

## Part I

### Setting Up Paypal Account

The process involved in setting up a paypal are described as follows

- Creating an account on Paypal's website with personal information such as email address and password.
- After creating an account and verifying email address, the user has the flexibility to switch to a login to paypal developer account and can switch to the sandbox environment mode from there.
- By default, there is an application created on paypal with its credentials i.e. client id and secret. There is also the flexibility of creating a new app with a different credential.
- Also by default, there are two sandbox testing accounts (personal and business) created to pay and receive dummy payments from your application.
- Paypal payment was therefore integrated into the UoH catering system software using Paypal Javascript SDK to display paypal payment buttons in combination with Paypal REST APIs on the local server.

The credentials testing process involves generating credentials and these credentials (client id and secret key) are verified to ensure their validity by checking the format of the credentials and also ensuring the user possesses appropriate permissions. After the credentials have been verified, the user can make test payments using the paypal sandbox environment to ensure the integration is working perfectly.

After integration and testing of the payment services, the developer can deploy the software to production for Live transactions. In this case, the sandbox account credentials are swapped to Live credentials.

Paypal also continues to monitor API credentials to ensure the validity and its correct usage by analyzing transaction information.

In this project, there are three(3) paypal REST API endpoints that were consumed using C#/.NET and they are as follows:

- Authentication (<https://api-m.sandbox.paypal.com/v1/oauth2/token>): This endpoint was consumed using .NET HttpClient. Which was set up to use the Basic authentication scheme. In this instance, the Client Id and Secret Key were used to set up the authorization request header. These credentials (Client Id and Secret Key) are therefore encoded into a base64-encoded string. The endpoint also takes a query parameter "grant\_type=client\_credentials" as "x-www-form-urlencoded".

```

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(
    "Basic", Convert.ToBase64String(
        System.Text.ASCIIEncoding.ASCII.GetBytes(
            $"{PayPalClientId}:{PayPalSecretKey}")));

var dict = new Dictionary<string, string>();
dict.Add("Content-Type", "application/x-www-form-urlencoded");

var req = new HttpRequestMessage(HttpMethod.Post, url + "?grant_type=client_credentials")
{
    Content = new FormUrlEncodedContent(dict)
};
HttpResponseMessage resp = await client.SendAsync(req);

client.Dispose();

```

The format of response from endpoint request for access token for authorization is json as shown below

#### ▲ [JSON]

```

scope: "https://uri.paypal.com/services/invoicing https://uri.paypal.com/services/vault/payment-tokens/read https://uri.paypal.com/service:
access_token: "A21AAJCiCURm-12cQlAtffzByPiESsvtZc-2fcB8K8u8pmZe1pIXMr6obq4lq_cwVcgze4kbPBSC79E8cagDI3LVHR0sWWv_A"
token_type: "Bearer"
app_id: "APP-80W284485P519543T"
expires_in: 32399
nonce: "2023-05-09T18:15:49Z8fGLsmOCVfBdM0HQ6RBUhTZ_mOEhHnWfQPYVKhbgt04"

```

- Create order (<https://api-m.sandbox.paypal.com/v2/checkout/orders>) : This endpoint is responsible for creating order (payment) for a specific order from the catering services application when an order is placed. The http request header for the create order endpoint takes an access\_token as the authorization header, content-type of “application/json” and the request body of this endpoint is of JSON format type.

The request body for the API endpoint is of type json which is key/value pair. It takes the “intent” of the transaction, “purchase\_units” which consists of the “currency\_code” and “value” i.e. the amount. As shown below.

```

PayPalOrder order = new PayPalOrder
{
    intent = "CAPTURE",
    invoice_id = invoice_id,
    reference_id = invoice_id,
    purchase_units = new[] {
        new PurchaseUnit
        {
            amount = new Amount
            {
                currency_code = "GBP",
                value = amount,

                breakdown = new Breakdown()
                {
                    item_total = new ItemTotal()
                    {
                        currency_code = "GBP",
                        value = amount
                    }
                }
            },
            items = list
        }
    }
};

```

The screenshot below shows the request header information and how the http request is sent to the create order endpoint.

```

using(var _client = new HttpClient())
{
    TempData["access_token"] = authResponse.access_token;
    _client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", authResponse.access_token);
    var requestHeader = new Dictionary<string, string>();
    requestHeader.Add("Content-Type", "application/json");

    StringContent content = new StringContent(JsonConvert.SerializeObject(order), Encoding.UTF8, "application/json");
    content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

    //call checkout order endpoint
    var oderEndpoint = baseUrl + checkoutOrderEndpoint;
    HttpResponseMessage response = await _client.PostAsync(oderEndpoint, content);

    if (response.IsSuccessStatusCode) {...}
}

```

The response from the capture order endpoint is also in the format of json and the endpoint also returns the HTTP status code 201 as the order id created. Attached below is the response from that endpoint.

```
└─ [JSON]
  id: "01L72968EP289290P"
  status: "CREATED"
  └─ links
    └─ [0]
      href: "https://api.sandbox.paypal.com/v2/checkout/orders/01L72968EP289290P"
      rel: "self"
      method: "GET"
    └─ [1]
      href: "https://www.sandbox.paypal.com/checkoutnow?token=01L72968EP289290P"
      rel: "approve"
      method: "GET"
    └─ [2]
    └─ [3]
      href: "https://api.sandbox.paypal.com/v2/checkout/orders/01L72968EP289290P/capture"
      rel: "capture"
      method: "POST"
```

Capture Order (<https://api-m.sandbox.paypal.com/v2/checkout/orders/{id}/capture>) : This endpoint captures the order and it takes the order id as the path parameter and also takes “access\_token” or paypal Client Id and Secret Key as the authorization on the request header.

```
using (var _client = new HttpClient())
{
    string captureOrder = _config.GetValue<string>("ApiRoute:CaptureOrder");

    // _client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
    // _client.DefaultRequestHeaders.Authorization = AuthenticationHeaderValue.Parse($"Bearer {token}");
    _client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(
        "Basic", Convert.ToBase64String(
            System.Text.ASCIIEncoding.ASCII.GetBytes(
                $"{PayPalClientId}:{PayPalSecretKey}")));

    StringContent content = new StringContent("", Encoding.UTF8, "application/json");
    HttpResponseMessage response = await _client.PostAsync($"{baseUrl}{captureOrder}/{orderId}/capture", content);
    if (response.IsSuccessStatusCode) ...
    else ...
}
```

The response format from the endpoint is of type json and it returns the HTTP status code 201. The returned response includes the order id, transaction status and the payer details.

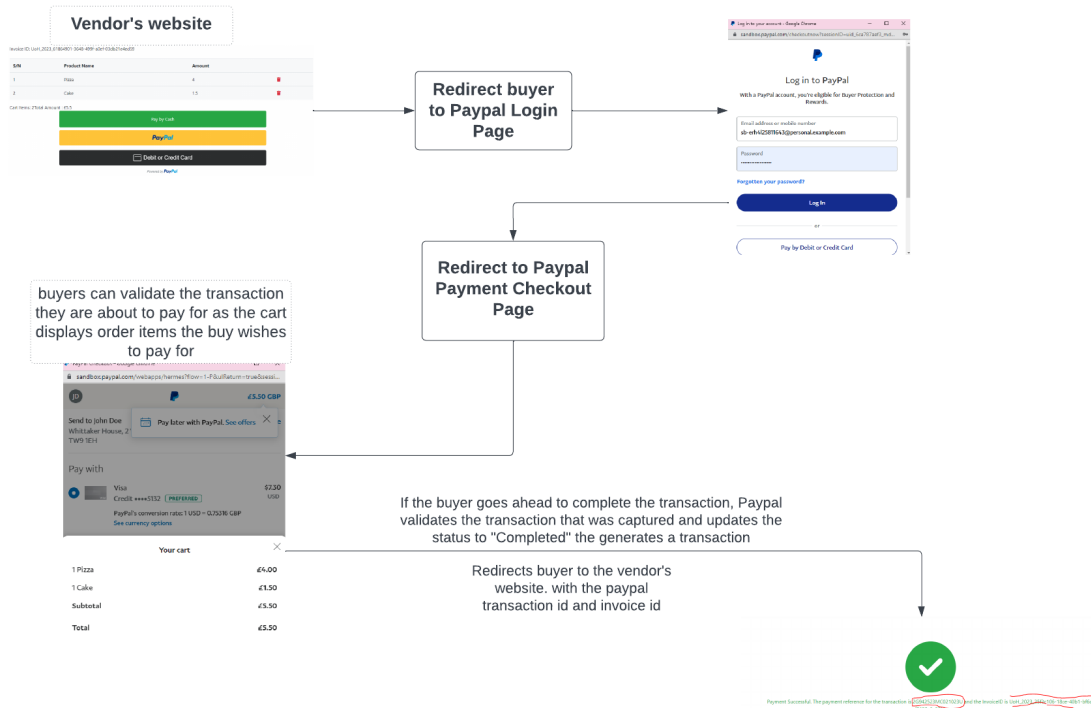
```
└─ [JSON]
  id: "39P02367B9965090A"
  status: "COMPLETED"
  └─ payment_source
    └─ paypal
      email_address: "sb-erh4l25811643@personal.example.com"
      account_id: "7WT6P2S5VFM5Q"
      └─ name
        given_name: "John"
        surname: "Doe"
      └─ address
        country_code: "GB"
    └─ purchase_units
    └─ payer
    └─ links
```

## Paypal Protocol in processing payment

In the context of the built artifact, There is a 3-layered process between the vendor, buyer and paypal and this protocol involves the exchange of data between them. The initial protocol is the protocol between the buyer and the vendor and this involves the buyer sending a request to the vendor's website server to purchase an item and the vendor's website displays a payment button which redirects the buyer to the third-party payment handler integrated (in this case, Paypal) for the buyer to make payment.

The subsequent protocol is between the vendor and Paypal whereby the vendor sends a secure HTTPS request to paypal to initiate the payment and this request body includes information about the transaction such as amount, currency code e.t.c. This payment is processed and the status of the payment is sent to the client or server of the vendor's application.

The other protocol between the buyer and paypal involves the point buyer being redirected to the paypal payment page to enter payment information or card details to complete the payment and the payment status is returned to the vendor's website.



The Paypal Event processes between the buyer, vendor and paypal is explained in the diagram above.

## Weakness of the built paypal integration system

Considering the artifact implemented integrating paypal third-party payment gateway and exchange protocols described above, there are some weaknesses identified in the integration and these include;

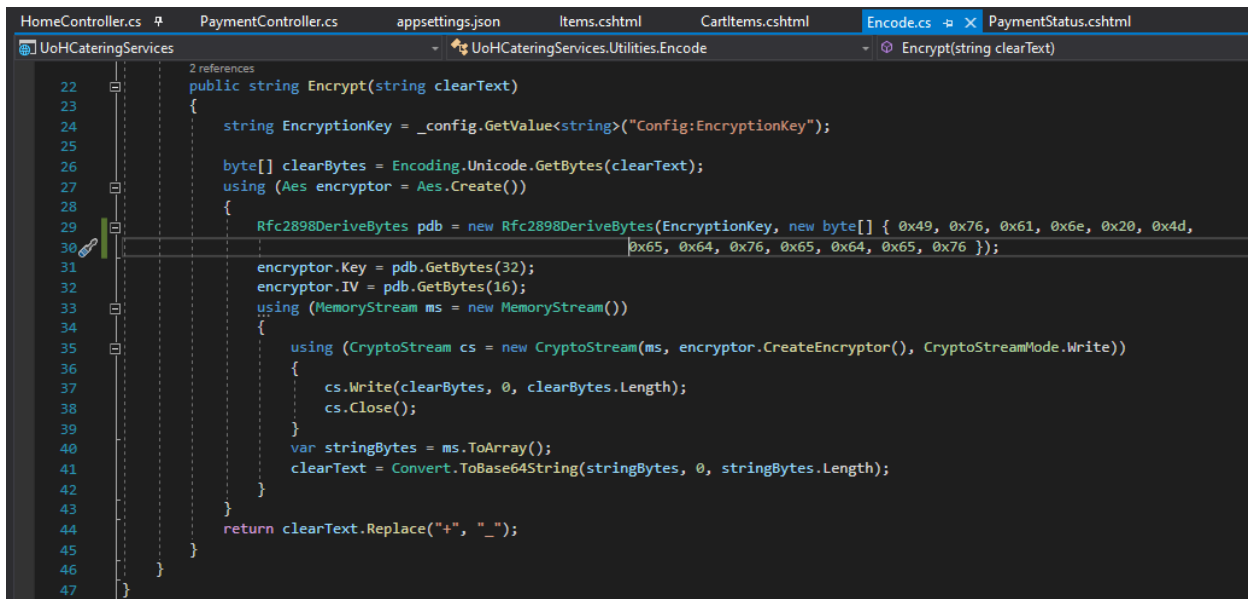
- Exchange of plain text as payload over a network call such as HTTP request. This payload data can be intercepted and manipulated by hackers during a network call request.
- Another weakness identified is the exposure of Client Id on the client-side of the application to render the paypal button used by Javascript SDK. This could be a lead for hackers to use the client id for malicious intent.
- Another key weakness pertaining to the flow of payment gateway implementation is the lack of tracking of failed payment/transactions. Integration of Paypal API involves sending requests and receiving responses over the network, and failure or downtime can happen during this network call, so this implementation lacks the flow to cater for transactions that were successful but payers/customers didn't get

value for. This implementation to track failed payments is called webhook. This current artifact doesn't include this.

- Sending request parameters as query parameters in plain text without encryption is another weakness in the system as this can be observed in the "Generate Access Token" endpoint. This endpoint takes the query parameter "grant\_type=client\_credentials" in its request.
- This can also be further observed on the API endpoint that captures order, this endpoint takes the order id as request path parameter on the URL. The order Id in this case serves as a payment reference and its being exposed on the request path

## Part II

Authentication Encryption: This can be described as a cryptographic technique that consists of encryption and authentication while promoting Confidentiality, Integrity and Authenticity (C.I.A) of exchanged data. This concept ensures that API resources/endpoints are protected from requests that are unauthorized. AES encryption algorithm can be used to achieve this. As shown below is the symmetric encryption algorithm for encrypting password during registration before it is saved to the database.



```
22 public string Encrypt(string clearText)
23 {
24     string EncryptionKey = _config.GetValue<string>("Config:EncryptionKey");
25
26     byte[] clearBytes = Encoding.Unicode.GetBytes(clearText);
27     using (Aes encryptor = Aes.Create())
28     {
29         Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, new byte[] { 0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d,
30                                     0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });
31         encryptor.Key = pdb.GetBytes(32);
32         encryptor.IV = pdb.GetBytes(16);
33         using (MemoryStream ms = new MemoryStream())
34         {
35             using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateEncryptor(), CryptoStreamMode.Write))
36             {
37                 cs.Write(clearBytes, 0, clearBytes.Length);
38                 cs.Close();
39             }
40             var stringBytes = ms.ToArray();
41             clearText = Convert.ToBase64String(stringBytes, 0, stringBytes.Length);
42         }
43     }
44     return clearText.Replace("+", "_");
45 }
46
47 }
```

When a registered user wants to log in, the provided password is encrypted and this encrypted text is compared to what is saved on the database before authenticating the user.

Generic Composition Methods of Authenticated Encryptions (AE) can be referred to as a combination of encryption and message authentication code (MAC) which is a value obtained by applying cryptographic hash function on a secret key to enhance confidentiality, integrity, and authenticity of data access. There are about three different methods of combining encryption and MAC and these include Encrypt-and-MAC, MAC-then-Encrypt, and Encrypt-then-MAC.

- Encrypt-and-MAC: This technique involves encrypting the plain text with an encryption algorithm and the MAC is generated from the encrypted data. In this type of technique, if a request is sent to a resource, the resource which receives the request first validates the message authentication code (MAC) before decrypting the received data.
- Mac-then-Encrypt: This type of technique involves the plain text being hashed with HMAC which is a MAC algorithm. The resulting MAC is then encrypted with an AES symmetric encryption algorithm alongside the data which are both sent as a request.
- Encrypt-then-MAC: This type of technique, the plain data is encrypted first with any symmetric encryption algorithm then the MAC is computed from the encrypted message. The MAC is then encrypted with a symmetric encryption algorithm and this encrypted message and the MAC that was encrypted are therefore sent to the endpoint. The endpoint decrypts the MAC first then validates it by using the decrypted message for the decryption if valid.

Single-Pass Authenticated Encryption is best described as an efficient and lightweight authentication encryption type that encrypt and authenticate data in a single pass i.e. data are only processed once which results in cipher text and authentication tag generated at the same time. Due to the lightweight feature of single-pass AE, it makes it suitable for softwares where fast authentication and encryption is needed.