Broken Access Control

Tackling OWASP's #1 Vulnerability

#whoami

Spas Genov

Senior QA Automation @NewsUK

Security enthusiast

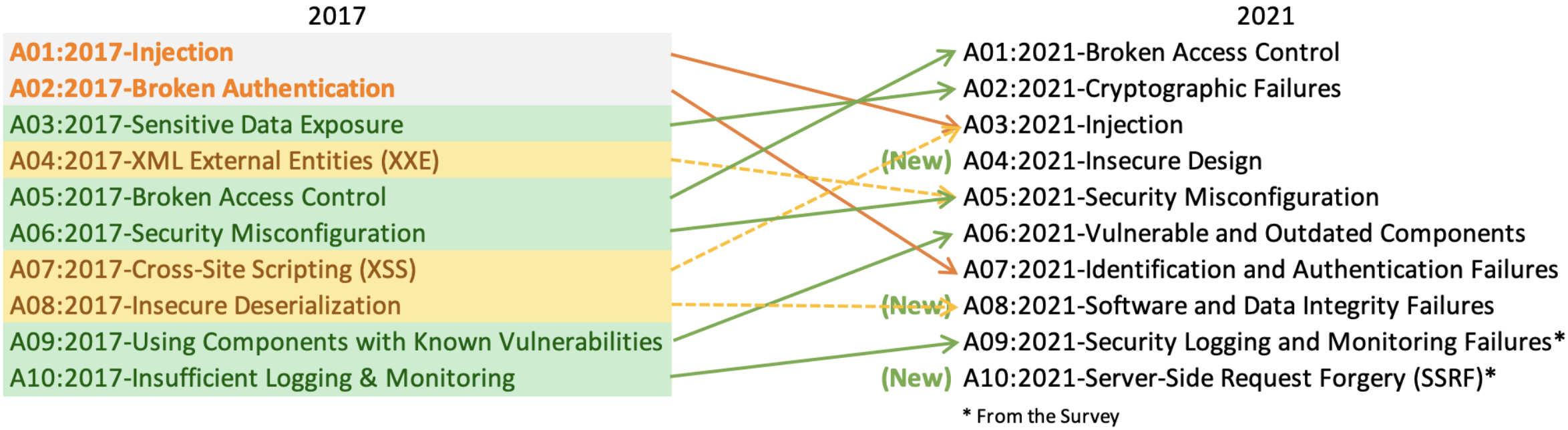
#whoami

/Spas-Genov (LinkedIn)

Senior QA Automation @NewsUK

W4nn4b3 h4x0r (eWPT, eJPT)

OWASP Top 10:2021



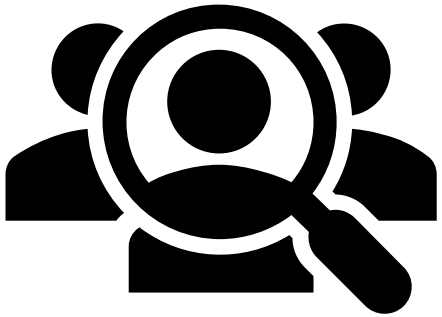

Terminology Confusion
Broken Aecess
Conirol IBAC)
Level Access
(aoA)

Terminology Confusion

- Broken Access Control (BAC)
- Missing Function Level Access Control (MFLAX)
- Broken Object Level Access (BOLA)
- Insecure Direct Object Reference (IDOR)
- Forceful browsing
- Authorization/Authentication issues

Core Concepts

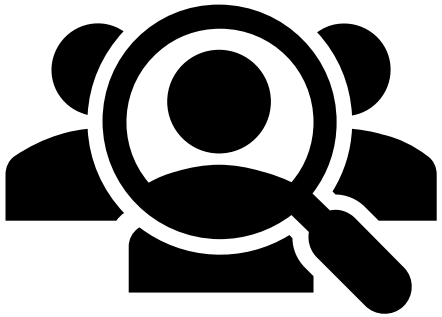
- **Authentication -** Verifies a user's identity (through username/password)



Username: admin
Password: sexylubo99

Core Concepts

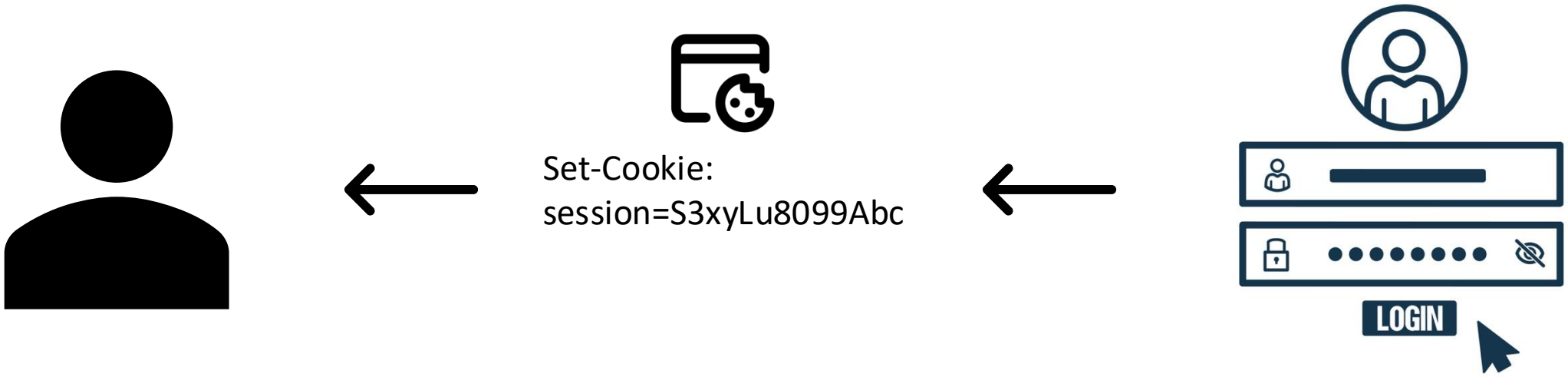
- **Session Management** - Maintains the user's authenticated state across requests (using session tokens)



Username: admin
Password: sexylubo99

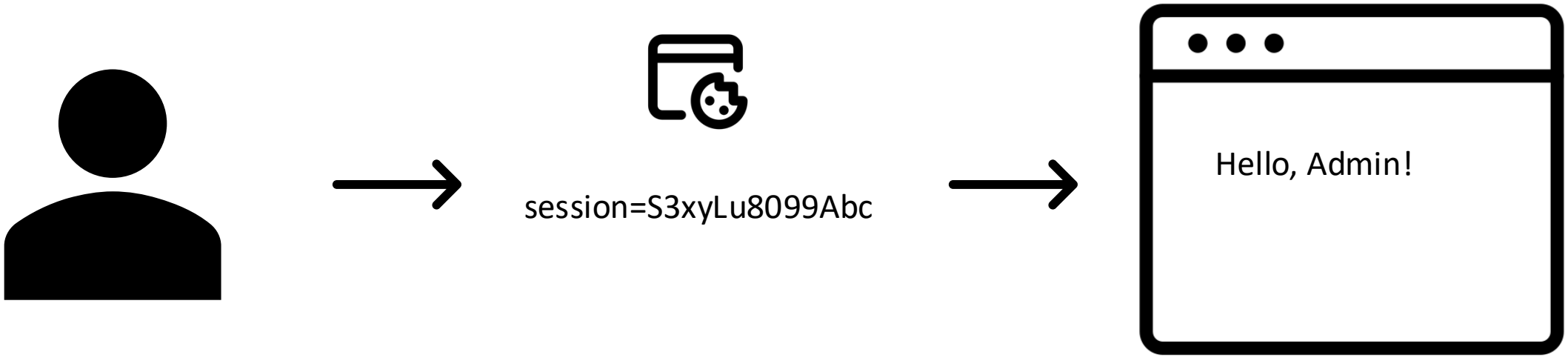
Core Concepts

- **Session Management** - Maintains the user's authenticated state across requests (using session tokens)



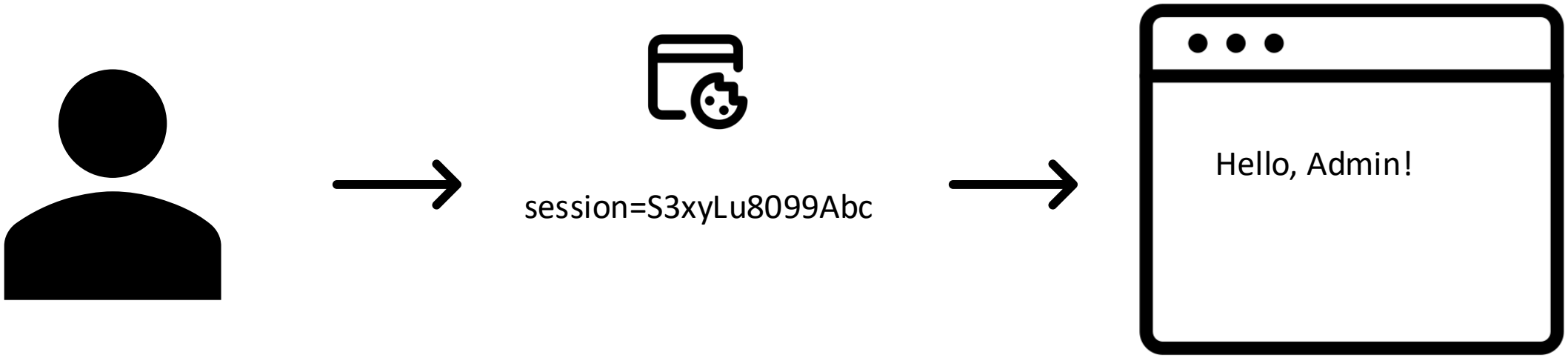
Core Concepts

- **Session Management** - Maintains the user's authenticated state across requests (using session tokens)



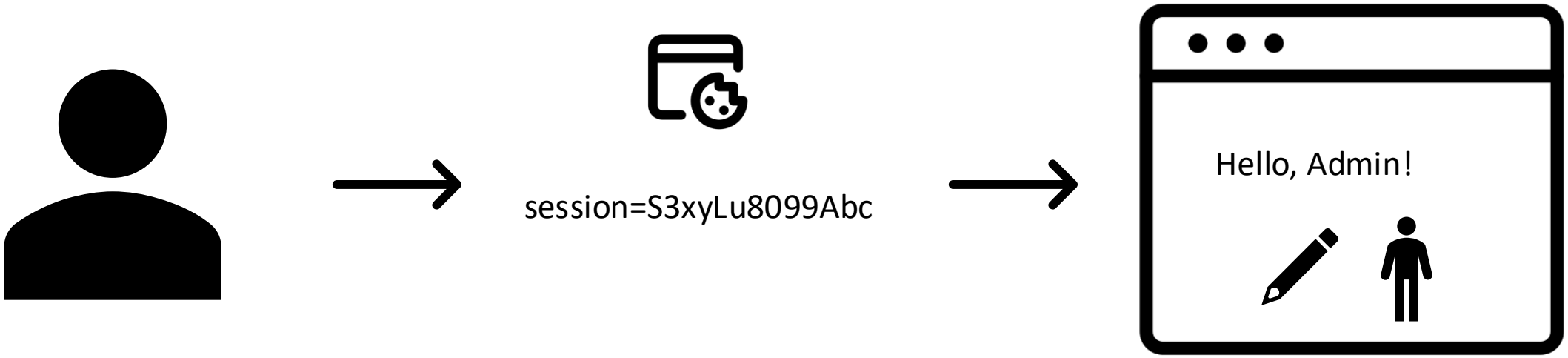
Core Concepts

- **Access Control** - Determines if an authenticated user has permission to perform specific actions



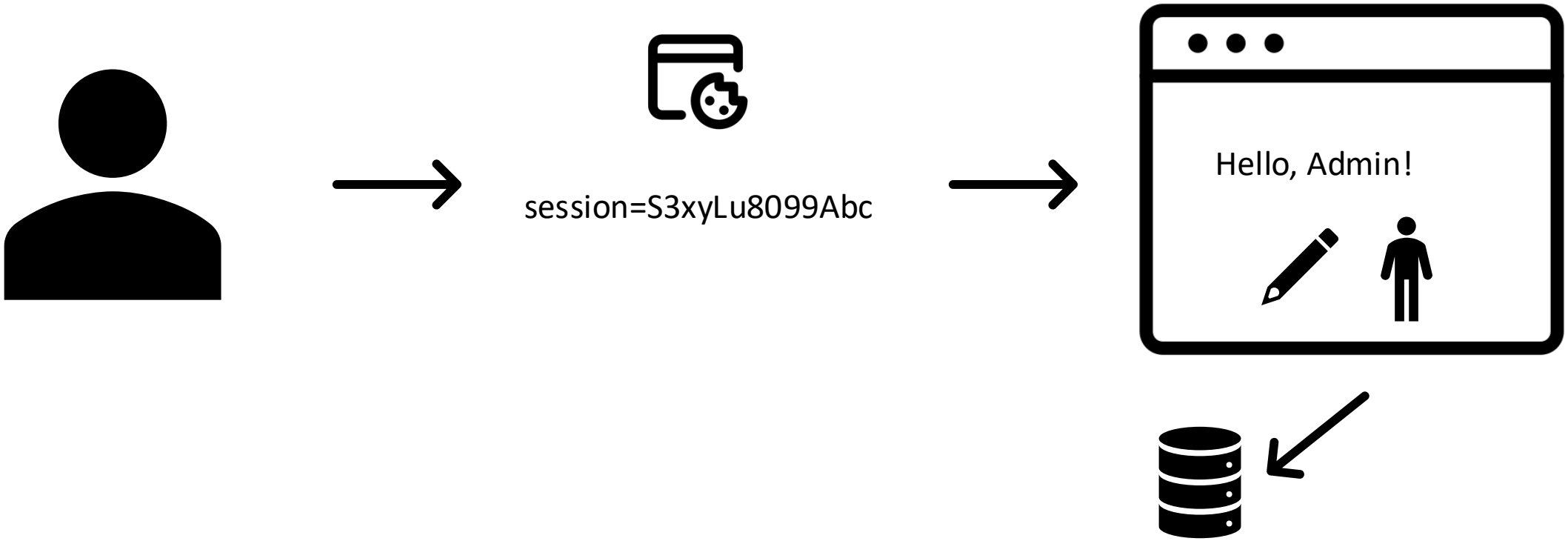
Core Concepts

- **Access Control** - Determines if an authenticated user has permission to perform specific actions



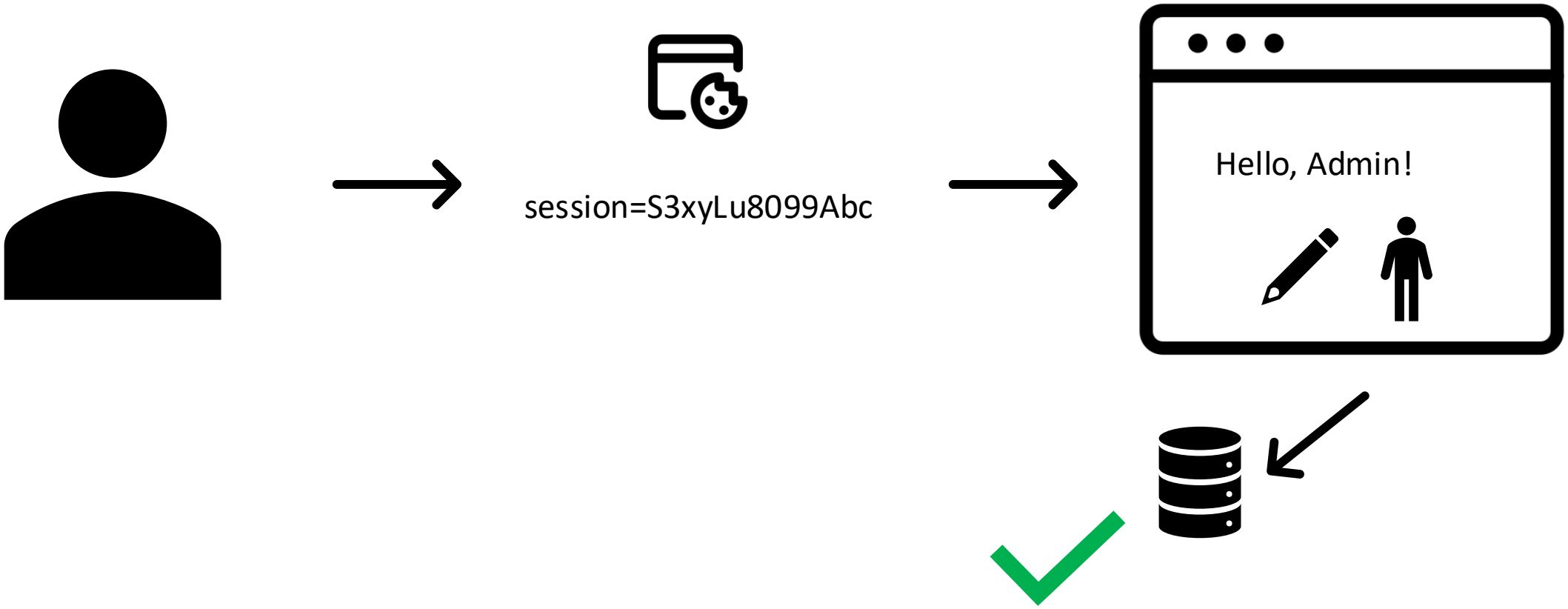
Core Concepts

- **Access Control** - Determines if an authenticated user has permission to perform specific actions



Core Concepts

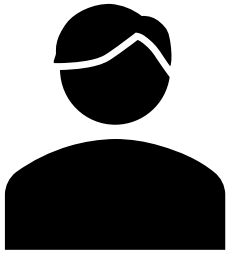
- **Access Control** - Determines if an authenticated user has permission to perform specific actions



Types of Access Control

- **Vertical Access Control:** Restricts access based on privilege levels (e.g., admin vs. regular user)

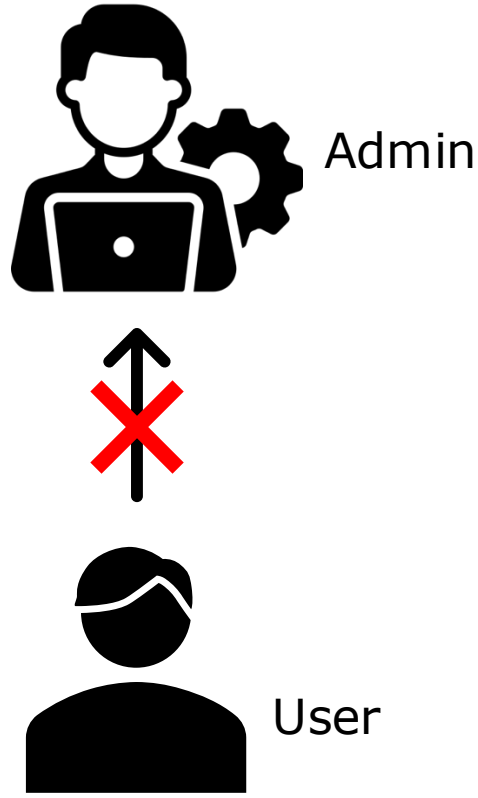
Admin



User

Types of Access Control

- **Vertical Access Control:** Restricts access based on privilege levels (e.g., admin vs. regular user)

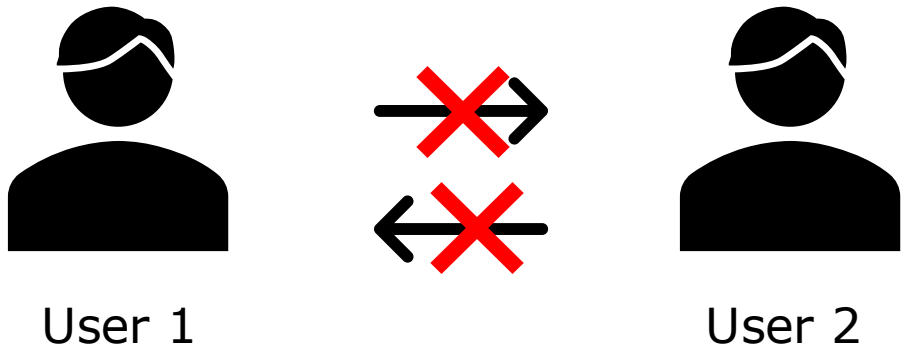


Types of Access Control

- **Horizontal Access Control:** Prevents users of the same privilege level from accessing each other's resources

Types of Access Control

- **Horizontal Access Control:** Prevents users of the same privilege level from accessing each other's resources

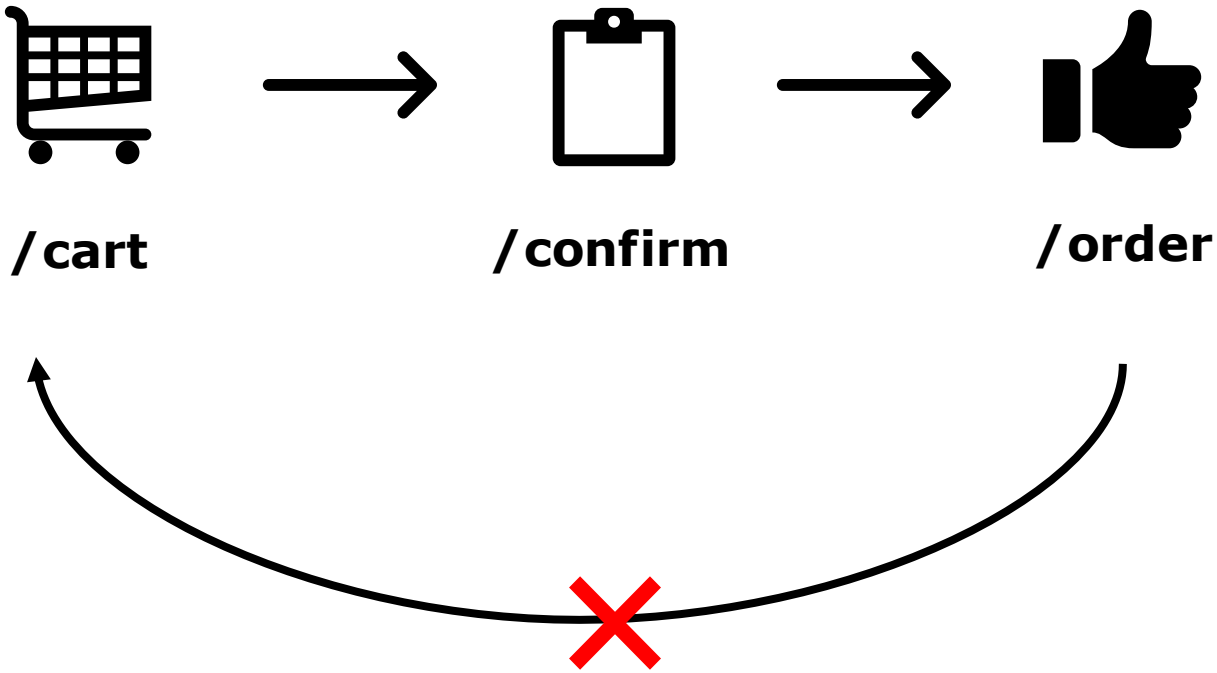


Types of Access Control

- **Context-Dependent Access Control:** Restricts access to actions or resources based on application state or user interaction sequence

Types of Access Control

- **Context-Dependent Access Control:** Restricts access to actions or resources based on application state or user interaction sequence



BAC 

Broken Access Control vulnerabilities

occur when applications fail to properly restrict what users can do, allowing them to act outside their intended permissions.

BAC 

Broken Access Control vulnerabilities occur when applications fail to properly restrict what users can do, allowing them to act outside their intended permissions.

These security failures typically result in three major consequences:

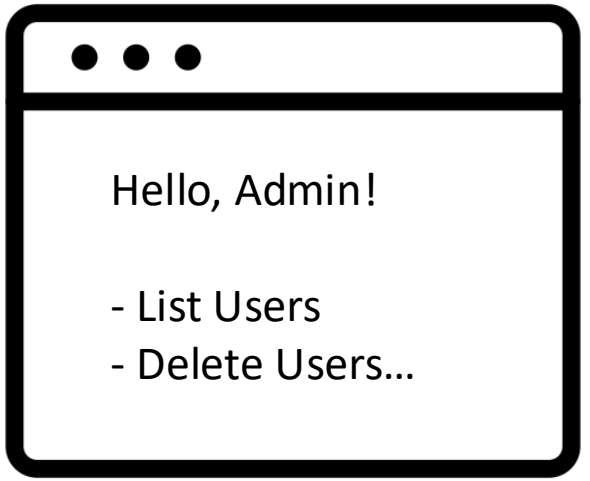
1. Improper access to restricted functionality
2. Unauthorized disclosure of sensitive information
3. Unauthorized modification or destruction of data

Vertical Privilege Escalation

- Users gain access to higher privilege functionality, for example, a regular user accessing the admin panel. Often exploited by manipulating parameters that control access

Admin 

https://site.com/login/home.php?admin=true
https://site.com/login/home.php?role=1



Vertical Privilege Escalation

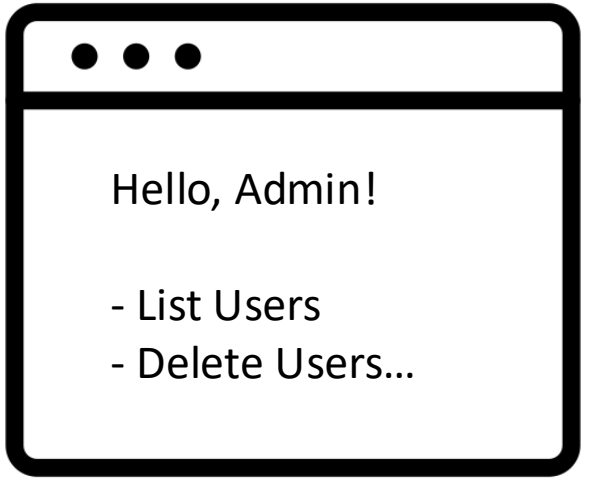
- Users gain access to higher privilege functionality, for example, a regular user accessing the admin panel. Often exploited by manipulating parameters that control access

Admin 

https://site.com/login/home.php?admin=true
https://site.com/login/home.php?role=1

Attacker 

https://site.com/login/home.php
https://site.com/login/home.php



Vertical Privilege Escalation

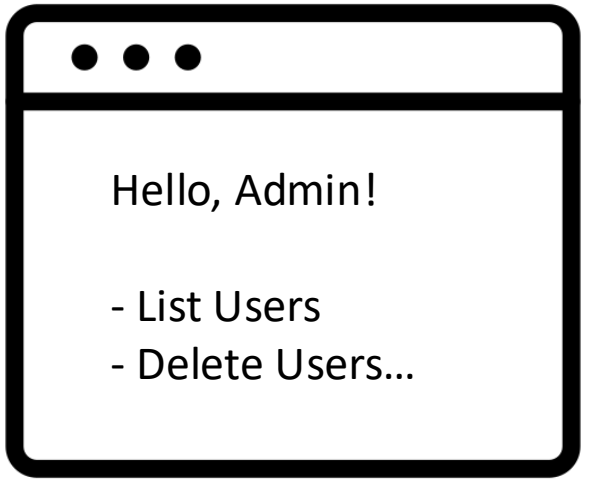
- Users gain access to higher privilege functionality, for example, a regular user accessing the admin panel. Often exploited by manipulating parameters that control access

Admin 

https://site.com/login/home.php?admin=true
https://site.com/login/home.php?role=1

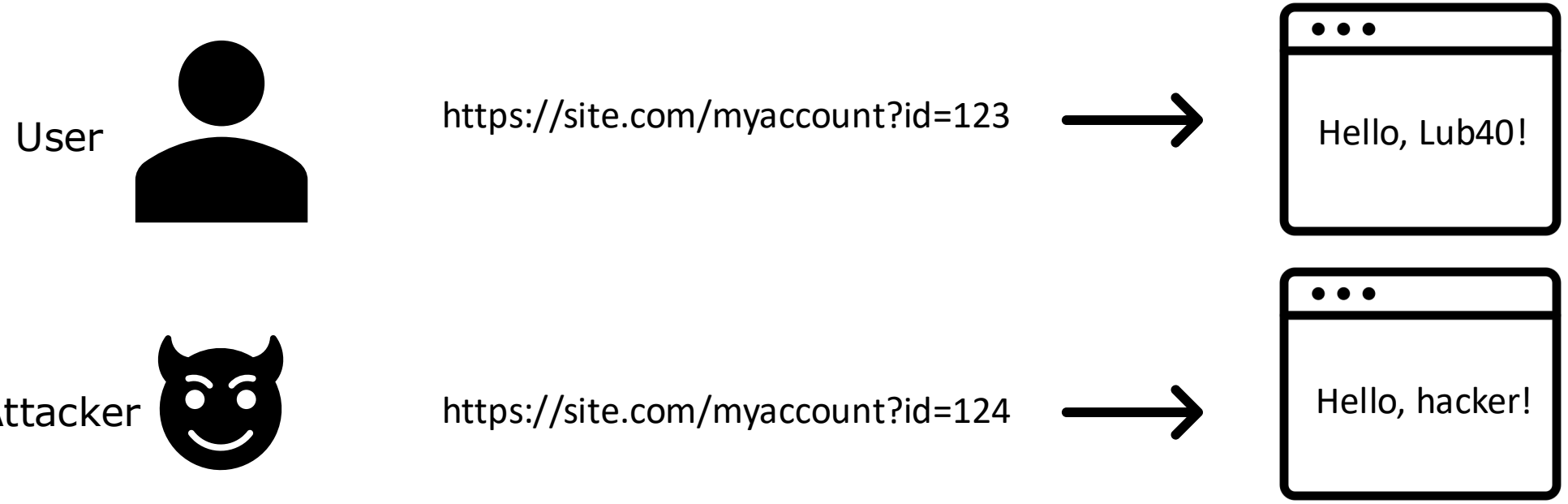
Attacker 

https://site.com/login/home.php?admin=true
https://site.com/login/home.php?role=1



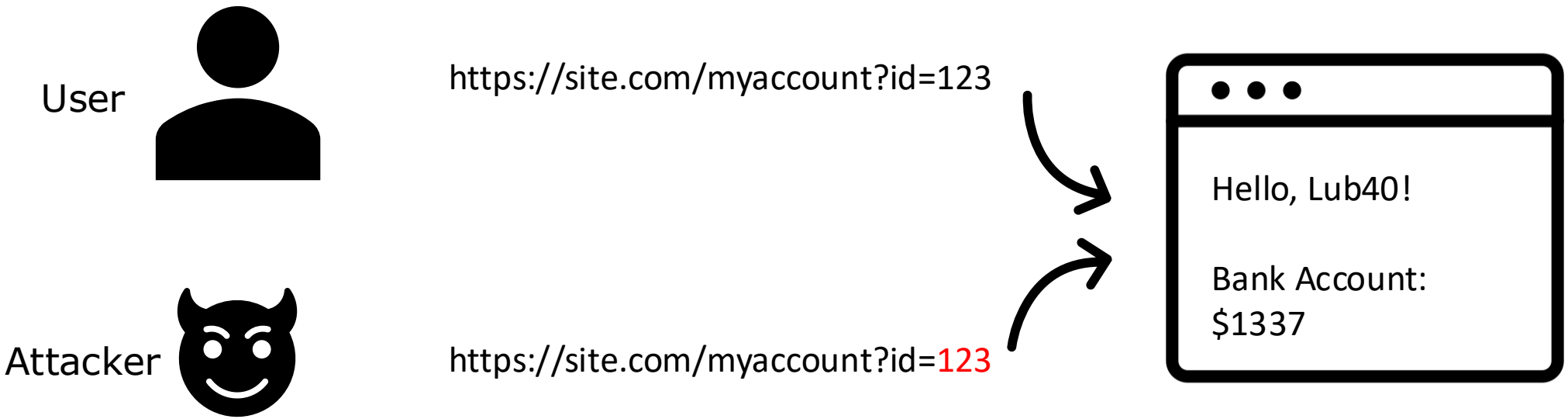
Horizontal Privilege Escalation

- Users access resources belonging to other users of the same privilege



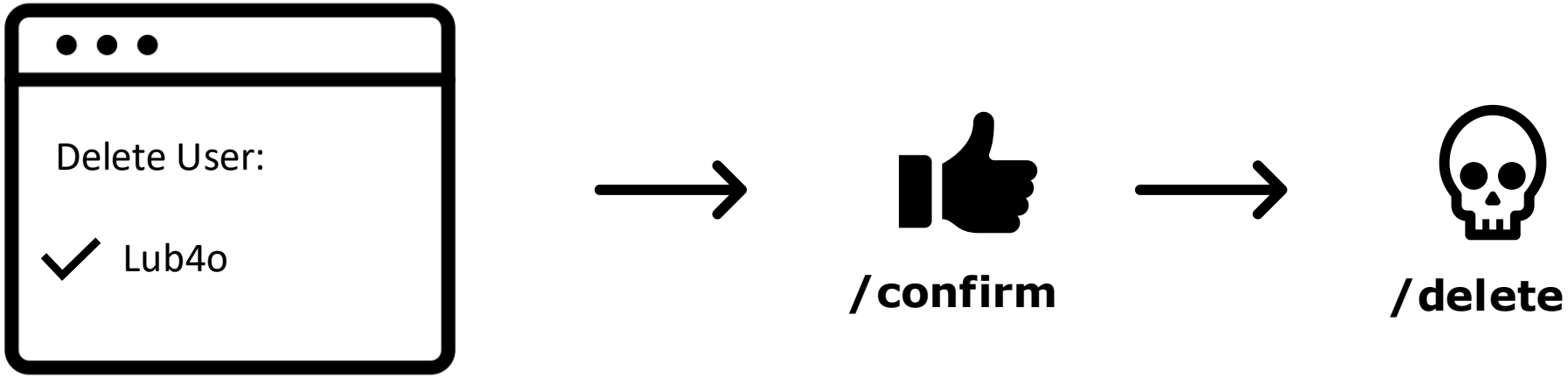
Horizontal Privilege Escalation

- Users access resources belonging to other users of the same privilege



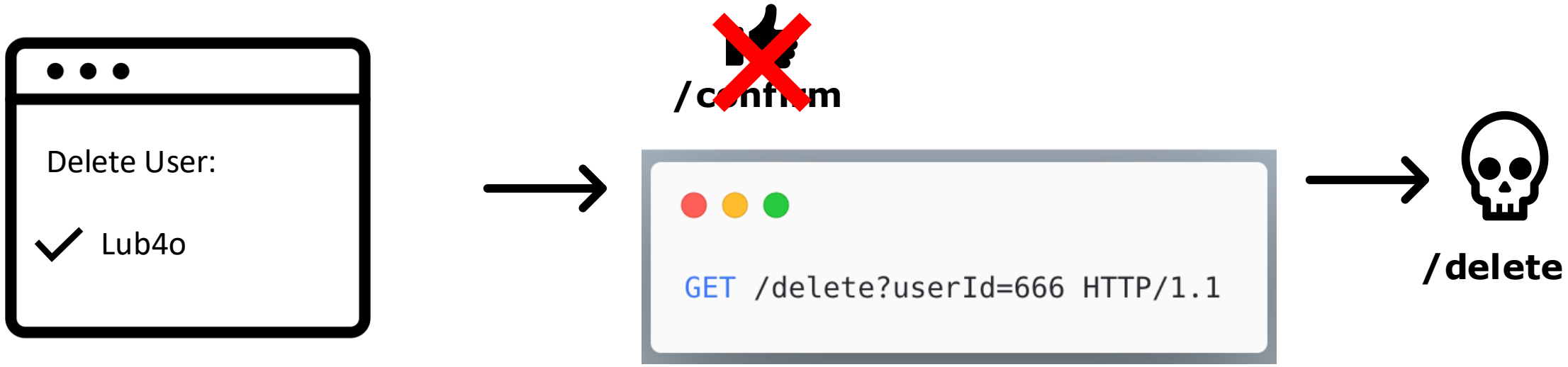
Multi-Step Process Vulnerabilities

- Access controls implemented on some steps but missing on others
- Attackers bypass the intended sequence
- Direct request to endpoint with missing controls



Multi-Step Process Vulnerabilities

- Access controls implemented on some steps but missing on others
- Attackers bypass the intended sequence
- Direct request to endpoint with missing controls



BAC Terminology

- **IDOR (Insecure Direct Object Reference):** Application exposes object IDs that can be manipulated to access unauthorized resources
- **BOLA (Broken Object Level Authorization):** Accessing objects belonging to other users by modifying identifiers
- **MFLAX (Missing Function Level Access Control):** Regular users accessing admin-only functionalit
- **Authentication Bypass:** Accessing protected resources while unauthenticated
- **Cross-Tenant Access:** Accessing another tenant's data in multi-tenant applications
- **Forceful browsing** - Directly accessing restricted URLs, bypassing intended navigation paths

Access Control (Testing Cheat Sheet)

1. **Parameter Manipulation**:🔢
   - Numeric ID substitution (**userId=1** → **userId=2**)
   - Email address substitution in endpoints
   - GUID/UUID substitution for non-sequential IDs
   - Hash value substitution

2. **Request Modification**:📝
   - Change HTTP methods (GET to POST, POST to /PUT/PATCH/DELETE...)
   - Modify cookies or session tokens
   - Alter hidden form fields

Access Control (Testing Cheat Sheet)

1. **Path & Endpoint Techniques**: 🔍
   - Direct access to privileged endpoints
   - Path traversal attacks
   - Forced browsing via case changes (/admin/ vs /ADMIN/)
   - Static file enumeration
   - Direct function calls with parameters

2. **Process Manipulation :** ⚙️
   - Bypass intended sequence in multi-step processes
   - Skip authorization steps

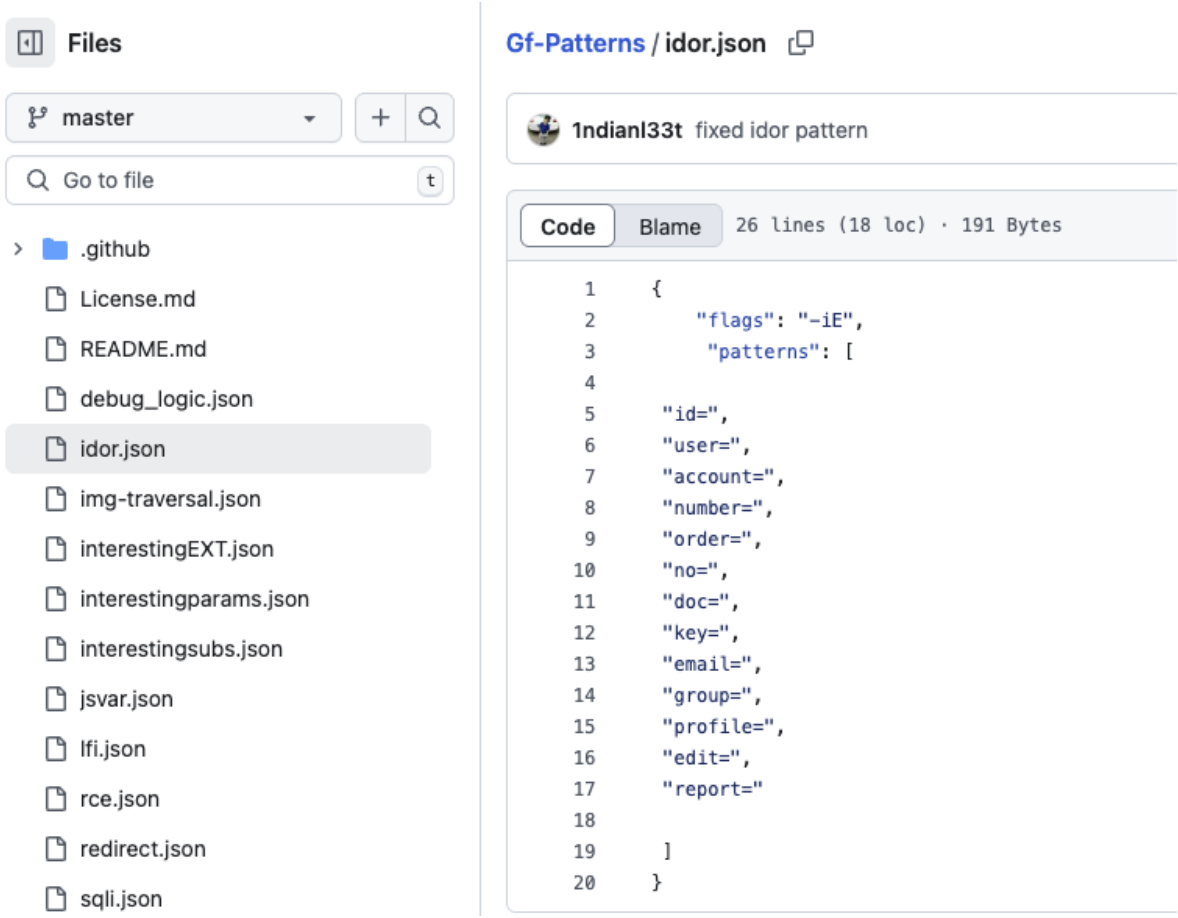
IDOR PARAMS

Common parameters:

id	**doc**
user	**key**
account	**email**
number	**group**
order	**profile**
no	**edit**

Functions:

Change mail/password
Upgrade/downgrade user role
Create/remove/update/delete context specific app data
Shipping, invoices and document viewing



https://github.com/1ndianl33t/Gf-Patterns/blob/master/idor.json

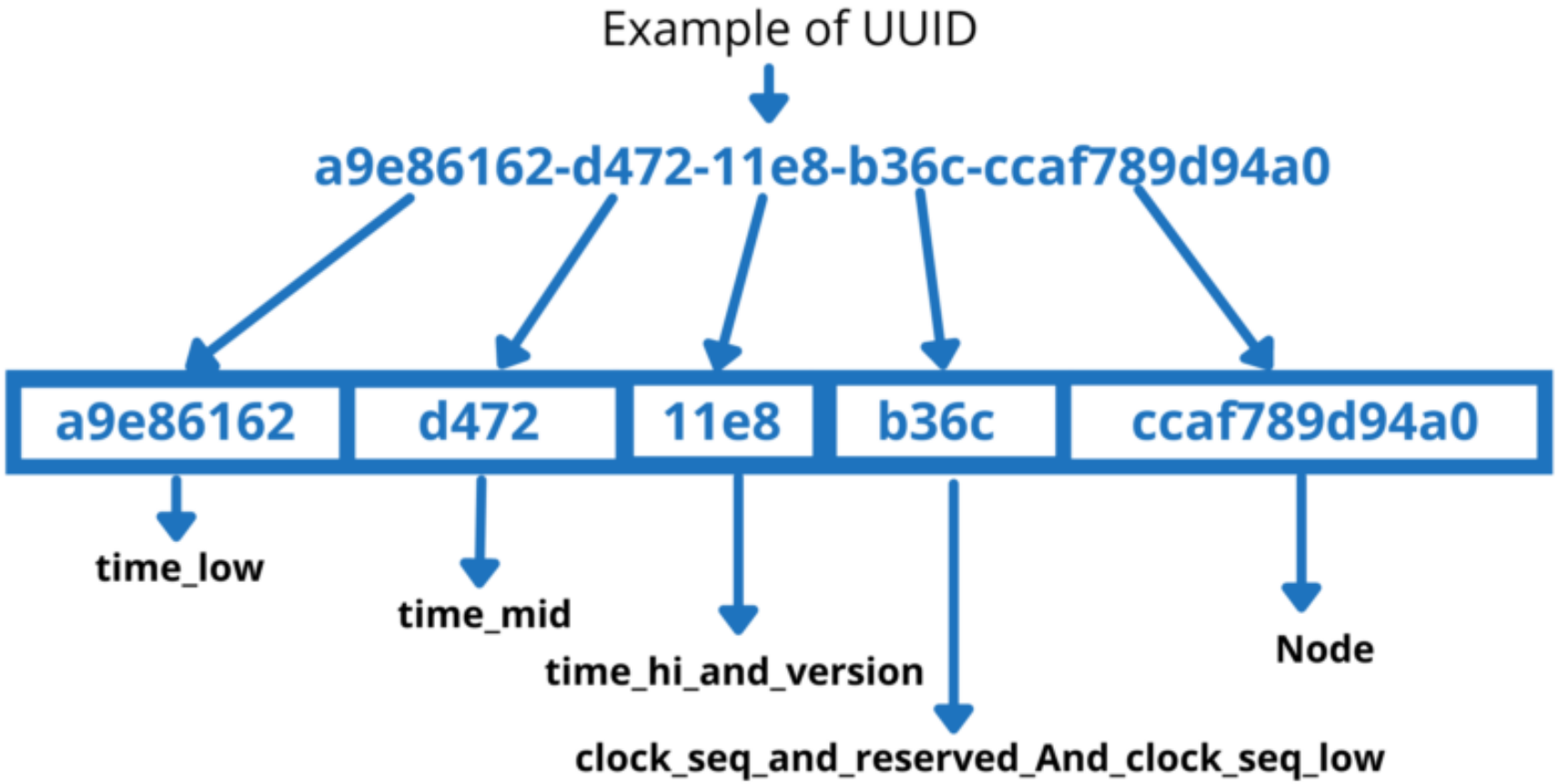
IDOR

```
POST /account/updatepasswd HTTP/1.1
Host: site.com
Connection: Close
Content-Length: 22
Cache-Control: max-age=0
Origin: https://site.com
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9, image/webp, image/apng
ٲ/*;9-0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=3214536754363414df3142gf2341

userid=1337&action=updatepasswd
```

Hash based IDOR

```
POST /account/updatepasswd HTTP/1.1
Host: site.com
Connection: Close
Content-Length: 22
Cache-Control: max-age=0
Origin: https://site.com
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9, image/webp, image/apng
ٯ/*;9-0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=3214536754363414df3142gf2341

userid=912134131a7611f2dfee0b92bf6b0eed&action=updatepasswd
```

UUID = No bug?


Example of UUID
a9e86162-d472-11e8-b36c-ccaf789d94a0
a9e86162
d472
11e8
b36c
ccaf789d94a0
time_low
time_mid
time_hi_and_version
clock_seq_and_reserved_And_clock_seq_low
Node

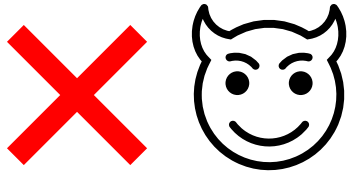
UUID = No bug?

User accounts identified by UUIDs like:
8f1a2b3c-4d5e-6f7g-8h9i-0j1k2l3m4n5o

The developers believed this approach was secure because:

- UUIDs are not sequential or predictable
- The extremely large keyspace (2^128 possibilities) makes brute-forcing impossible
- UUIDs don't reveal information about the user like sequential IDs might

- **Account data access**:
  /account-details?uuid=8f1a2b3c-4d5e-6f7g-8h9i-0j1k2l3m4n5o
- **Password reset functions**:
  /reset-password?user=8f1a2b3c-4d5e-6f7g-8h9i-0j1k2l3m4n5o



The Hidden Vulnerability

The application had an **"account linking"** feature that allowed users to link their accounts with family members. When viewing linked accounts, the application returned a JSON response (**UUID Leakage**)

```
{
  "primary_account": {
    "name": "Lub4o",
    "uuid": "8f1a2b3c-4d5e-6f7g-8h9i-0j1k2l3m4n5o",
    "linked_accounts": [
      {
        "name": "Mary Jane",
        "relationship": "spouse",
        "uuid": "9a8b7c6d-5e4f-3g2h-1i0j-9k8l7m6n5o4p"
      }
    ]
  }
}
```

The Hidden Vulnerability

The application had an **"account linking"** feature that allowed users to link their accounts with family members. When viewing linked accounts, the application returned a JSON response

```
POST /change-contact-info?uuid=9a8b7c6d-5e4f-3g2h-1i0j-9k8l7m6n5o4p HTTP/1.1
Host: finance.site.com
Cookie: session=spz_session_cookie

{"email":"malicious@attacker.com"}
```

Testing Approaches – Black Box

1. Map the application thoroughly
2. Identify potential parameters used for access control
3. Test with multiple accounts of different privilege levels
4. Manipulate parameters systematically
5. Use automation tools like Burp Authorize extension

Testing Approaches – White Box Testing

1. Review code for access control implementation
2. Look for these red flags:
   - Violations of principle of least privilege
   - Missing/weak access control checks
   - Missing controls on certain API methods
   - Trust on client-side input for access decisions
3. Validate findings on running application

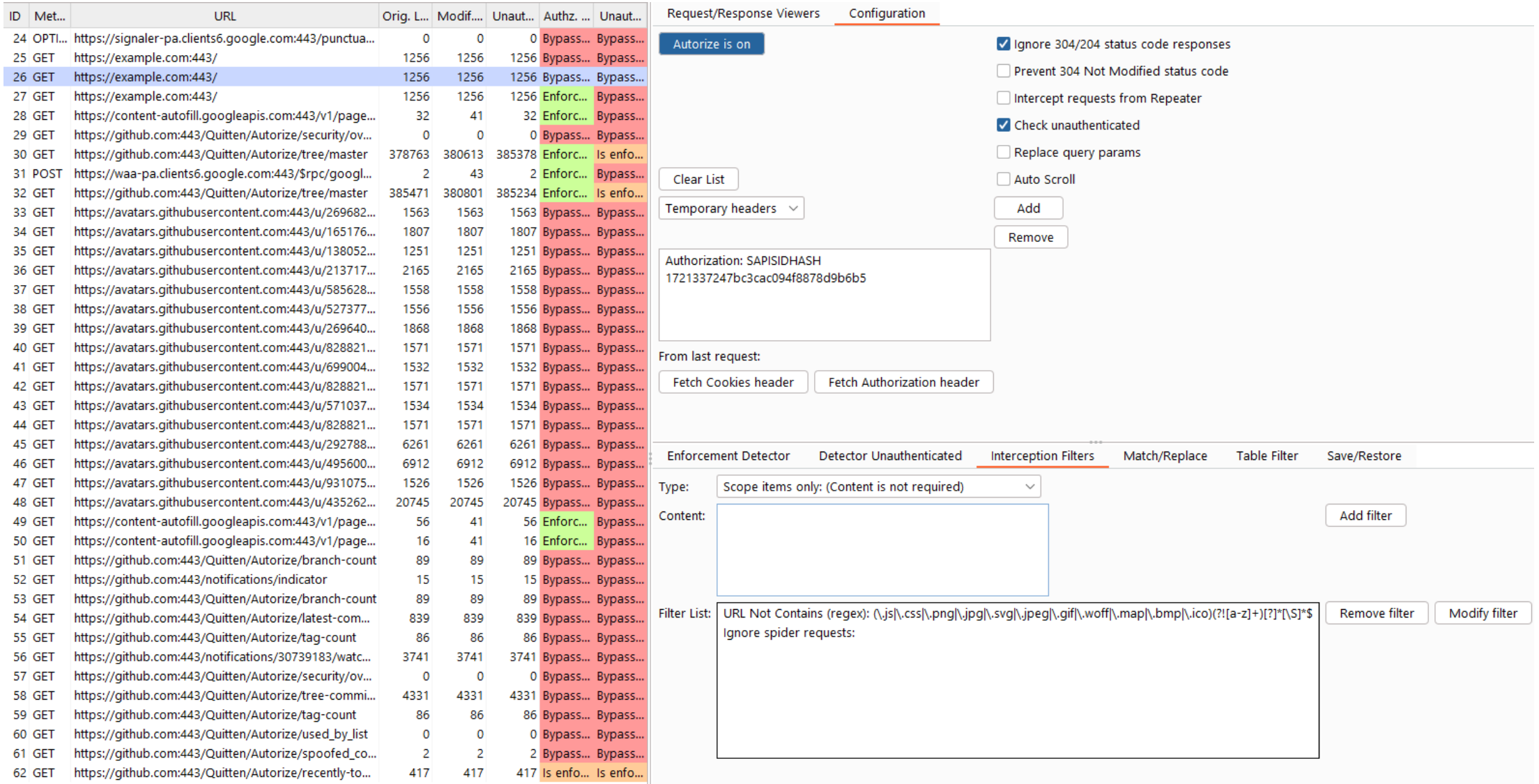
Testing Matrix

	Update Password	Update Email	Change Account Data	Upgrade Account to Admin	View Logs
Admin	yes	yes	yes	yes	yes
User	yes	yes	no	no	no
Unauthenticated	no	no	no	no	no

Text = Should they be able to do it?

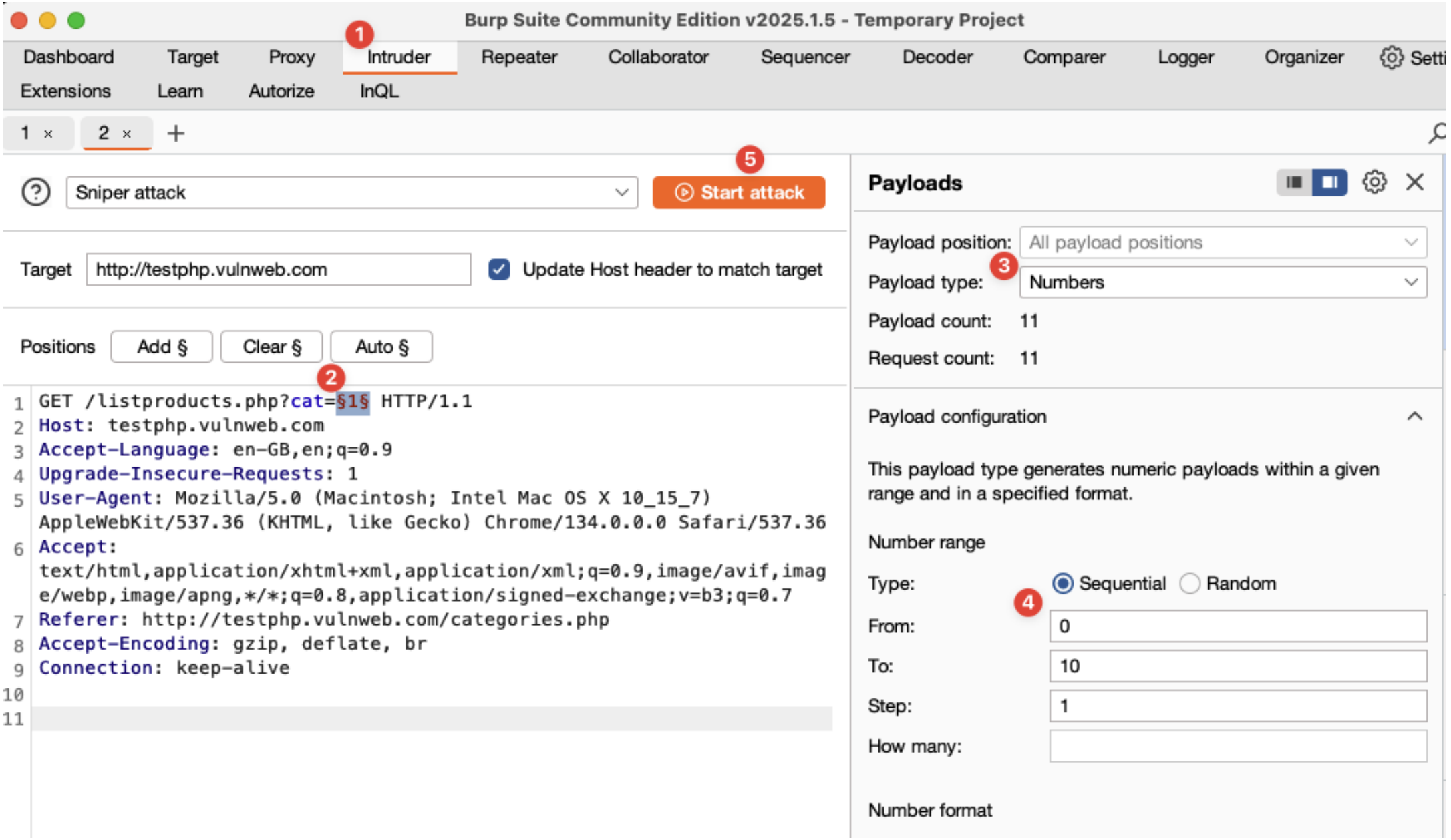
Color = could they do it? (red is bad)

Burp Suite Add-ons: Autorize, AutoRepeater, AuthMatrix, Authz

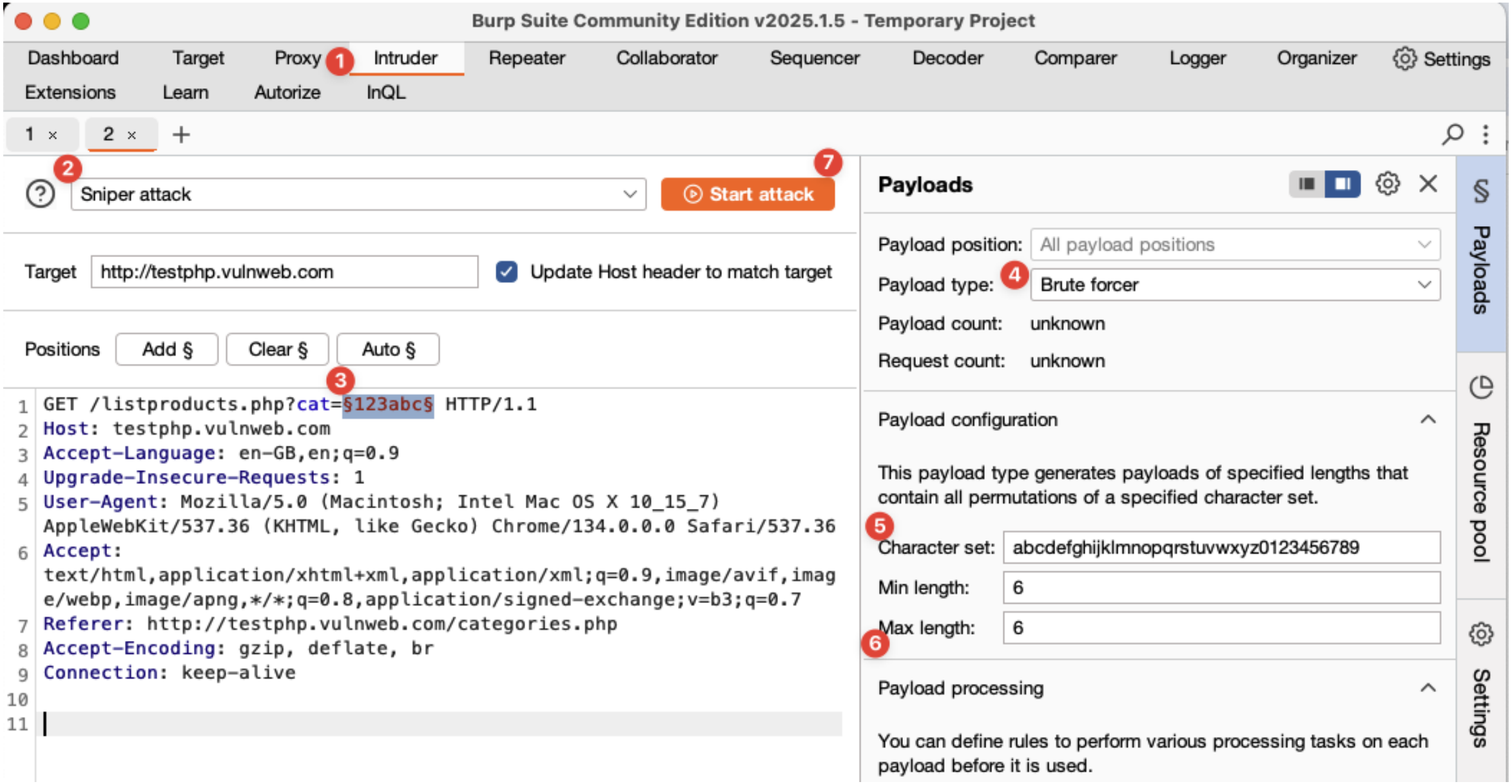


“Easy IDOR hunting with Autorize?” by InsiderPHD https://www.youtube.com/watch?v=2WzqH6N-Gbc

Burp Suite Intruder – Sniper Attack (Numbers)



Burp Suite Intruder – Sniper Attack (Brute Force)



Preventing Broken Access Control

1. **Design Defensively** 
   - Implement centralized access control 
   - Deny access by default 
   - Principle of least privilege 

2. **Validate Server-Side** 
   - Never trust client inputs 
   - Verify permissions for every request 
   - Validate resource ownership 

3. **Build Robust Controls** 
   - Use mature authorization frameworks 
   - Implement role/attribute-based access control 
   - Log and monitor access attempts 

Thank You!

Questions? I'm here to help! 🙋💪

Resources:

Web Security Academy - Access Control

- https://portswigger.net/web-security/access-control

Web Application Hacker's Handbook

- Chapter 8 - Attacking Access Controls

OWASP Top 10 - A01 Broken Access Control

- https://owasp.org/Top10/A01_2021-Broken_Access_Control/

OWASP Web Security Testing Guide - Authorization Testing

- https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/05-Authorization_Testing/README

OWASP Application Security Verification Standard - V4 Access Control

- https://owasp.org/www-pdf-archive/OWASP_Application_Security_Verification_Standard_4.0-en.pdf