Secure Coding v2.0

Agenda

- Standard Guidelines to protect backend APIs
- Parameter Tampering Attacks on APIs (amount tampering)
- HTTP Parameter Pollution Attack
- Session Management Issues
- Security Flaws in implementing protocols like Oauth 2.0
- Client Side Injection : XSS & CSRF attacks

Standard Guidelines to protect APIs (Authentication / Oauth)

Authentication

- Don't use `Basic Auth` Use standard authentication (e.g. JWT, OAuth).
- Don't reinvent the wheel in `Authentication`, `token generation`, `password. Use` standards.
- Use `Max Retry` and jail features in Login.
- Use encryption on all sensitive data.

OAuth

- Always validate `redirect_uri` server-side to allow only whitelisted URLs.
- Always try to exchange for code and not tokens (don't allow `response_type=token`).
- Use `state` parameter with a random hash to prevent CSRF on the OAuth authentication process.
- Define the default scope, and validate scope parameters for each application.

OAuth 2FA OTP Bypass

For main paytm application, the requests to endpoint oauth2/authorize contains client_id value as "market-app"

For GoldenGate application , the requests to endpoint oauth2/authorize contains client_id value as "merchant-golden-gate-app"

From Paytm Web , the requests to the same endpoint contains client_is value as "paytm-web"

We came to following conclusions after observing this endpoint:

1. Response to oauth2/authorize with client_id as paytm-web , market-app is HTTP 200 OK and asks for OTP sent to mobile Number. Subsequently , the api validate/otp receives a response HTTP 303 SEE OTHER Redirect containing the "code" in it's Location Header which when finally hits oauth with a parameter client_secret gives us an access_token (sso Token) in response for the user.

1. Response to oauth2/authorize with client_id as "merchant-golden-gate-app" , gives us directly a "code" in the response.

We use this code with a client_secret to oauth/authozie to get an access_token (sso Token) without requirement of an OTP for the user potentially bypassing of the current OTP Validation system at OAuth.

This is a problem for OAuth because :

1. It allows to fully bypass current OTP Protection for login for All users at our Platform.
2. Only thing an attacker needs to change at OAuth request is the parameter client_id to Bypass our current OTP Protection , which makes exploitability of this attack easy.

HTTP Parameter Pollution (HPP)

- Supplying multiple parameters in GET/POST Requests with the same name
- Bypass input validation based on how application handles multiple parameters

POST /oauth2/seller/forgotpass HTTP/1.1
Host: persona.paytm.com
X-forwarded-host: test.com
User-Agent: Mozilla
Referrer: https://persona.paytm.com/oauth2/authorize?client_id=
Cookie: _vwo_uuid_v2= ; tvc_vid= ; _gid=;
email=seller@paytm.com&email=attacker@attacker.com&client_id=seller-web-login

HTTP/1.1 200 OK
{status:"SUCCESS"}

- Sends forgot Password reset Email to both email IDs : seller@paytm.com as well as attacker@attacker.com
- Leads to a Full Account compromise without any user interaction

Critical Insecure Direct Object Reference - Account Takeover
pguat.paytm.com

- Able to claim and Login into any user's account
- Highly severe , can lead to compromise of large number of user's account if exploited in the wild

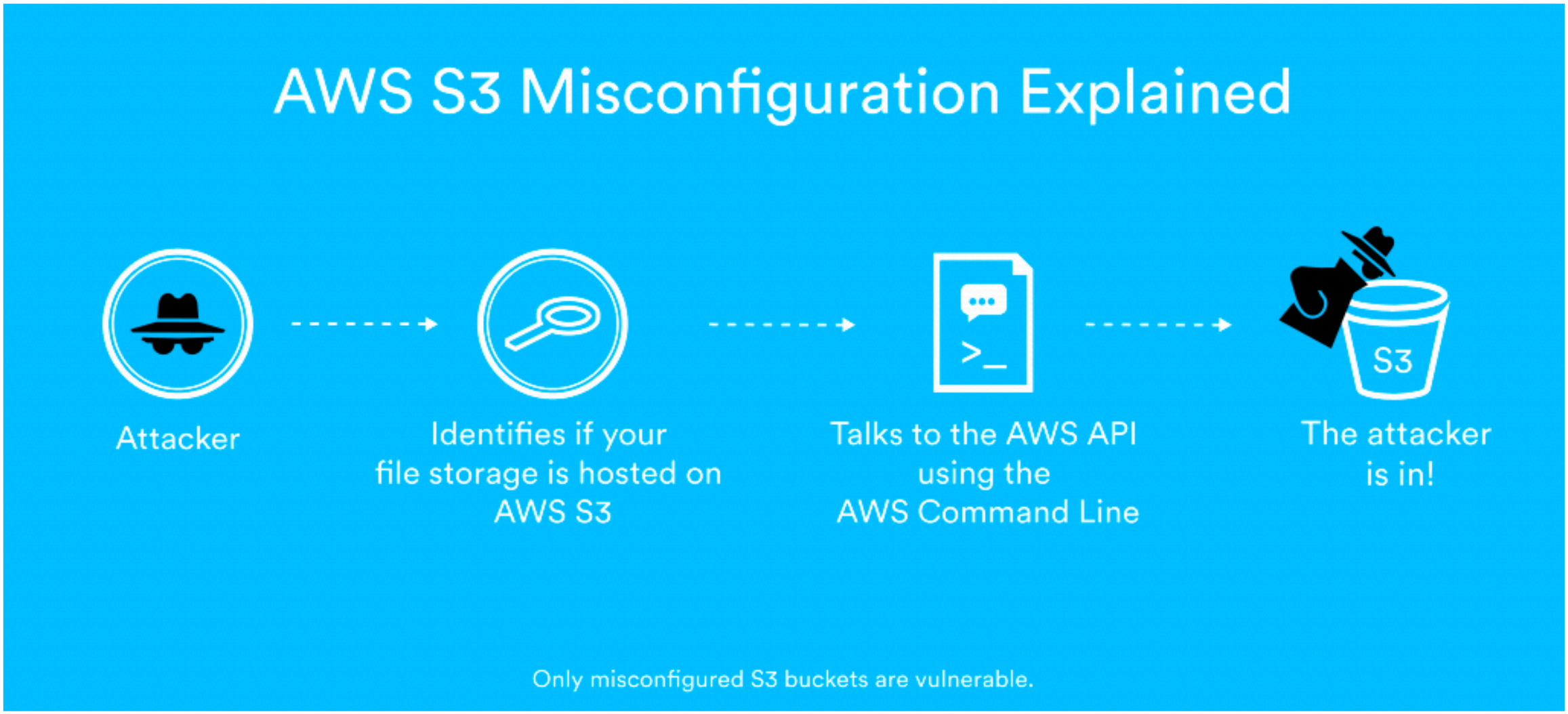
```
POST /PayTMSecured/app/sales/subUserSaveEdit HTTP/1.1
Host: pguat.paytm.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://pguat.paytm.com/PayTMSecured/app/sales/subUsers
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
X-CSRF-TOKEN: b6350b58-3477-4a13-b050-0dcc1f71fcc5
Content-Length: 139
Cookie: []

fName=ashutosh&lName=kumar&contact=9717576102&email=kamarashutosh1@gmail.com&role=REPRESENTATIVE&status=REPRESENTATIVEACTIVE&userId=8945361
```

- Modifying the value of POST Request parameter userID=“”
- Attacker able to perform the POST Request action for different values of userID's
- No check on backend to validate if the userID belongs to the Logged in user
- Leading to a Full Account Compromise

AWS S3 Bucket Misconfigurations

Paytm



AWS S3 Bucket Misconfigurations

Types of Security Misconfiguration at S3 Bucket permissions :

Amazon S3 bucket allows for full anonymous access

Amazon S3 bucket allows for arbitrary file listing

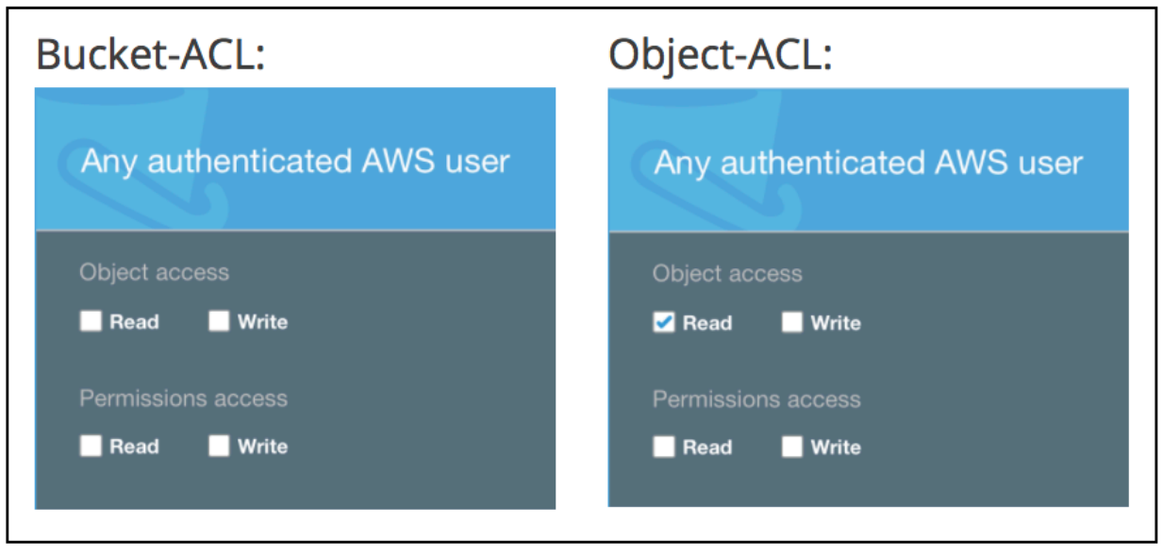
Amazon S3 bucket allows for arbitrary file upload and exposure

Amazon S3 bucket allows for blind uploads

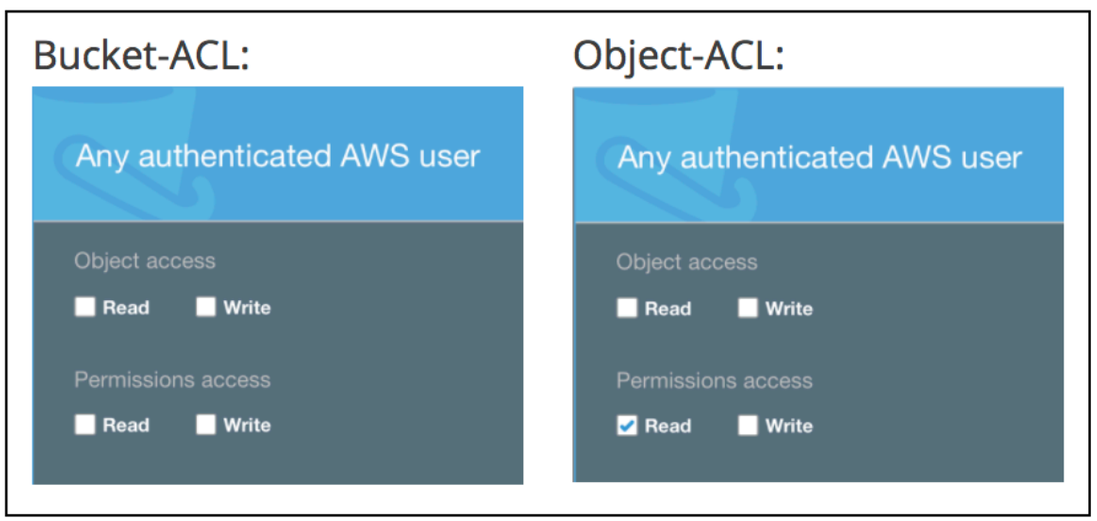
Amazon S3 bucket allows arbitrary read/writes of objects

Amazon S3 bucket reveals ACP/ACL

With the following ACL setup inside AWS S3:



AWS S3 Bucket Misconfigurations




```
$ aws s3api get-object-acl --bucket test-bucket --key read-acp.txt
{
    "Owner": {
        "DisplayName": "fransrosen",
        ...
```

This means `READ_ACP` can be different for each object, independently of the settings on the bucket.

AWS S3 Bucket Misconfigurations