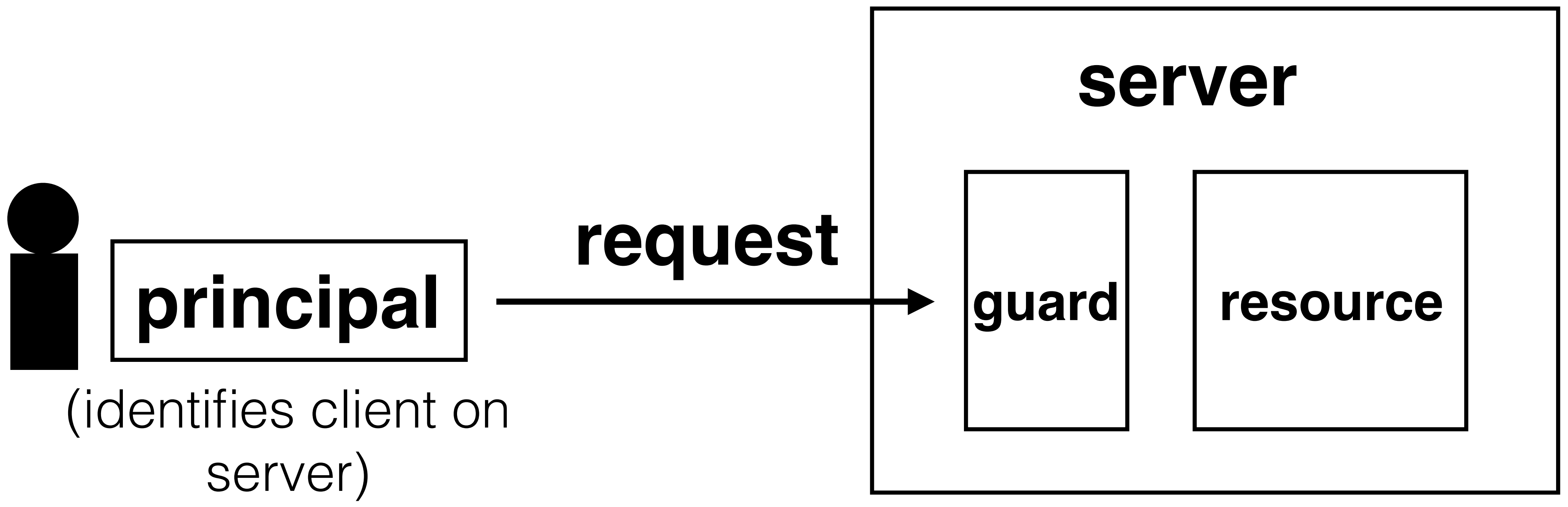
6.033 Spring 2015

Lecture #22

- **Principal Authentication via Passwords**

complete mediation: every request for resource goes through the guard



guard typically provides:

authentication: is the principal who they claim to be?

authorization: does principal have access to perform request on resource?

username	password
arya	valarMorghul1s
jon	w1nterIsC0ming
sansa	LemonCakesForever
hodor	hodor

```
check_password(username, inputted_password):
    stored_password = accounts_table[username]
    if len(stored_password) != len(inputted_password)
      return false
    for i in range(len(stored_password)):
      if stored_password[i] != inputted_password[i]:
        return false
    return true
```

problem: adversary with access to server can get passwords

username	hash(password)
arya	de5aba604c340e1965bb27d7a4c4ba03f4798ac7
jon	321196d4a6ff137202191489895e58c29475ccab
sansa	6ea7c2b3e08a3d19fee5766cf9fc51680b267e9f
hodor	c6447b82fbb4b8e7dbcf2d28a4d7372f5dc32687

```
check_password(username, inputted_password):
  stored_hash = accounts_table[username]
  inputted_hash = SHA1_hash(inputted_password)
  for i in range(len(stored_hash)):
    if stored_hash[i] != inputted_hash[i]:
      return false
  return true
```

top 10 passwords from a leak of 32 million passwords in 2009

source: Imperva, “Consumer Passwords Worst Practices”

password	number of users
123456	290,731
12345	79,078
123456789	76,790
Password	61,958
iloveyou	51,622
princess	35,231
rockyou	22,588
1234567	21,726
12345678	20,553
abc123	17,542

password usage has not improved in recent years. see, e.g.,
https://www.yahoo.com/tech/here-are-500-passwords-you-probably-shouldnt-be-using-96467697789.html
http://adamcaudill.com/2012/07/12/yahoos-associated-content-hacked/
http://www.huffingtonpost.com/2012/06/08/linkedin-password-leak-infographic_n_1581620.html
http://blogs.wsj.com/digits/2010/12/13/the-top-50-gawker-media-passwords/

username	hash(password)
arya	de5aba604c340e1965bb27d7a4c4ba03f4798ac7
jon	321196d4a6ff137202191489895e58c29475ccab
sansa	6ea7c2b3e08a3d19fee5766cf9fc51680b267e9f
hodor	c6447b82fbb4b8e7dbcf2d28a4d7372f5dc32687

```
check_password(username, inputted_password):
  stored_hash = accounts_table[username]
  inputted_hash = SHA1_hash(inputted_password)
  for i in range(len(stored_hash)):
    if stored_hash[i] != inputted_hash[i]:
      return false
  return true
```

problem: adversary can create rainbow table

```
username | salt       | hash(password + salt)
arya     | 5334900209 | c5d2a9ffd6052a27e6183d60321c44c58c3c26cc
jon      | 1128628774 | 624f0ffa577011e5704bdf0760435c6ca69336db
sansa    | 8188708254 | 5ee2b8effce270183ef0f4c7d458b1ed95c0cce5
hodor    | 6209415273 | f7e17e61376f16ca23560915b578d923d86e0319
```

```
check_password(username, inputted_password)
  stored_hash = accounts_table[username]
  inputted_hash = SHA1_hash(inputted_password + salt)
  for i in range(len(stored_hash)):
    if stored_hash[i] != inputted_hash[i]:
      return false
  return true
```

how can we avoid transmitting the password over and over?

session cookies

how can we protect against phishing attacks, where an adversary tricks a user into revealing their password?

challenge-response protocols

turn phishing into an online attack (sitekey)

one-time passwords

bind authentication and authorization

…

how do we initially set (bootstrap) or reset a password?

- Using passwords securely takes some effort. Storing **salted hashes**, incorporating **session cookies**, dealing with **phishing**, and **bootstrapping** are all concerns.
- Thinking about how to use passwords provides more **general lessons**: consider human factors when designing secure systems, be explicit, small improvements are worthwhile, etc.
- There are always **trade-offs**. For instance, all of the methods discussed today add security, but also complexity.