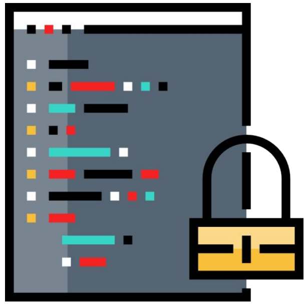
Secure Coding: Fix From The Root



About me

- Product security Engineer
- wisdomfreak1
- Bug bounty Hunter
- Traveler

As a Developer

1. Understand Requirements
    a. Understanding the project requirements, including functional specifications.
2. Plan and design
    a. outlining the overall structure, database schema, and technology stack to be used.
3. Code Development
    a. Write clean, efficient, and maintainable code following coding standards and best practices.
4. Testing and Quality Assurance
    a. Develop and execute comprehensive testing strategies, including unit tests.

But what about Security?

आएं

A Scenario

"Rahul bought a 1 year car insurance from Icici Lombard at 20000 Rs"

Breaking the assumptions

"Break Each and Every assumption made by the system."

The assumptions

1. Rahul pays “Rs. 20000” for the insurance
2. Rahul pays for “ICICI Lombard” car insurance
3. “Rahul” pays the insurance
4. Rahul pays Rs 20000 for “one year” car insurance
5. Rahul pays Rs 20000 for a one year “Car insurance”

Bugs Found in past

1. Account takeover of any user
2. Buying Things for free
3. Bypass transaction fee
4. Buying products and get full refund (not fraud)
5. Making myself as an Owner of an org

Common Vulnerabilities & Fixes

1. SQL Injection - Use prepared statements or parameterized queries.
2. XSS - Encode user-generated content before rendering it to the page.
3. CSRF - Implement anti-CSRF tokens
4. IDOR - proper access controls and authorization checks for every request.
5. File upload - check magic bits, file extension, content type and so on ...

In this talk will cover

1. Mass Assignment
2. Race Condition
3. State management
4. Insecure Direct Object Reference (IDOR)
5. Cross Site Scripting
6. Business logic Issues
    a. Workflow based

Mass Assignment

Mass Assignment: Ruby on Rails, NodeJS
Autobinding: Spring MVC, ASP NET MVC
Object injection: PHP

Mass Assignment

In order to streamline the development process and enhance productivity, many frameworks offer convenient mass-assignment functionality. This empowers developers to efficiently inject a complete set of user-entered data from a form directly into an object or database.

Example

1. <u>form for editing a user's account information</u>

```
<form>
    <input name="userid" type="text">
    <input name="password" type="text">
    <input name="email" text="text">
    <input type="submit">
</form>
```

2. <u>Object that the form is binding to</u>

```
public class User {
  private String userid;
  private String password;
  private String email;
  private boolean isAdmin;
}
```

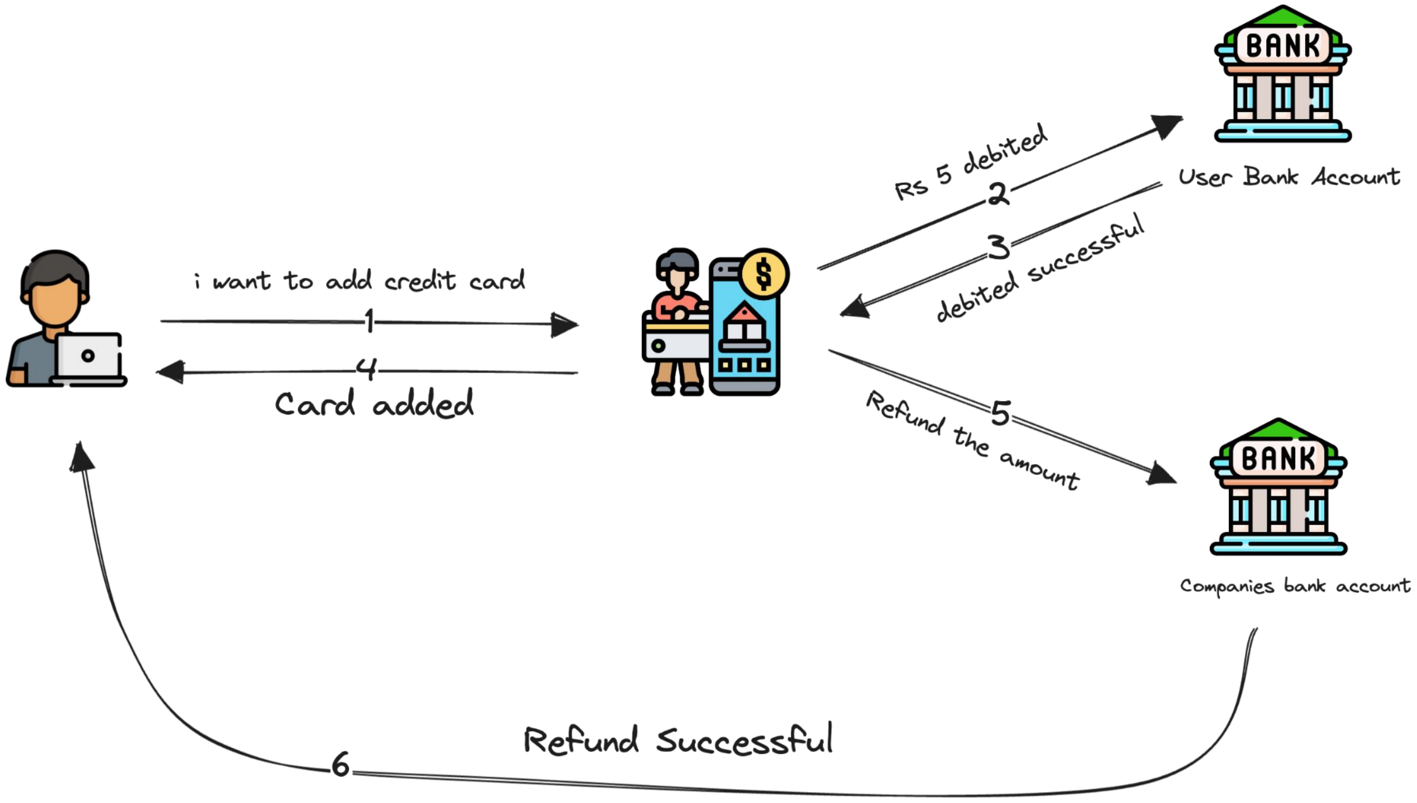
3. <u>Http Request</u>

```
POST /update/
...
userid=nullcon&password=heker@786&email=nullcon@heker.com
&isAdmin=true
```

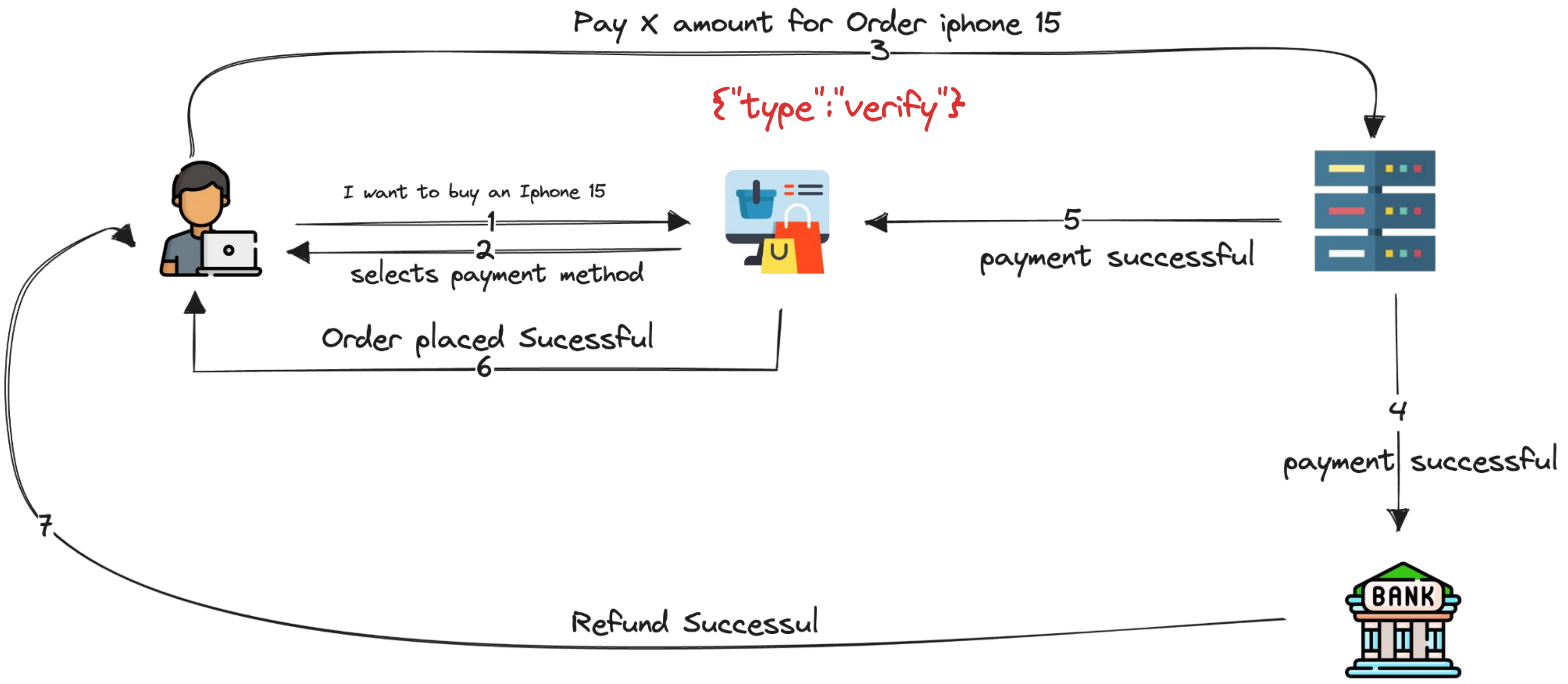
Case Study

1. Refund total amount for product
2. Changing role from Normal user to Owner

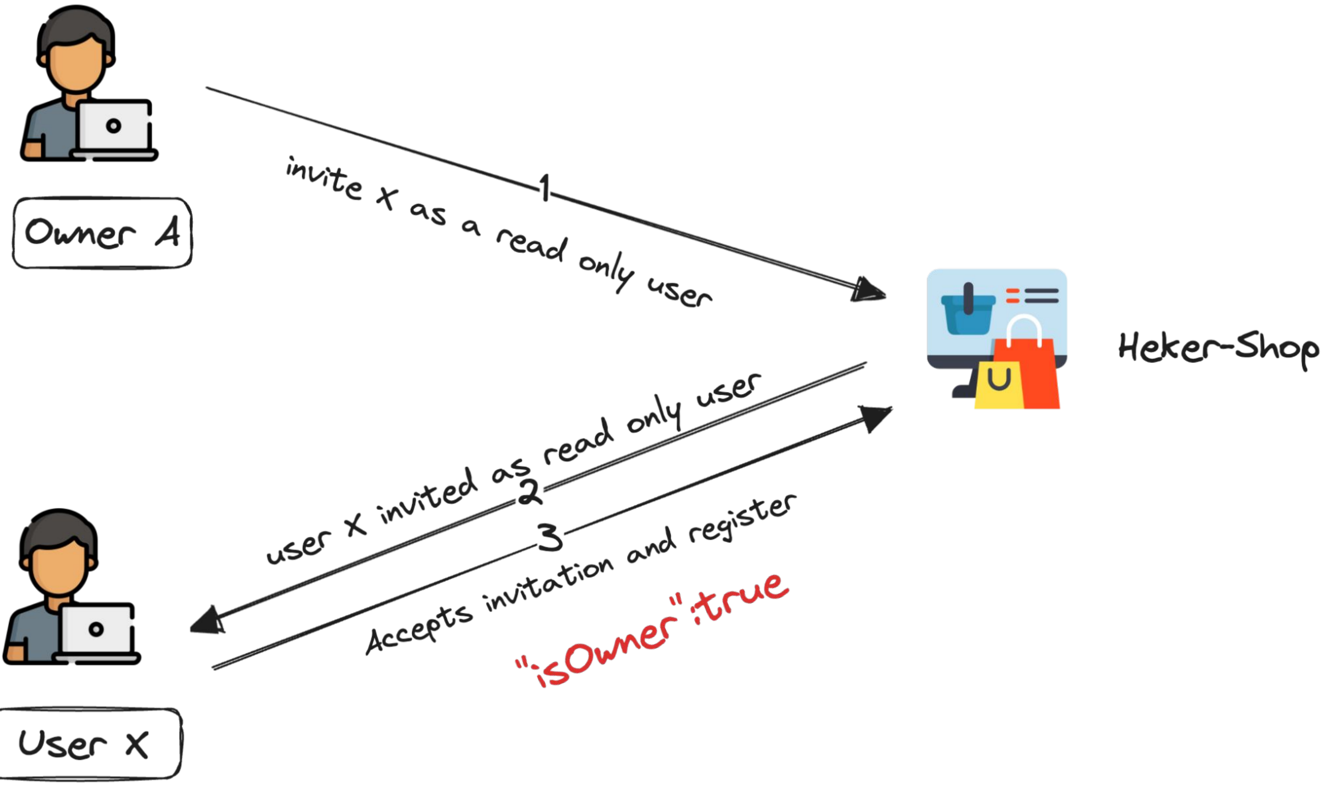
Refund total amount for product



Refund total amount for product



Changing role from User to Owner


Owner A
1
invite x as a read only user
Heker-Shop
user x invited as read only user
2
3
Accepts invitation and register
"isOwner":true
User X

Bug Fix

1. Only the fields that are meant to be editable by the user are included in the DTO

```
public class UserRegistrationFormDTO {
 private String userid;
 private String password;
 private String email;

 // isAdmin field is not present
}
```

Bug Fix

2. Language & Framework specific solutions
   a. Allow-listing
   b. Block-listing

Allow-listing

```
@Controller
public class UserController
{
   @InitBinder
   public void initBinder(WebDataBinder binder,
WebRequest request)
   {

binder.setAllowedFields(["userid","password","email"
]);
   }
...
}
```

Block-listing

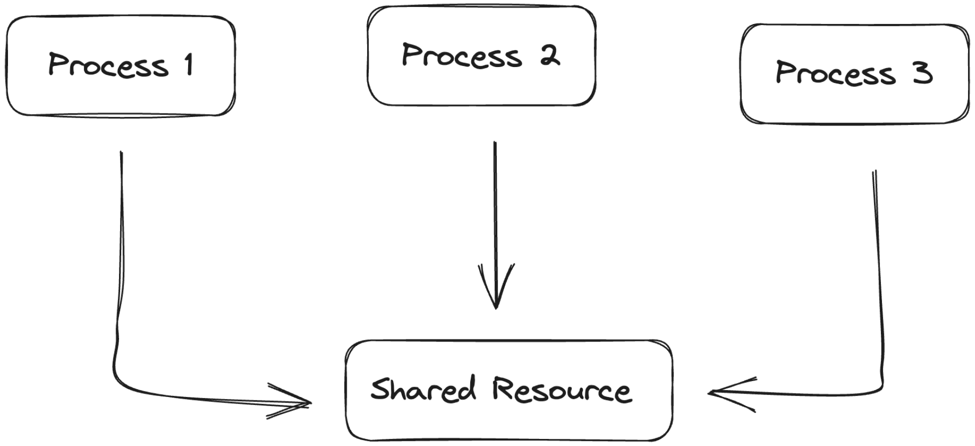
```
@Controller
public class UserController
{
  @InitBinder
  public void initBinder(WebDataBinder
binder, WebRequest request)
  {

binder.setDisallowedFields(["isAdmin"]
);
  }
...
}
```

Race Condition

Race Condition

when the behaviour of a program becomes unpredictable due to the timing of different threads or processes. It's like a "race" where multiple threads compete to access and modify shared resources or critical sections of code.

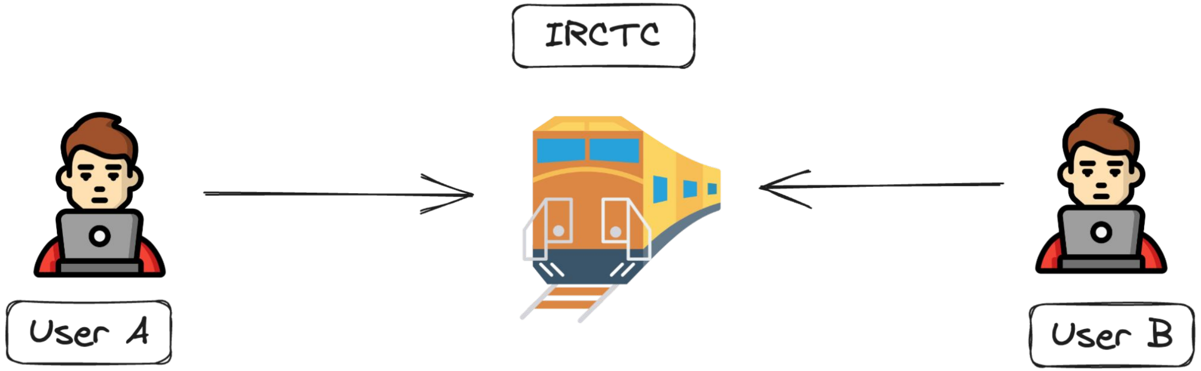


Race Condition

Example:

In a scenario where the last seat (S1 - 12L) is available, both **User A** and **User B** of IRCTC are concurrently attempting to book the seat.

Due to a race condition, both users will be assigned the same seat, resulting in a conflicting booking for S1 - 12L.



Case Study

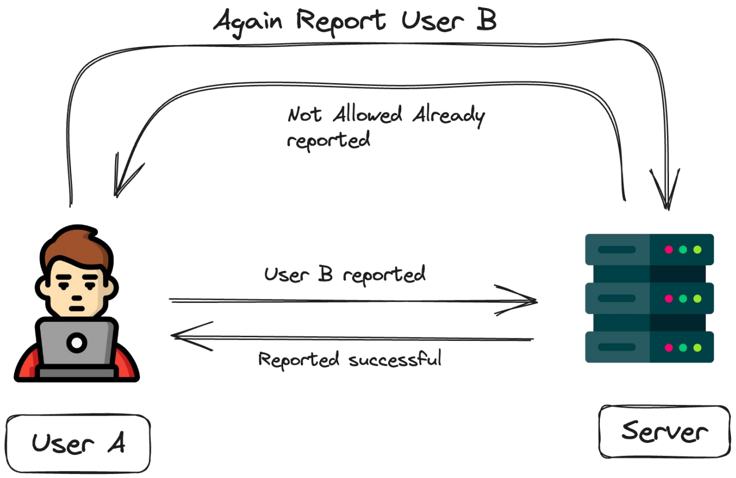
1. Block user
2. Chaining of 2 bugs to get N amount

Credits: https://www.youtube.com/watch?v=tKJzsaBIZvI&t=41s

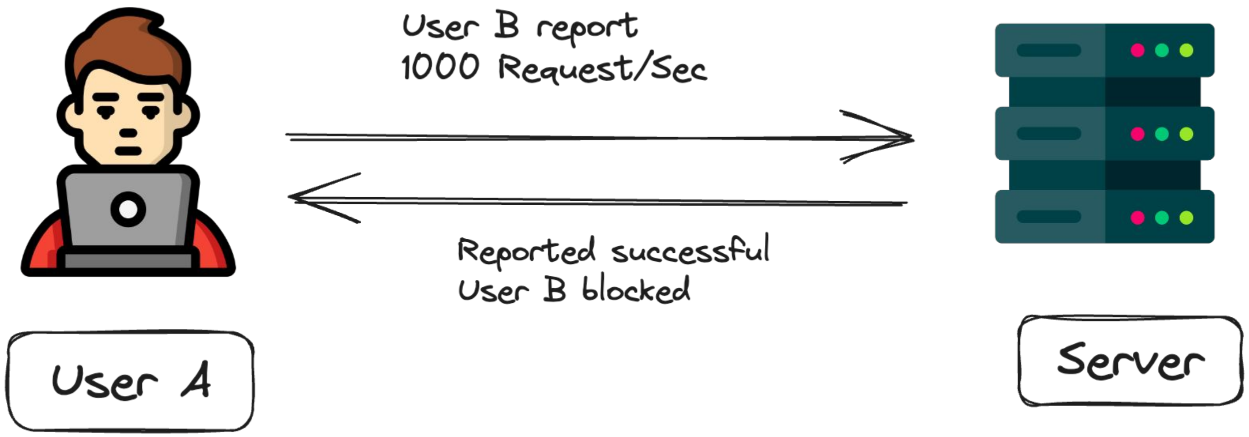
User Blocked

Happy Flow

While going through the community rules and regulation of the website i found that a user will be block if someone reports more than 10 time temporary 2 hrs. 50 times for a day and if 100 time user will be blocked permanently.



The Bug


User B report
1000 Request/Sec
Reported successful
User B blocked
User A
Server

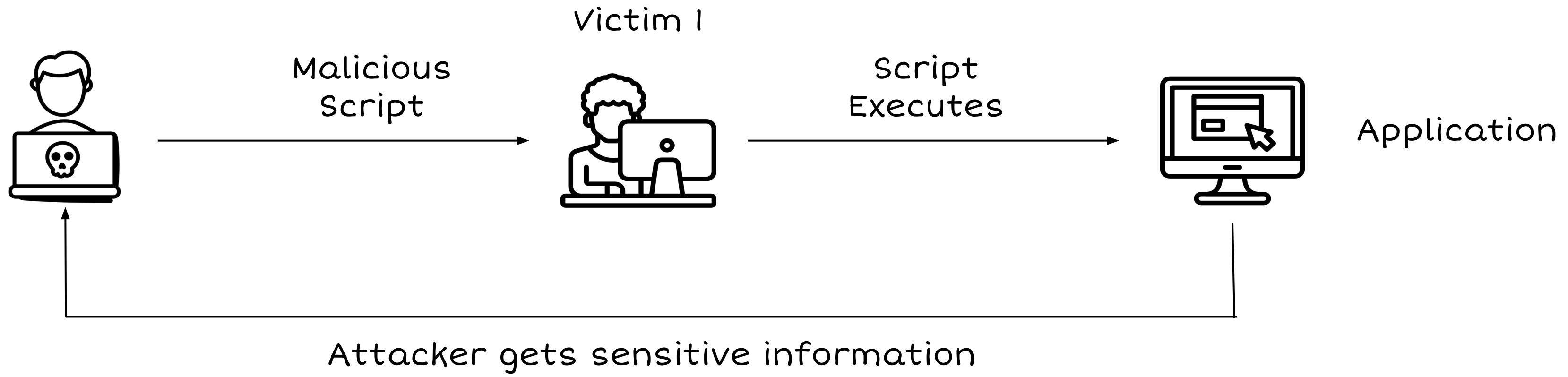
Bug Fix

1. **Synchronization Mechanisms:** Use synchronization mechanisms like **locks** to control access to shared resources or critical sections of code
2. **Atomic Operations**
3. Implement message **queues** to decouple tasks and ensure that operations are processed in a controlled and synchronized manner.
4. **Transactional Databases:** This ensures that either all operations in the transaction succeed or none of them do, maintaining data consistency.

Cross Site Scripting (XSS)

Xss

XSS is a client side attack in which the application executes arbitrary javascript code. When an attacker successfully injects malicious scripts into a web page, the scripts are executed by the victim's browser, which can eventually perform unauthorized actions on behalf of victim, steal session cookies, etc.

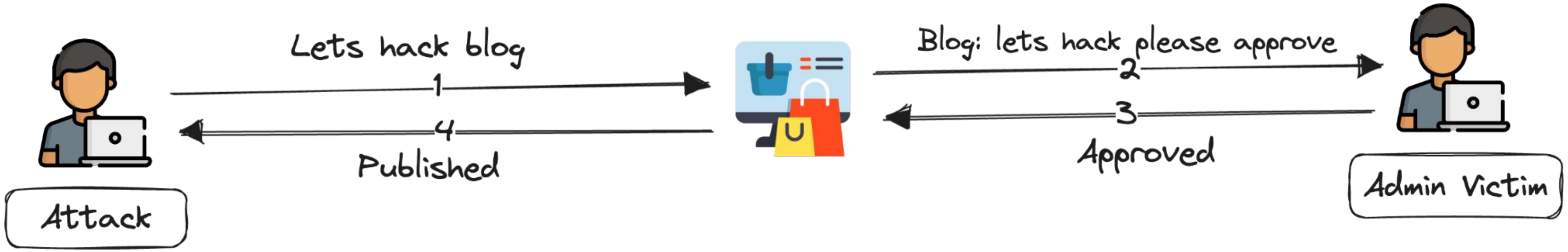


The Bug

Happy flow

The application allows user to write community blogs which when approved the moderator or admin community users can read it.

The Bug


Lets hack blog
1
4
Published
Attack
Blog: lets hack please approve
2
3
Approved
Admin Victim

Bug Fix

1. Encoding the User input
    a. <script>alert(1)</script>
    b. **Example: &lt;script&gt;alert(1);&lt;/script&gt;**

2. Sanitation of user Input

Bug Fix

1. Satanization of user Input

Example:

1. Browsers doing weird stuff
   <O7> test </O7>

2. Lets add attribute foo="bar"
   <O7 foo="bar">test</O7>

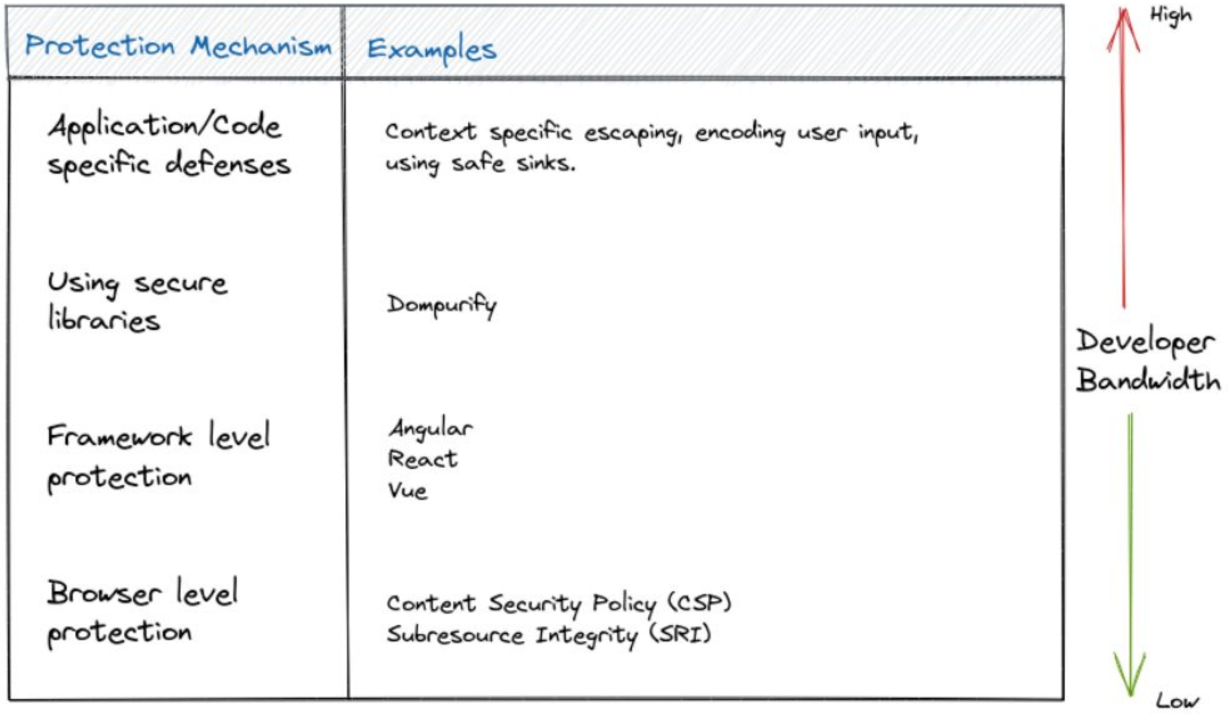
3. what if add <h1> in attribute?
   <O7 foo="bar <h1>">test</O7>

4. **<O7 foo="bar <img/src=1 onerror=alert(1)>">test</O7>**

Note: to see how browser see it user document.body.innerHTML;
console.log(document.body.innerHTML)

Credits: https://www.youtube.com/watch?v=HUtkW2gjC8Q

CLIENT XSS - DEFENCES



Protection Mechanism	Examples
Application/Code specific defenses	Context specific escaping, encoding user input, using safe sinks.
Using secure libraries	Dompurify
Framework level protection	Angular React Vue
Browser level protection	Content Security Policy (CSP) Subresource Integrity (SRI)

High

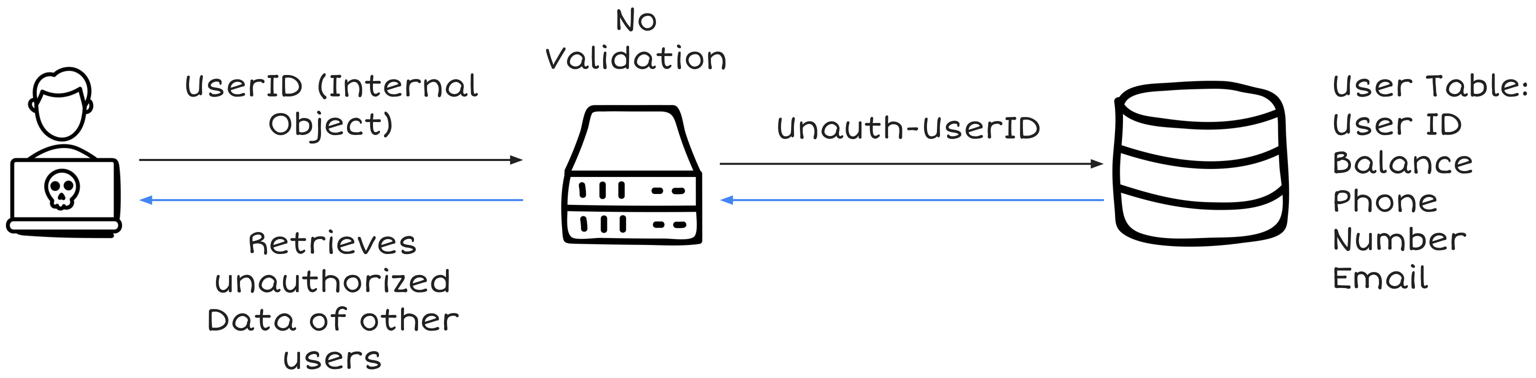
Developer Bandwidth

Low

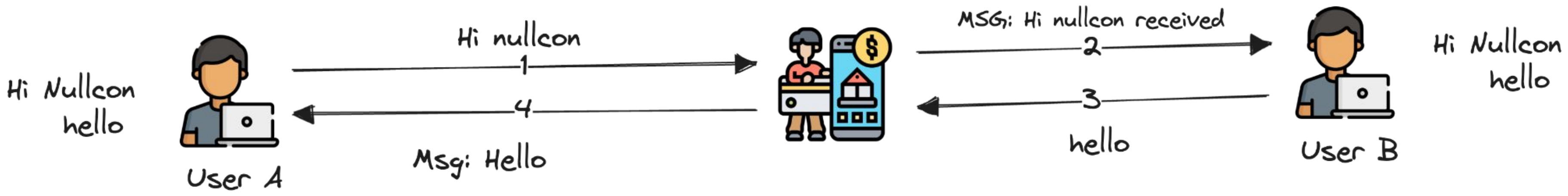
Insecure Direct Object Reference (IDOR)

IDOR

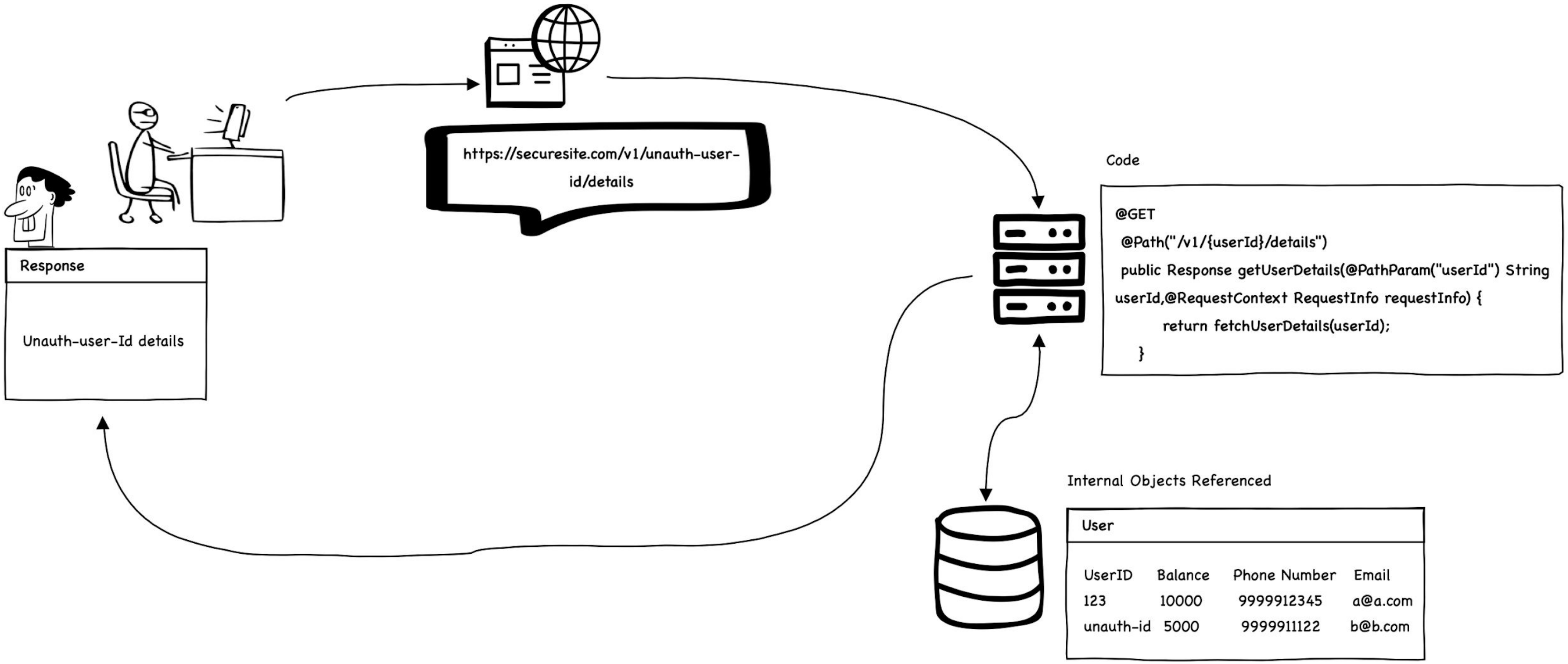
Insecure Direct Object Reference is an access control vulnerability that occurs when an internal object is exposed externally and is user controlled.



The Bug


Hi Nullcon
hello
User A
Hi nullcon
1
4
Msg: Hello
MSG: Hi nullcon received
2
3
hello
User B
Hi Nullcon
hello

Concept


https://securesite.com/v1/unauth-user-id/details
Response
Unauth-user-Id details
Code
Internal Objects Referenced
User

```
@GET
 @Path("/v1/{userId}/details")
 public Response getUserDetails(@PathParam("userId") String
userId,@RequestContext RequestInfo requestInfo) {
        return fetchUserDetails(userId);
    }
```

UserID	Balance	Phone Number	Email
123	10000	9999912345	a@a.com
unauth-id	5000	9999911122	b@b.com

Good Code/Bad code

```
@GET
 @Path("/v1/{userId}/info")
 public Response getUserinfo(@PathParam("userId") String
userId,@RequestContext RequestInfo requestInfo) {
      return fetchUserDetails(userId);
   }
```

✗

```
@GET
@Path("/v1/info")
public Response getUserinfo(@RequestContext RequestInfo
requestInfo) {
      String xAuthId = requestInfo.getAuthId();
      return fetchUserDetails(xAuthId);
   }
```

✓

Remediation

- As a golden rule, do not expose internal objects externally
- If had to be exposed, ensure the values are not guessable and Access Control checks are in place when retrieving data

State management

State management

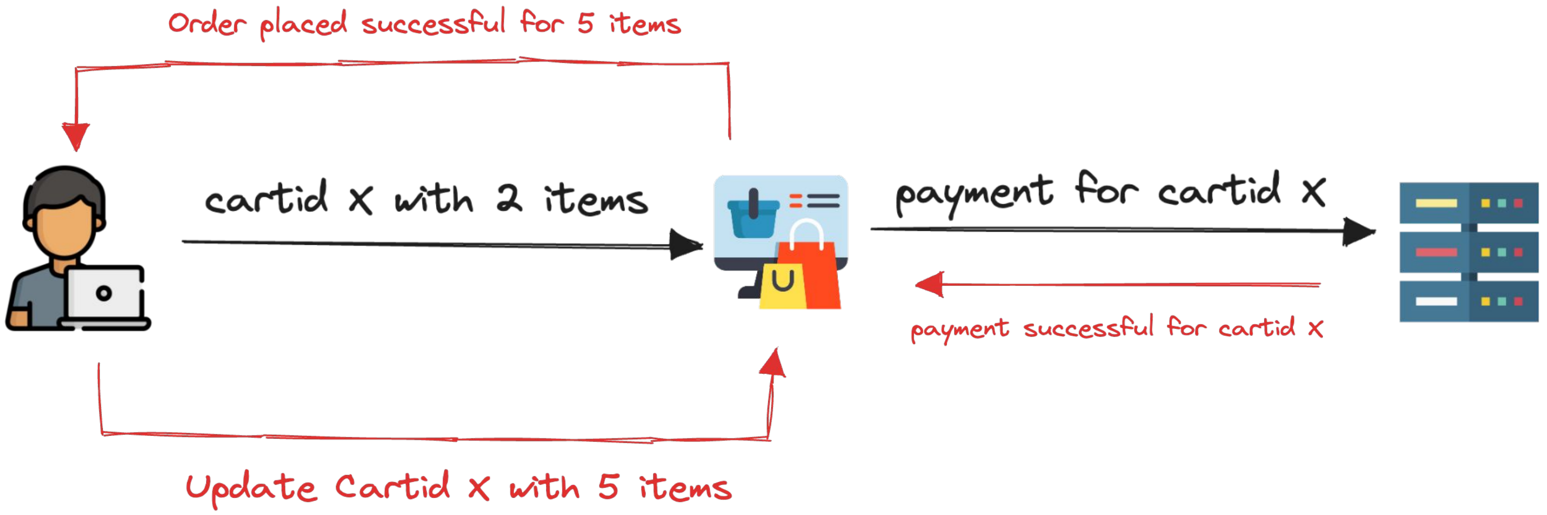
State management refers to the process of managing and maintaining the state or data of an application or system throughout its lifecycle.

Application state management is the process of maintaining knowledge of an application's inputs across multiple related data flows that form a complete business transaction

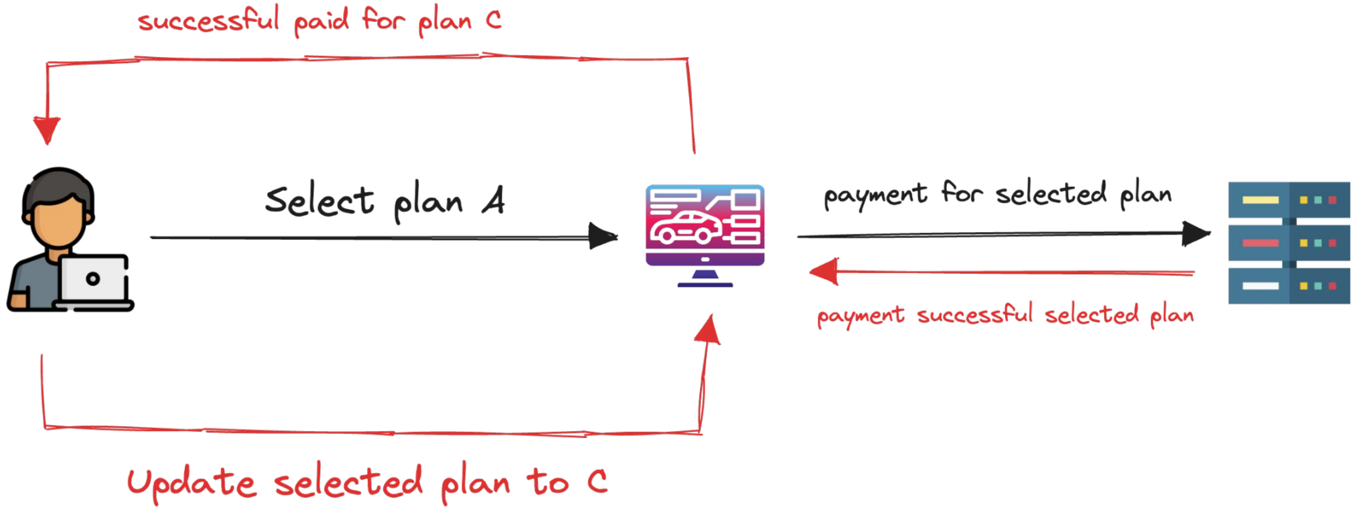
Case Study

1. Updating cart will making payment
2. Updating Insurance plan while making payment

Updating cart will making payment


Order placed successful for 5 items
cartid X with 2 items
payment for cartid X
payment successful for cartid X
Update Cartid X with 5 items

Updating Insurance plan while making payment

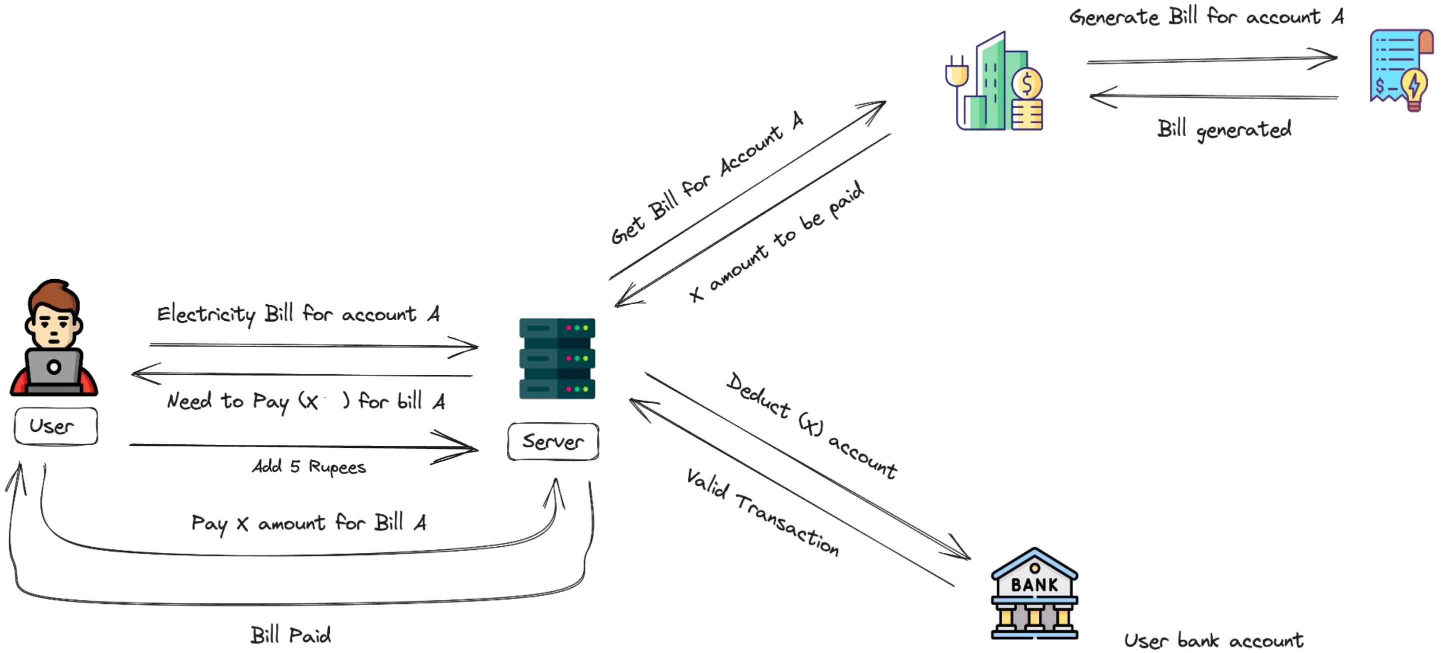

successful paid for plan C
Select plan A
payment for selected plan
payment successful selected plan
Update selected plan to C

WorkFlow Based

The Bug

1. Transaction initiation phase was vulnerable.
2. Attackers gain access to the transaction process.
3. Within the transaction body, they identify the "Transaction charge" parameter.
4. Attackers exploit this access to remove the "Transaction charge" api.
5. The removal effectively circumvents the associated convenience fee during the payment.
6. This manipulation allows attackers to bypass paying the convenience fee, potentially leading to unauthorized cost savings.

The Bug


Generate Bill for account A
Bill generated
Get Bill for Account A
X amount to be paid
Electricity Bill for account A
Need to Pay (X) for bill A
Add 5 Rupees
User
Server
Deduct (X) account
Valid Transaction
Pay X amount for Bill A
Bill Paid
BANK
User bank account

Thank You