Cyber Offense and Defense

UNIVERSITÀ DELLA CALABRIA
il Campus per eccellenza

Authentication + Business logic vulnerabilities

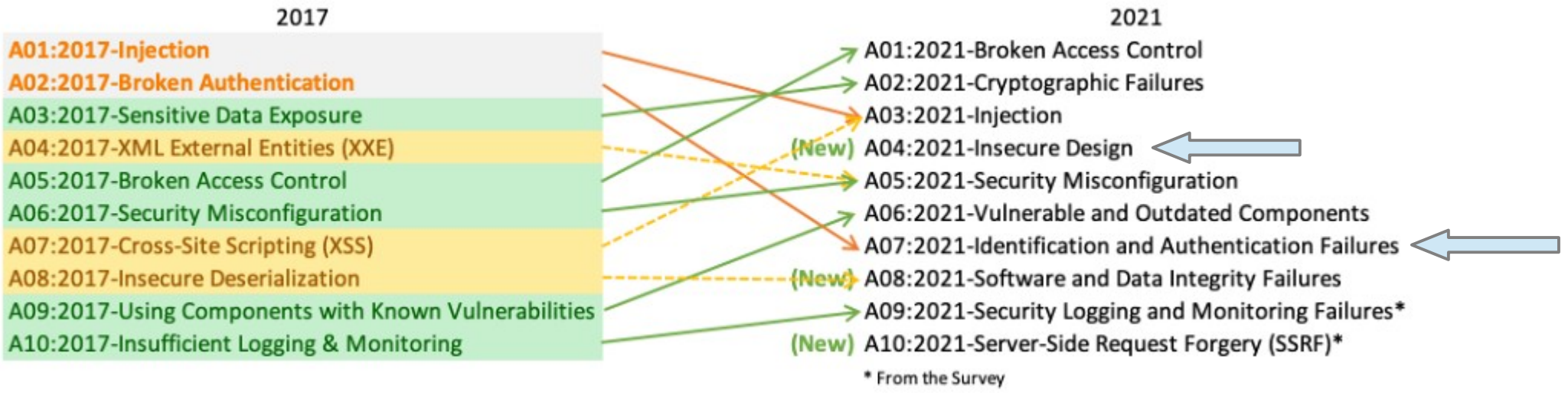
Mario Alviano

Main References

Computer Security: Principles and Practice – Chapter 3
Bug Bounty Bootcamp – Chapter 17
https://portswigger.net/web-security/authentication
https://portswigger.net/web-security/logic-flaws

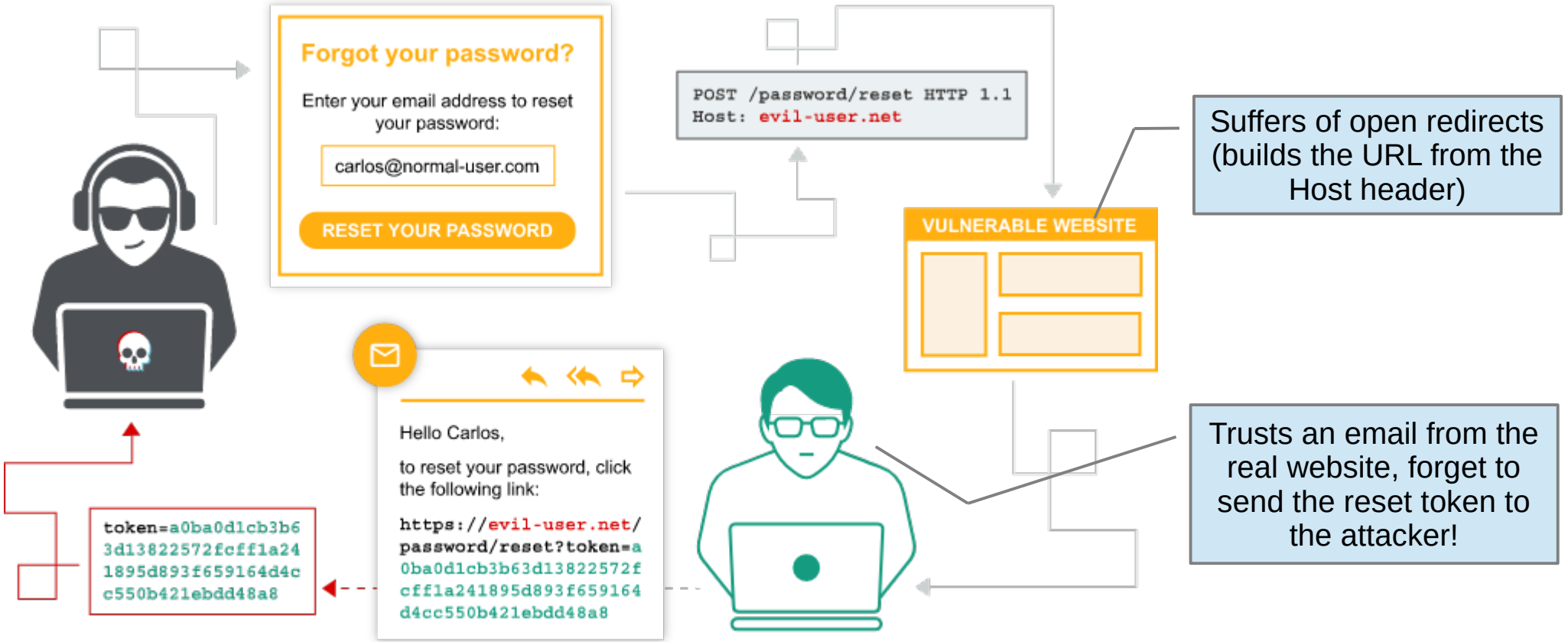
OWASP Top Ten

A broad consensus about the most critical security risks to web applications



User Authentication

The fundamental building block and the primary line of defense



Suffers of open redirects (builds the URL from the Host header)

Trusts an email from the real website, forget to send the reset token to the attacker!

Conceptually simple, authentication vulnerabilities can be among the most critical

AuthN vs AuthZ

Authentication (AuthN) verifies who the user claim to be

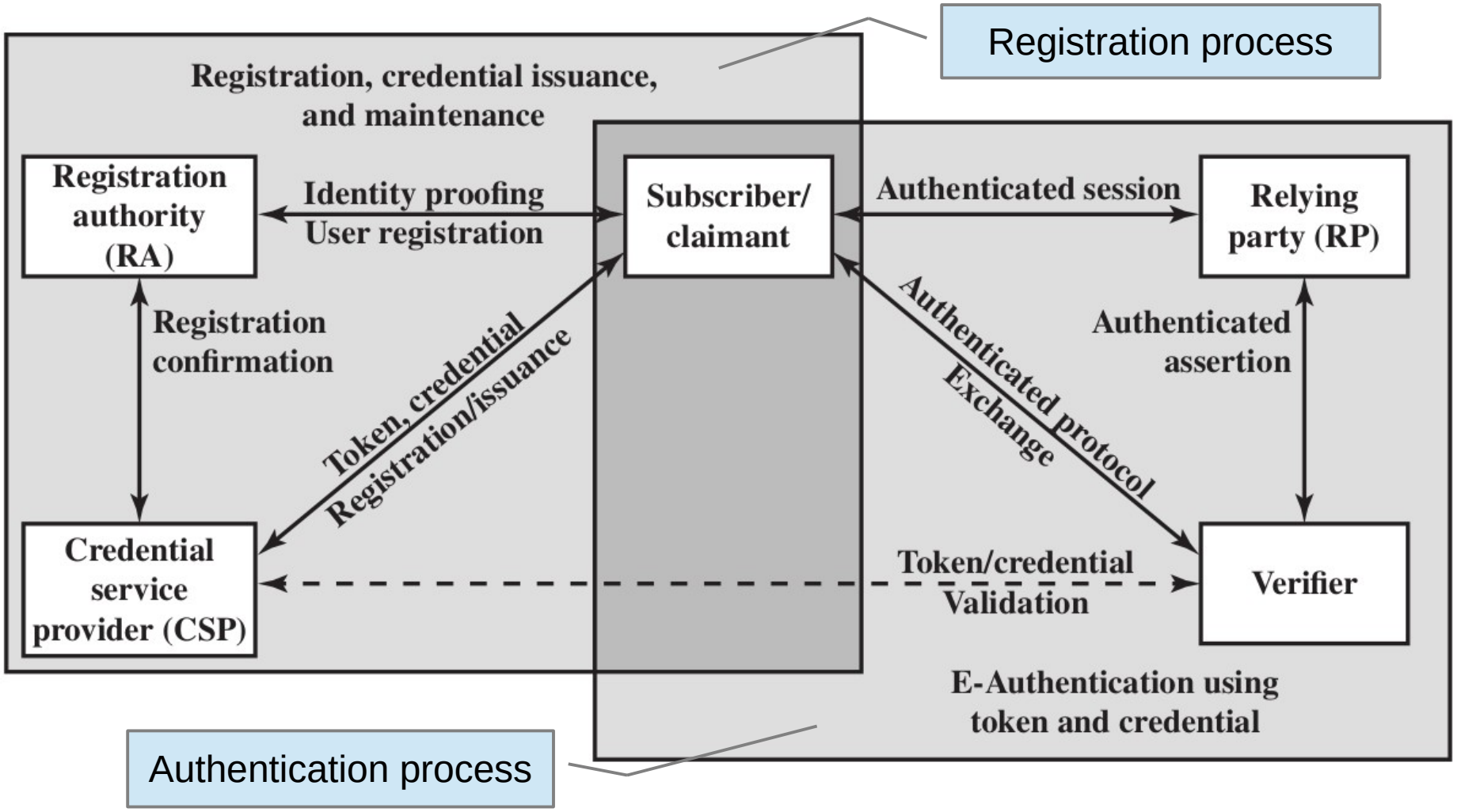
In a website AuthN determines if the user claiming to be Alice is the same person who created the account

Authorization (AuthZ) verifies what the user is allowed to do

After Alice is authenticated, her permissions include the authorization to access other users' profile? To delete them?

AuthZ is often referred to as Access Control

A Model for Digital User Authentication (NIST SP 800-63-3)



Authentication Factors

Something you know
(knowledge factors)
Password, PIN, answers
to prearranged questions

Something you have
(possession factors)
Tokens from electronic keycards,
smart cards, and physical keys

Something you are
(static inherence factors)
Static biometrics like
fingerprint, retina, and face

Something you do
(dynamic inherence factors)
Dynamic biometrics like voice pattern,
handwriting characteristics, and typing rhythm

Authentication Factors

Something you know
(knowledge factors)
Password, PIN, answers
to prearranged questions

may be guessed or stealed,
may be forgotten

Something you have
(possession factors)
Tokens from electronic keycards,
smart cards, and physical keys

may be forged or stealed,
may be lost

Something you are
(static inherence factors)
Static biometrics like
fingerprint, retina, and face

Something you do
(dynamic inherence factors)
Dynamic biometrics like voice pattern,
handwriting characteristics, and typing rhythm

false positives and false negatives,
user acceptance, cost, and convenience

Authentication Factors

Something you know
(knowledge factors)
Password, PIN, answers
to prearranged questions

may be guessed or stealed,
may be forgotten

Something you have
(possession factors)
Tokens from electronic keycards,
smart cards, and physical keys

may be forged or stealed,
may be lost

Something you are
(static inherence factors)
Static biometrics like
fingerprint, retina, and face

Something you do
(dynamic inherence factors)
Dynamic biometrics like voice pattern,
handwriting characteristics, and typing rhythm

false positives and false negatives,
user acceptance, cost, and convenience

Multifactor authentication
Use multiple authentication factors (not two times the same factor)

Password-based Authentication

Users provide credentials, a username and a password. The system compares the credentials with those stored, granting access if a match is found

Attack strategies and countermeasures

- Offline dictionary attacks
  - Prevent unauthorized access to the password file
  - Implement intrusion detection measures to identify a compromise
- Specific account attack
  - Lock out the account after a 5 failed login attempts (possibly just for some time)
- Popular password attack
  - Inhibit the use of common passwords
  - Monitor authentication requests to identify patterns
- Password guessing against single user
  - Make passwords difficult to guess (minimum length, character set, no well-known words)
- Workstation hijacking
  - Invalidate sessions after a period of inactivity
- Exploiting user mistakes (passwords written down, sharing of passwords, default passwords)
  - User training, intrusion detection, multifactor authentication
- Exploiting multiple password use
  - User training

Hashed Passwords

Don’t save users’ passwords!
Not in clear, not even encrypted

If a website can send you your password,
that website stores your password somewhere, and it’s bad!
If a website can check if your new password is similar to the previous five,
that website stores your last five passwords somewhere, and it’s bad!

**Don’t store the password,
store the hash value of the password
combined with a salt value!**

Make the hash function slow
(ask for the hash value of the
hash value of the hash value...)

Storing new credentials

Password

Salt

Randomly generated

Slow hash function

The hash value of the hash value of...

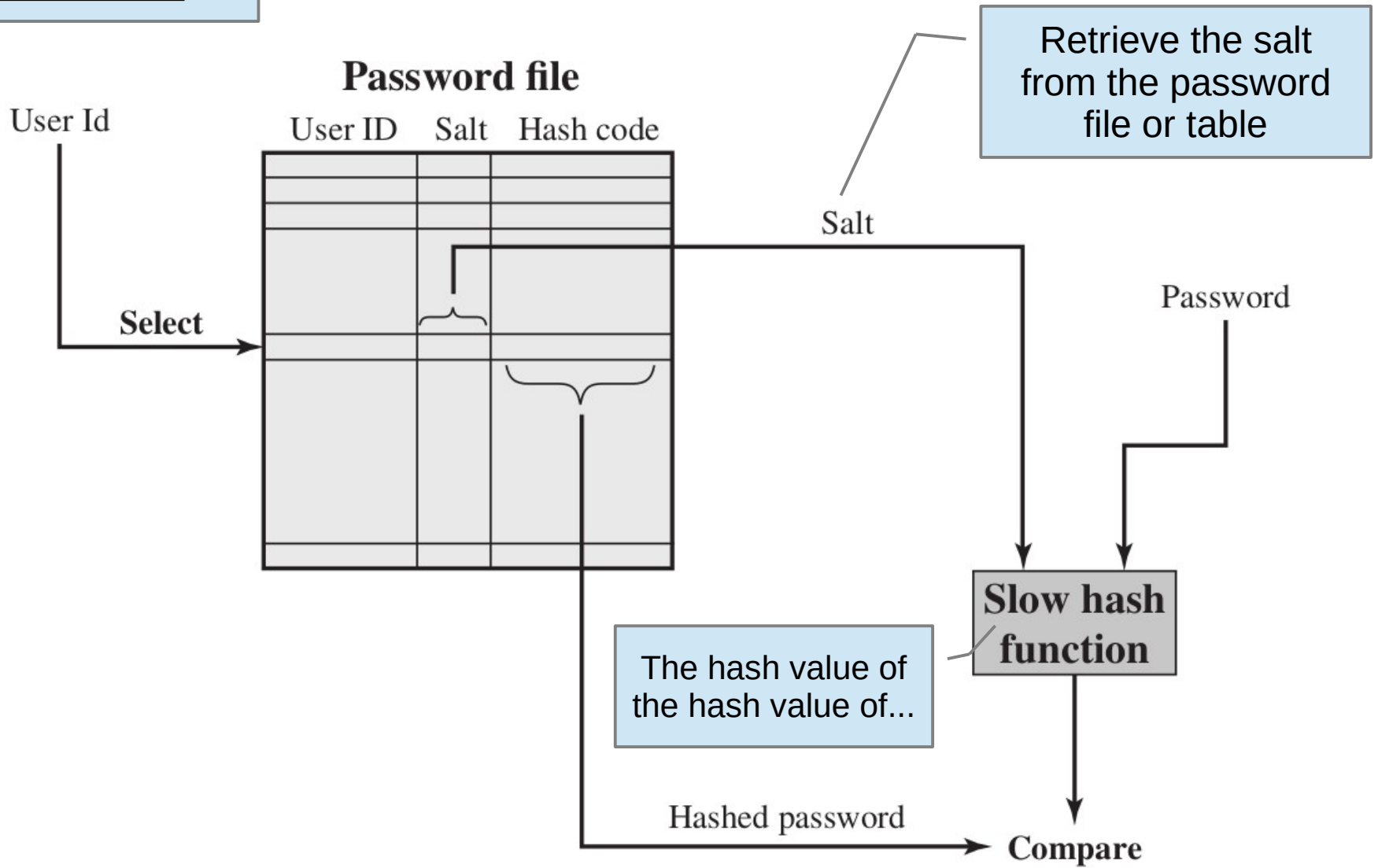
Load

Password file

User ID	Salt	Hash code
		•
		•
		•

Store the salt and the hash code, not the password!

Verifying credentials



Salt and hash purposes

Duplicate passwords will get different hash codes (if they get different salts)

Salt makes rainbow tables exponentially bigger (so they cannot be produced)

If the hash function is sufficiently slow,
brute-force cracking is practically impossible.
Modern password hash/salt scheme use Bcrypt,
which includes a cost variable
(the number of “the hash value of”)

Password Protocol

Client

Host

Hi! I'm user U

Hi U! Could you hash your password with *h()*, then combine with the **nonce** *r* and hash with *f()*?

U, User

U

r, random number
h(), *f*(), functions

(*r*, *h*(), *f*())

Sure! I will use my password P' and send what you asked for

P'

f(*r* , *h*(*P'*))

if *f*(*r* , *h*(*P'*)) = *f*(*r*, *h*(*P*(*U*)))
then yes else no

Let me do the same calculation using the hash code of the password of user U, h(P(U))

yes/no

The nonce protects against replay attacks

Token Protocol

Very similar to password protocol

Client

Host

U, User

U

r, random number
$h(), f()$, functions

$(r, h(), f())$

$P' \rightarrow W'$
password to
passcode via token

$f(r, h(W'))$

if $f(r, h(W')) = f(r, h(W(U)))$
then yes else no

yes/no

User password is used to generate a token (static passcode or one-time random passcode)

The host stores the static passcode h(W(U)) or can generate the OTP W(U)

How do AuthN vulns arise?

Weak AuthN mechanisms
(eg. no protection against brute-force attacks)

Logic flaws leading to bypass AuthN
(also referred to as **broken authentication**)

Vulnerabilities in password-based login

Users either register for an account themselves or
they are assigned an account by an administrator.

This account is associated with a unique username and a secret password,
which the user enters in a login form to authenticate themselves.

An attacker must not obtain or guess login credentials.

https://portswigger.net/web-security/authentication/password-based

Brute-force attacks

Try different credentials,
randomly chosen, from wordlists,
from OSINT, with mutations, …

Usernames
Often predictable, eg.
mario.alviano@unical.it
alviano@mat.unical.it

Any good guess for Francesco Panceza?

Avoid easily guessable usernames for high-privileged accounts, eg. admin, administrator, root, god, …
(maybe use a random suffix)

Brute-force credentials is easier if usernames can be enumerated!
Different status codes, error messages or response times
for existing and nonexisting usernames lead to username enumeration.

Flawed brute-force protection

Slow down login attempts
- Lock the attacked account
- Block attacker's IP address

Possible implementation flaw
Account is locked after 5 attempts, reporting it to the user.
Use it for username enumeration!

Possible implementation flaw
IP attempt counter is reset after successful login.
Bypass by alternating brute-force attack and successful login to a known account.

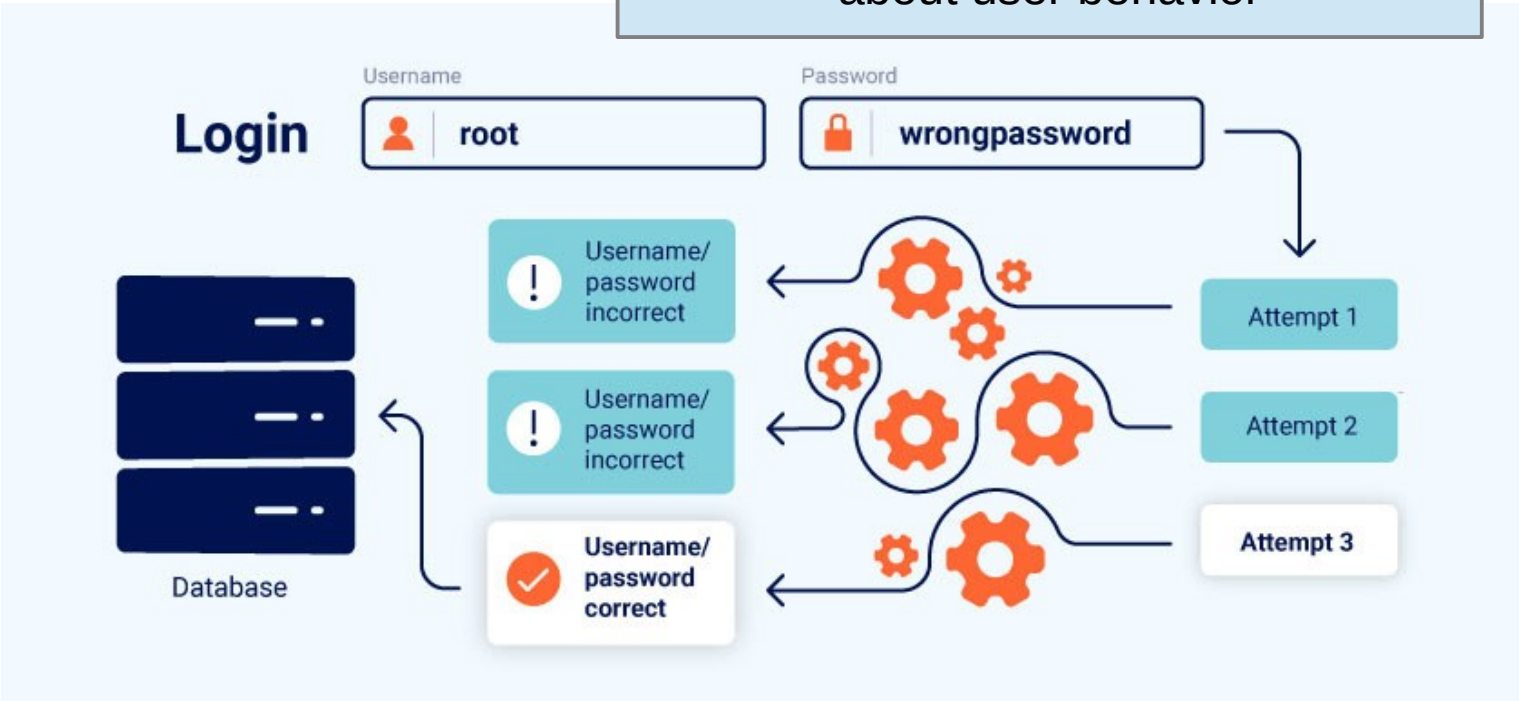
Prevention

- Take care with user credentials
  - Don’t store passwords
  - Use HTTPS (redirect any HTTP request to HTTPS)
- Don't count on users for security
  - Implement an effective password policy
- Prevent username enumeration
  - Generic and identical error messages for invalid username or password
- Implement robust brute-force protection
  - Slow down bro’! Implement CAPTCHA test with every login attempt
- Triple-check your verification logic
  - Use well established libraries (don’t implement AuthN by yourself!)
- Don't forget supplementary functionality
  - Should users be able to register by themselves?
  - Is password reset/change properly implemented?
- Implement proper multi-factor authentication
  - Two is better than one… **two different factors**!

https://portswigger.net/web-security/authentication/securing

Business logic vulnerabilities

They arise from flawed assumptions about user behavior



Different response times for incorrect username (attempt 1), incorrect password (attempt 2) and correct credentials (attempt 3).
Use it for username enumerations, and then to brute-force passwords!

https://portswigger.net/web-security/logic-flaws

Flawed assumption

User will provide data exclusively via a web browser.

Let's rely only on client-side validation! (SO BAD!)

Use ZAP to skip client-side validation.
Use Javascript Console to skip client-side validation.
Use Python Requests to skip client-side validation.

When an attacker deviates from the expected user behavior, the application fails to handle the situation safely

Examples

- Excessive trust in client-side controls
- Failing to handle unconventional input
- Making flawed assumptions about user behavior
- Domain-specific flaws
- Providing an encryption oracle

Let’s have a look at them…
now or the next time

https://portswigger.net/web-security/logic-flaws/examples

Prevention

Business logic vulnerabilities are often specific of a given application.

To prevent them
- Developers and testers must understand the domain of the application
- No implicit assumptions about user behavior
- No implicit assumptions about the behavior of other parts of the application

Domain-Driven Design (DDD)?

Questions