# HACKING AUTHENTICATION CHECKS IN WEB APPLICATIONS

ASHISH RAO
&
SIDDHARTH ANBALAHAN

OWASP
The Open Web Application Security Project

# About Ashish

OWASP
The Open Web Application Security Project

- 4 years of IT Security Experience
- Security Consultant and Researcher – Application and Code Security Practice
- Expertise in performing Security Design reviews and Security Code Reviews
- Developed Code Review Checklists and Automation scripts for many platforms
- Conducted Trainings on Secure Development of Web and Mobile applications for different platforms

http://artechtalks.blogspot.in/

# About Siddharth

- 10 years of IT industry experience
- 6 years information security experience
- Senior Security Consultant – Application Security Testing Practice
- Co-Author of the book "Application Security in ISO 27001 Environment".

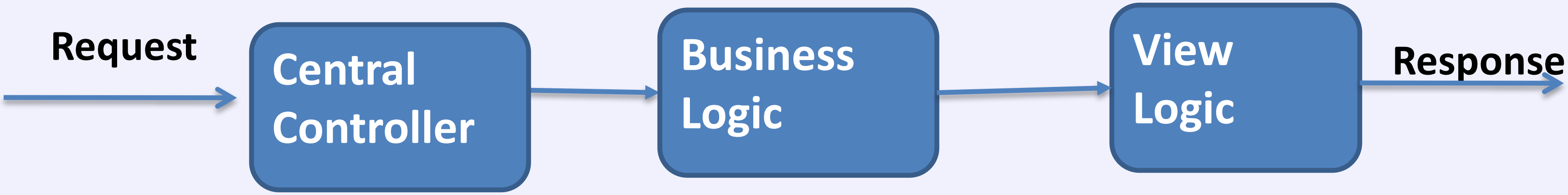
# INDEX

- Hacking Authentication Checks in Web Applications
  - Hacking Application Designs
  - Hacking J2EE Container Managed Authentication
  - Hacking Control Flow in J2EE
  - Hacking Insecure POSTBACK implementation in .NET

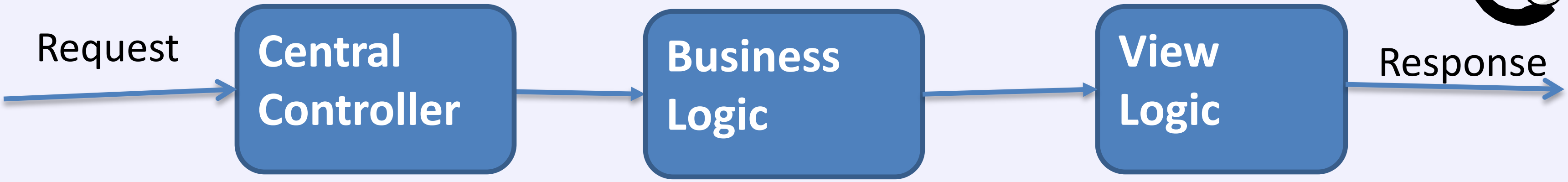
# HACKING APPLICATION DESIGNS

# Hacking Application Designs

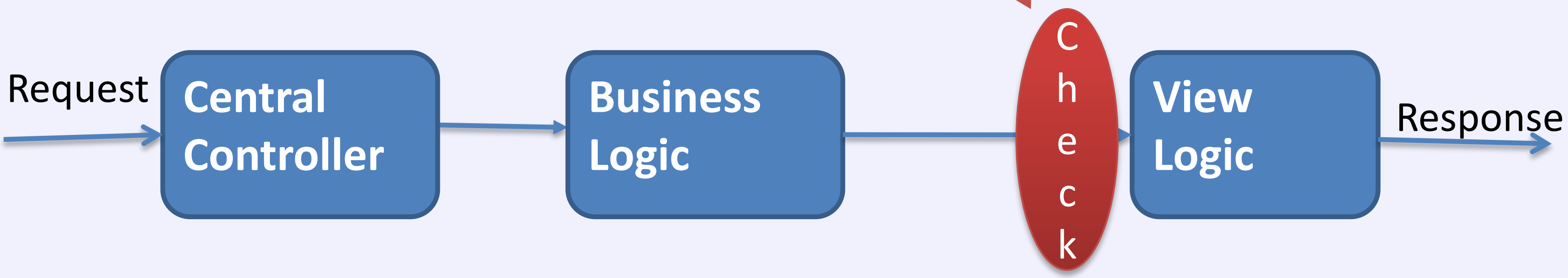
- Applications usually designed using MVC – A model- view - controller technique
- There is logical segregation of code
- Components interact with each other in **sequence**



If we have to implement an authentication check in this sequence where would we place it?



# Most developers place it here

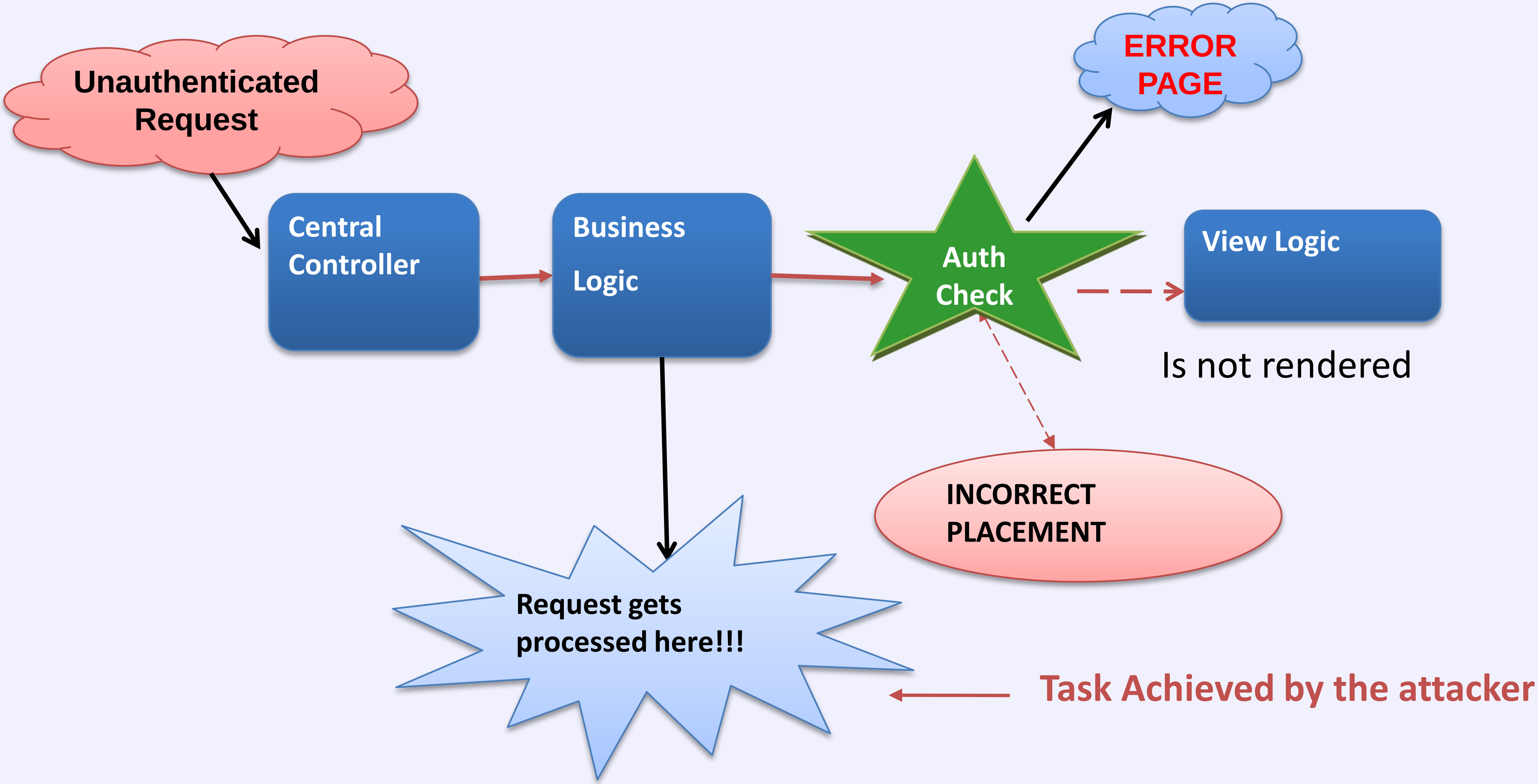


Only for the views

*Assumption: Forms and Pages presented as views in the application will be accessed first. These forms or pages are the only way to send form submissions or internal requests for changing data.*

# Lets see what goes wrong

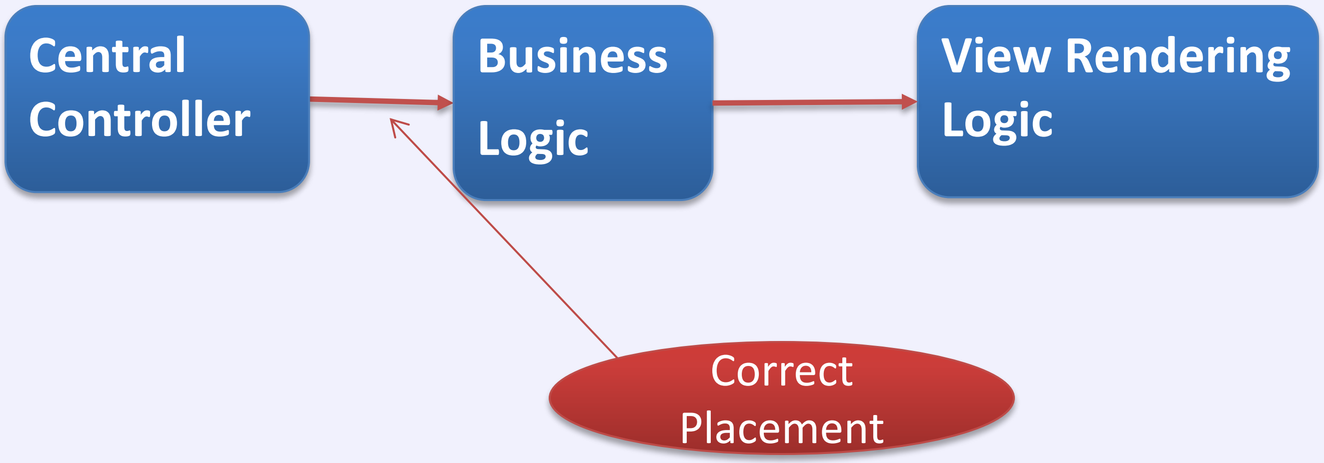
OWASP
The Open Web Application Security Project



# DEMO –

## Unauthorized access due to incorrect placement of checks

- **Security Measures:**
  - Place all validation checks before request processing logic



# HACKING J2EE CONTAINER MANAGED AUTHENTICATION

# Container Managed Authentication Flaw

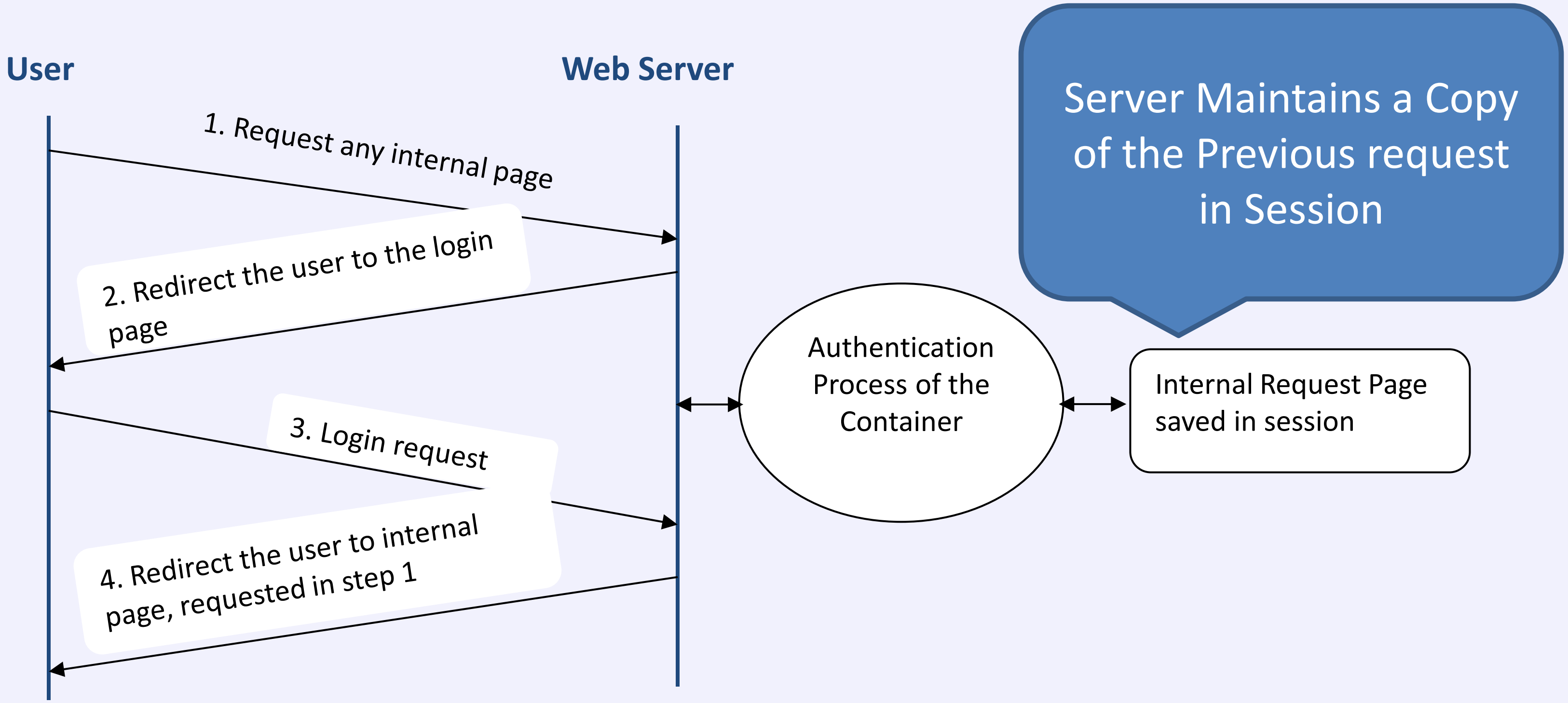
- J2EE Contained Managed Authentication
  - Configuration of “security-constraints” in deployment descriptor – web.xml
  - Responsible for blocking unauthenticated access to internal resources

× An attack similar to CSRF can be performed by a remote hacker to bypass authentication

# Container Managed Authentication Flow

OWASP
The Open Web Application Security Project

- Typical Flow of Container Managed Authentication


User
Web Server
1. Request any internal page
2. Redirect the user to the login page
3. Login request
4. Redirect the user to internal page, requested in step 1
Authentication Process of the Container
Internal Request Page saved in session
Server Maintains a Copy of the Previous request in Session

# What is the flaw?

OWASP
The Open Web Application Security Project

- **The Catch?**
  - ▪ Server Maintains a Copy of the Previous request in Session
  - ▪ Once the user is authenticated the server process the request previously stored

× An **attacker** can send any POST request for an internal action via a CSRF technique, when an unsuspecting victim logs in, the internal action will get processed.

# DEMO –

## Hacking J2EE Container Managed Authentication

- **Similarity with CSRF:** The only requirement is that the user must login after the malicious unintended request has been sent to the server using the same browser

- **Local Attack**: A local attacker may forge a request from victim’s browser, and keep the login page open. When the victim logs in, the malicious request will get executed

- **Security Measures:**
  - All requests that result in change of data, transactions, should be accompanied with a token
    - Token should unique for each user session
    - Token should be random making it difficult to guess
    - Should be always validated at the server

# HACKING CONTROL FLOW IN J2EE

# Control Flow Flaws

Does this code look ***safe*** to you?

```
String username = session.getAttribute(“user”);
if (username == null)
{
        response.sendRedirect(“Access Denied Page”);
}
….
Business Logic Processing
```

# Control Flow Flaws

- But what is wrong in it?

```
String username = session.getAttribute(“user”);
if (username == null)
{
        response.sendRedirect(“Unauthorized Page”);
}
….
Business Logic Processing
```

# Control Flow Flaws

- I am checking for an authenticated session
  And I am then redirecting
  unauthenticated user

```
String username = session.getAttribute(“user”);
if (username == null)
{
        response.sendRedirect(“Access Denied Page”);
}
….
Business Logic Processing
```

# Control Flow Flaws

Have you ever wondered, what if the execution do not stop **here**?

```
String username = session.getAttribute(“user”);
if (username == null)
{
        response.sendRedirect(“Access Denied Page”);
}
….
Business Logic Processing
```

# Control Flow Flaws

Business logic would get executed even for unauthenticated request.

```
String username = session.getAttribute(“user”);
if (username == null)
{
        response.sendRedirect(“Access Denied Page”);
}
….
Business Logic Processing
```

In reality this is **not protected**

# Control Flow Flaws

× The execution flow does not stop after the *response.sendRedirect* call

× Entire page is processed and then the user is redirected to error page

× Thus, the business logic remains unprotected

# DEMO –

## Unauthorized access due to control flow flaw

- **Security Measures:**
  - Terminate the execution flow after redirection call.

```
String username = session.getAttribute(“user”);
if (username == null)
{
        response.sendRedirect(“Access Denied Page”);
        return;
}
….
Business Logic Processing
```

# HACKING INSECURE POSTBACK AUTHENTICATION IN .NET

# INSECURE POST-BACK

- **POSTBACK**
  - POSTBACK in ASP.NET is an event that occurs whenever an action is performed by a control in the ASP.NET page
- *ISPOSTBACK*?
  - Property of ASP.NET Page that allows developers to check if page is “called” or “refreshed” as a result of a control event OR called for the first time

# INSECURE POST-BACK

- Mixing authentication check and *ISPOSTBACK*

```
protected void Page_Load(object sender, EventArgs e)
{
        If (!IsPostBack)
        {
                lblTitle.text = “Create Employee”
s
                If (!Request.IsAuthenticated)
                {
                        Response.Redirect(“~/Error.aspx”);
                }


        }
}
```

# INSECURE POST-BACK

- **Assumption**: A form will be accessed by an authenticated user first time only by clicking on a link that displays the form

× An **attacker** can send a POST request for creating a new employee.

× The *IsPostBack* condition will fail and therefore will not invoke the authentication check

# DEMO –

## Hacking insecure POSTBACK based authentication check in ASP.NET

# INSECURE POST-BACK

- **Security Measures:**
  - IsPostBack property check should be independent of the authentication check.

```
protected void Page_Load(object sender, EventArgs e)
{
        If (!IsPostBack)
        {
                lblTitle.text = "Create Employee"
        }

If (!Request.IsAuthenticated) {
                        Response.Redirect("~/Error.aspx"); }
```

- **Closing Notes**
  - Authentication flaws can be avoided by placing careful consideration to design and the way applications behave
  - Is the placement of authentication check correct?
  - Is it secure your processing logic?
  - Is there a control flow behavior that you need to test?

# Questions

# Blogs and Reference

- http://artechtalks.blogspot.in/2013/02/j2ee-container-managed-authentication.html
- http://artechtalks.blogspot.in/2013/02/insecure-postback-based-authentication.html
- http://packetstormsecurity.com/files/119129/Insecure-Authentication-Control-In-J2EE.html

*-- Watch out this space for more blogs*

Thank You
&
Share your feedback with us.

rao.ashish20@gmail.com
AND
sidhanbu@gmail.com