KNIGHT INK

DANCING WITH THE DEVIL:
AN API HACKER'S SHOWDOWN WITH TRACEABLE'S API SECURITY SOLUTION

I have gained unauthorized access to over a million patient records, taken remote control of law enforcement vehicles, transferred money in and out of bank accounts that didn't belong to me, and hacked cryptocurrency exchanges through their APIs. Now, I face my greatest challenge yet. Traceable.

SUMMARY

This white paper documents my targeting and exploitation of an API protected behind Traceable and whether it was effective in detecting and stopping my API attacks in an attempt to get at the data.

AUTHOR INFORMATION

Alissa Valentina Knight
Partner
Knight Ink
+1 702 766 6362
1980 Festival Plaza Drive
Suite 300
Las Vegas, NV 89135
ak@knightinkmedia.com

TRACEABLE.

PUBLICATION INFORMATION

This white paper is sponsored by Traceable.

Initial Date of Publication:
January 2023
Revision: 1.0

TABLE OF CONTENTS

KEY TAKEAWAYS

- APIs are the pipes and infrastructure that carry valuable data of our digital age and they must be secured to keep the data safe.

- Without proper authentication, authorization, and encryption APIs can become a huge security liability.

- The Open Web Application Security Project (OWASP), a respected security organization, validated the uniqueness of securing APIs by creating a top 10 list of API Security threats, separate from their top 10 list focused on web threats.

- To be effective, API security solutions must understand

- Application business logic - what's a legitimate versus illegitimate request (context) who is requesting it (authentication) if they have permission to request it (authorization) if the requestor is a human or a bad bot (network telemetry).

- Traceable and its unique approach to API security proved very capable of fending off my attacks for both the threats listed in the OWASP API top 10 list AND the OWASP (web) Top 10 list.

INTRODUCTION

INTRODUCTION

WHY I WROTE THIS PAPER

APIs demand a different approach to securing our applications than traditional web application techniques, such as WAFs. These new requirements have resulted in API security "solutions" appearing rapidly and everywhere, making claims about their ability to protect your APIs. While sales and marketing literature can easily make claims about abilities, we all know that how a product actually performs in production can vary greatly from the literature. Thus the impetus for this research paper.

I have been hacking APIs over the last decade, targeting APIs in healthcare that led to unauthorized access to over a million doctor and patient records via mHealth (mobile health) and FHIR (Fast Healthcare Interoperability Resources) APIs; taking remote control of law enforcement vehicles through the automaker's APIs; and transferring money in and out of bank accounts using financial services and FinTech APIs, including cryptocurrency exchanges.

Thus, I'm well qualified to test the efficacy of API security solution claims. So I decided I'd go head-to-head with an API security solution, and share my findings. But first, let's cover some background and context.

APIs IN MODERN SOFTWARE APPLICATIONS

Today, monolithic applications running on single web servers have all but been replaced by microservices in container orchestrators, such as AWS Elastic Container Service (ECS), Fargate, and Kubernetes just to name a few. Brokering access to these containers from the outside and also container-to-container traffic are application programming interfaces (APIs). APIs are the glue that enables application-to-application traffic. API consumers (or API clients) can be mobile apps to even passenger transport vehicles, such as planes, trains, and automobiles. APIs can be thought of as a Rosetta Stone between different applications allowing them to talk.

History of APIs The history of web APIs goes back over two decades to the late 90s along with the release of the first web API (XML) by SalesForce at the IDG Demo conference on February 07, 2000.

Despite the history of paper-based APIs dating back to the 1940s-1970s, the first web API was launched at the IDG Demo conference in 2000 by SalesForce which used Extensible Markup Language (XML) for messaging patterns in information sharing. Today, there are different types of APIs in use, graphQL, gRPC, RESTful (REST) APIs, and Simple Object Access Protocol (SOAP) APIs, which were originally developed by Microsoft (which aren't as commonly found but are still used).

REST APIs use different formatting for the data exchanged between the API and the API consumer, including JSON and XML, while SOAP APIs use XML. REST APIs use a URL to request information instead of XML used by SOAP, making it far easier for developers to work with and can return results in Command Separated Value (CSV), JavaScript Object Notation (JSON), and Really Simple Syndication (RSS). Simply put, REST APIs that use JSON are far more commonly found than SOAP and XML.

API SECURITY

With APIs came an expanding attack surface beyond the web applications and databases. This new attack surface demanded a different approach to security. The need for a new security control to protect APIs is largely due to the fundamental differences between how traditional web applications process data versus how it's done with APIs.

In traditional web applications, the website is displayed (rendered) in the user's web browser due to data processing being performed on the server side (the side of the web server). This necessitated a WAF to protect unauthorized access into the web server, backend servers, and network infrastructure.

With APIs, to provide application functionality, the user interface (which is the API consumer) leverages APIs to get the needed data from the backend. This then leaves the job of rendering and maintaining the application state to the client. Thus, APIs have unprecedented reach and access to our precious data, and to the business logic of our applications, making them an attractive target for hackers and a challenging target to protect.

API SECURITY CHALLENGES

APIs serve and alter data. Their sole purpose is to take requests for specific data, or change of that data, from API consumers, request that data or change from the backend data source, then provide the data or change confirmation to the requestor. However, things can go bad quickly when the API requests aren't properly authenticated or authorized. In my past API hacking campaigns, I've found that a lack of proper API security is more common than not. I've found that most organizations today are still using traditional approaches to securing their web applications, and those approaches can't be used effectively to secure the APIs which now make up the majority of those applications.

An effective API security solution must understand the application business logic and what is a legitimate, versus illegitimate, API request (context); who is requesting it (authentication); whether or not they have permission to request it (authorization); and whether or not the person requesting it is indeed a human versus synthetic traffic generated by bad bots or tools (network telemetry). All of these requirements demand greater fidelity and even more innovative ways of doing attack detection and response. API security demands that we move beyond only packet inspection against web application firewall (WAF) rules.

API SECURITY SOLUTIONS COME TO MARKET

Realizing that WAFs were ineffective against the evolving nature of tactics and techniques being used by sophisticated threats, pureplay API threat management solutions arrived on the scene in 2015 and began establishing a new cybersecurity product category. This new breed of application security solutions focused on API security and used new approaches for detecting and preventing API attacks.

These new solutions required capturing and processing large amounts of network traffic, so they were designed to sit unobtrusively on the edge of the network where API traffic was flowing and to look for sequences of API calls that signaled malicious behavior. Because this meant sifting through a large amount of data, looking for patterns and deviations from those patterns, applying machine learning algorithms to sift through the data was a natural fit.

TRACEABLE

TRACEABLE

Most of us in the security industry are highly skeptical when we hear that "machine learning" or "AI" is going to solve our security problems (or any real-world problem for that matter). But one undisputed fact about AI/ML is that its likelihood of success is directly proportional to the amount and quality of data that it has access to.

ENTER TRACEABLE

Disrupting these other previous approaches to API security, Traceable came out of stealth in 2020 with a newer approach to detecting attacks on APIs using distributed tracing with ML. This unique approach to detecting API abuses using distributed tracing is what piqued my interest in its efficacy in stopping an API hacker like myself.

Traceable says they use distributed tracing and many machine learning algorithms to determine if API requests 1) went through all the required checks, including authentication and authorization; 2) called any unexpected services or granted access to unauthorized data; 3) what path the user took to the API; and 4) who the user/attacker is, regardless of their attempts to hide.

UNDERSTANDING DISTRIBUTED TRACING

To to understand why Traceable and their distributed tracing piqued my interest, it's important to understand what distributed tracing is. While it's common for it to be used in observing the performance and uptime of microservices-oriented architectures, it is novel to use it for detecting attacks on APIs. So what is it?

I find using analogies is a good way to explain extremely labyrinthine topics. For anyone who has used Apple's FindMy app, or something similar, you'll see that, once you add your device for tracking, it will literally map out everywhere the device goes. The app will tell you, step-by-step, where a device, such as your Apple Airpods, have moved (Figure 1).

FIGURE 1. Apple's FindMy App

Source: Apple

This is a great analogy of how distributed tracing works. Tracing works by tagging every request as it moves across distributed services in multi-cloud environments. It uses a unique identifier for every microservice, container, and infrastructure it touches. However, instead of tracking packets from hop to hop, similar to traceroute, distributed tracing allows the tracking of that unique identifier all the way from the top of the application stack, to the application's data layer, and to the infrastructure beneath it, providing rich context on everything that sees it.

Because services-oriented architectures are disparate by nature, unlike their monolithic predecessors, it's extremely difficult to demystify how transactions traverse the multiple layers of an API/microservices system -- to follow the path it takes from the API at the edge, through the microservices, and to the back-end. Distributed tracing solves this.

With its distributed tracing backend and ML algorithms, Traceable claims they can detect the attacks, and keep out the "evil-doers" targeting vulnerable APIs. So, are their claims true? Let's find out.

THE RESEARCH

THE RESEARCH

OWASP LISTS

The OWASP Top 10 lists are industry-accepted guidance of what OWASP believes those protecting web applications and APIs should be fundamentally concerned about when it comes to hardening web applications and APIs against attacks. It's for this reason that both OWASP Top 10 lists were selected as a rubric for my testing in this research.

Who is OWASP? The OWASP or Open Web Application Security Project is a nonprofit foundation, powered by a global brain trust of community members, that publishes open-source software, guides, tools, and methodologies for securing web applications across hundreds of chapters worldwide. OWASP is famously known for publishing the OWASP Web Application Testing Guide, OWASP Top 10 vulnerabilities in web applications, and now, the OWASP API Security Top 10, among others.

OWASP understands fundamentally that the way you attack APIs is different than what you do for the web, which is why OWASP created a separate list dedicated to API threats. If OWASP believes that APIs require their own list of top 10 threats from traditional web applications, then it should be without contestation that they should be secured differently as well.

For more information on each threat in the OWASP API Security Top 10 list and the OWASP (web) Security Top 10 list, see APPENDIX A.

TARGET APPS

Traceable markets itself as being able to detect and protect against the API attacks defined in the OWASP API Top 10 threats, and also against the traditional web application attacks described in the OWASP (web) Top 10 threats, as a WAF is able to do. If that's true then Traceable is a two-for-one deal, so of course I needed to test against those threats as well. But, I could not find a single test application that had known vulnerabilities for all of the threats on both lists, so I ended up using two apps.

My two target vulnerable applications were, crAPI, a training/demo application with purposely vulnerable APIs (https://github.com/OWASP/crAPI), and Juice Shop, a purposely vulnerable web application (https://owasp.org/www-project-juice-shop)

Both crAPI and Juice Shop have been developed with vulnerabilities inherent in their code in order for white hat hackers/penetration testers to exercise their skills in hacking web applications and APIs. While crAPI was designed specifically around the OWASP API Security Top 10, Juice Shop was designed with vulnerabilities more specific to web applications, such as the 10 threats defined by the OWASP Top 10.

TEST ENVIRONMENT

Throughout this research, a separation of duties was put into place where I (as the attacker) did not participate in the setup or implementation of anything on the back end which a cloud service provider (CSP) or application owner would be responsible for. Another individual on the Knight Ink team, "Blue Team" member Chris Daniels, configured the infrastructure, installed the vulnerable applications, and later Traceable. The ML training was performed by Traceable. While I did have a login to the Traceable platform, it was read-only and was enabled in order for me to see evidence of Traceable's actions, after my attack attempts.

To establish a baseline, both vulnerable applications were first installed and tested without Traceable. I validated that I could break into the apps by exploiting each vulnerability type listed in the OWASP lists, and I recorded how I did it (listed in the upcoming Tactics & Techniques section).

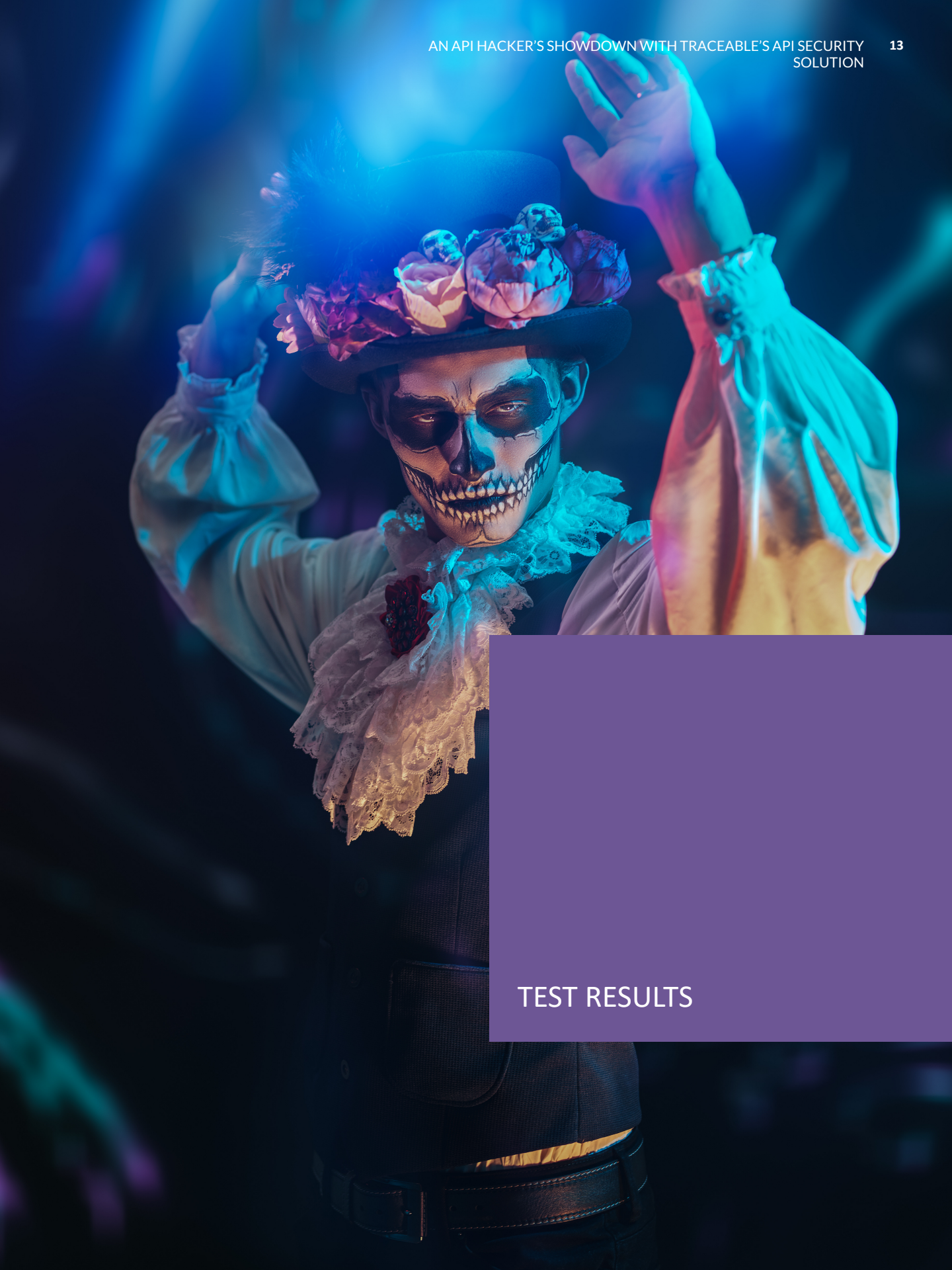
After I had my baseline, Traceable was deployed.

With crAPI, Traceable was deployed in front of the application using an NGINX plugin. With Juice Shop it was deployed in a sidecar configuration, which injects into the application. Because Traceable uses ML, it needs a training period to baseline each application. Traceable told me that this training happens through normal application use in a standard enterprise deployment. Since we didn't have any real users in our test environment, they provided traffic generation scripts for us to run to make sure that their platform was properly trained for each application.

TOOLS

When hacking APIs, I have my "go-to's" like any penetration tester. The tools I spend the most of my time in are Burp Suite and Postman, with each tool having its own strengths and weaknesses. While I do use other tools such as, mitmproxy, Kiterunner, and FuzzAPI for fuzzing, the tool will always of course depend on the engagement and the rules of engagement (RoE).

Tool	Description	Download
Burp Suite	Burp Suite by Portswigger is offered as both a free community edition and a professional version. While the professional version isn't going to break the bank for some, the free community edition should be feature packed enough for most users. The power behind Burp Suite is its numerous modules in what really amounts to a "Swiss army knife" for web application penetration testers.	www.portswigger.net
Postman	When all you need is a powerful API client and none of the other penetration testing tools like automated vulnerability scanners offered in Burp Suite, Postman should be your go-to. It's a feature-rich API client that offers the ability to create API requests from scratch, including not just the body but headers and the ability to specify OAuth tokens. Postman's power really lies in its collections allowing you to create and save the different attacks you might be using in a penetration test. Postman collections were instrumental in this project for being able to export my different attacks and send them to other members of this project team.	www.postman.com



TEST RESULTS

TEST RESULTS

This table summarizes the results of the efficacy of Traceable to detect and prevent the attacks I executed against both crAPI (representative of the OWASP API Security Top 10) and Juice Shop (representative of the OWASP Web Security Top 10).

OWASP API SECURITY TOP 10 RESULTS

crAPI + Traceable		
Attack Type	**Detection**	**Prevention**
API1:2019 Broken Object Level Authorization	✓	✓
API2:2019 Broken User Authentication	✓	✓
API3:2019 Excessive Data Exposure	✓	✓
API4:2019 Lack of Resources & Rate Limiting	✓	✓
API5:2019 Broken Function Level Authorization	✓	✓
API6:2019 Mass Assignment	✓	✓
API7:2019 Security Misconfiguration*	✓	✓
API8:2019 Injection	✓	✓
API9:2019 Improper Assets Management*	✓	✓
API10:2019 Insufficient Logging & Monitoring*	✓	✓

OWASP (WEB) TOP 10 RESULTS

Juice Shop + Traceable		
Attack Type	**Detection**	**Prevention**
A01:2021-Broken Access Control	✓	✓
A02:2021-Cryptographic Failures	✓	✓
A03:2021-Injection	✓	✓
A04:2021-Insecure Design*	✓	✓
A05:2021-Security Misconfiguration	✓	✓
A06:2021-Vulnerable and Outdated Components	✓	✓
A07:2021-Identification and Authentication Failures	✓	✓
A08:2021-Software and Data Integrity Failures*	N/A**	N/A**
A09:2021-Security Logging and Monitoring Failures*	✓	✓
A10:2021-Server-Side Request Forgery	✓	✓

***** Means the attack type identified generally is not directly attackable but instead leads to exploitable situations. A positive on these items means that the tool captured and alerted on symptoms of this attack type so that the security team could mitigate and prevent related issues.

** N/A=Not applicable in Docker deployment modes

Tactics & Techniques

TACTICS & TECHNIQUES

SPOILER ALERT: If you are planning to take the crAPI challenge and discover the crAPI vulnerabilities yourself, then don't read this section. Here I list out how I exploited each of the attack patterns in each OWASP list, OWASP API Security Top 10 via crAPI, and OWASP (web) Top 10 via JuiceShop. The step-by-step for how I exploited the OWASP API Security Top 10 vulnerabilities are listed in APPENDIX B.

crAPI (OWASP API SECURITY TOP 10)

Vulnerability Category	Specific Vulnerability Tested
API1:2019 Broken Object Level Authorization	BOLA vulnerabilities exist in the vehicle location function at {crapi_url}:8888/identity/api/v2/vehicle/<UUID>/location. This is because crAPI doesn't implement authorization controls at the object level, typically in the form of names, numbers, or IDs, which allows attackers to specify their own object referencing another user's resource.
API2:2019 Broken User Authentication	I discovered the check-otp function in the change password page in a different version of the API that allowed me to brute-force the OTP field with no rate limiting. This allowed me to change the password of another user and log in as them. The exploitation of this vulnerability requires chaining of multiple vulnerabilities inherent in the *forgot password page and the OTP function as well as navigating to a previous version of the API* where rate limiting isn't implemented.
API3:2019 Excessive Data Exposure	The community forum of crAPI is vulnerable to Excessive Data Exposure wherein if I create a new post and capture it using Burp Suite, or any other proxy, the actual API response contains the email address of all the users who posted to the forum. A second attack scenario is possible in the function for uploading a personal video to a user's profile. When capturing the API response to the upload, the API includes the shell commands used to perform the video conversion for the codec. This can be used to chain other attacks together to then perform command injection on the server by sending a request back to the API containing a modified command string to the codec conversion command by adding && <shell command>.

Vulnerability Category	Specific Vulnerability Tested
API4:2019 Lack of Resources & Rate Limiting	This vulnerability can be found in the chained attack scenario from API2:2019 Broken User Authentication above. By navigating to the previously published version of the API for the check-otp function, it's possible to fuzz the one-time passcode where there is no rate limiting implemented.
API5:2019 Broken Function Level Authorization	Broken function-level authorization refers to the user hierarchical permissions system being incomplete or broken as the name implies. By using the "DELETE" HTTP verb in a request to the administrator endpoint videos function at {crapi_url}/identity/api/v2/admin/videos/<#> it's possible to delete another user's video in the system from a regular user account.
API6:2019 Mass Assignment	Application frameworks will often automatically map user input in HTTP requests into internal objects. The function to generate a QR code for returning items will create a balance for the user once the item is returned. However, due to a mass assignment vulnerability in the {crapi_url}//workshop/api/shop/orders/<#> endpoint, it's possible to send an API request with a status of RETURNED for that specific order #, creating a balance in the account without the user having actually returned the item.
API7:2019 Security Misconfiguration	Security misconfigurations are not directly attackable but lead to exploitable situations.These can manifest themselves in different scenarios, from a misconfiguration in TLS that allows sensitive data to be sent in clear text without encryption; improper authentication configuration; or extra HTTP verbs that aren't necessary. After my attack campaign and looking at what Traceable detected and alerted on, I could see it called out vulnerabilities such as unauthenticated API calls that where handling sensitive data. APIs that were lacking encryption, and when only basic authentication was being used instead of more advanced and more secure types.

Vulnerability Category	Specific Vulnerability Tested
API8:2019 Injection	Using the same $ne (not equal) operator for MongoDB with the coupon_code parameter, I was able to successfully list working coupon codes that gave me different percentage discounts for my purchase. By using NoSQL injection against the {crapi_url}/community/api/v2/coupon/validate-coupon endpoint, I was able to apply a 75% discount code (TRAC075) giving me 75% off my purchase.
API9:2019 Improper Assets Management	Improper assets management is not directly attackable but leads to exploitable situations. It's important to be aware of these types of vulnerabilities being present in a system. It's a method I've used successfully in many previous API penetration tests. An example of improper assets management is when an organization leaves a previously vulnerable API accessible to the Internet which allows an attacker to target that version instead of having to go head-to-head with the latest API version with adequate security controls. For example, the attacker would hit https://api.victim.com/v1/api when https://api.victim.com/v2/api is the current and more secured version. After my attack campaign and looking at the Traceable API inventory, I could see that it was easy to identify shadow APIs that were externally facing, as well as easily filter for active APIs that, for example, had "v1" in their endpoint name.

Vulnerability Category	Specific Vulnerability Tested
API10:2019 Insufficient Logging & Monitoring	Insufficient logging and monitoring is not directly attackable but leads to exploitable situations. Not properly capturing the details of what's happening in the system leaves large blind spots. I have seen many real-life breach scenarios where not enough logging and monitoring was identified as making incident response and forensics (finding the who, what, where, when, and why) nearly impossible. For example, the Optus breach that hit the headlines recently in Australia, as recent as September, where 2.1 million records were exposed. In post-mortems of the breach it was discovered that no logging and monitoring was in place and that this allowed the adversaries to continue to come back over a long period of time to quietly and slowly dump data from the vulnerable APIs. After my attack campaign and looking at what Traceable logged and monitored about the application and all of it's API interactions, it was clear that Traceable was not leaving any blind spots. I mean, I could see even the payloads of every communication I had with crAPI.

JUICE SHOP (OWASP WEB TOP 10)

Vulnerability Category	Specific Vulnerability Tested
A01:2021 - Broken Access Control	This vulnerability is exploitable when a request is sent to the API endpoint {juiceshop_url}/rest/basket/<#> containing the basket ID of another user. Due to a failure by Juice Shop to authorize the request is coming from the actual basket owner, it allows me to view the basket contents of other users. Chaining this vulnerability with an account takeover, it was possible to then submit the basket contents as an order costing the victim money for the ordered items.
A02:2021 - Cryptographic Failures	The response packet in the reset password link returns all of the user details for the user including their password, which is using an unsalted MD4 hash able to be easily cracked with any free tool. In attempting to brute force the password reset security question, Traceable detected those brute force attempts and prevented me from continuing the reset effort.
A03:2021 - Injection	SQL injection is possible against the login endpoint at {juiceshop_url}/rest/user/login that allowed me to log in as an administrator. By simply specifying admin ' or '1'='1'-- in the email address field with any character as the password, I'm able to bypass authentication and log in with administrator privileges.
A04:2021 - Insecure Design	Insecure Design can encompass a number of different failures, including relying on hidden directories or files for protecting them from unauthorized access. It presumes only a legitimate request would come in for a hidden directory or file, since the person would need to know it exists beforehand. I leveraged Burp Suite's crawler in order to attempt to access a text file of typical directories and files on a web server. Traceable did detect these attempts as unauthorized attempts to access multiple directories and files outside of the web root.
A05:2021 - Security Misconfiguration	This vulnerability is introduced in a depreciated B2B interface that isn't in use anymore and was never properly removed. Using a Javascript beautifier (jsbeautifier) against the main-es2018.js file, I was able to grep for common web file extension support, such as php, apsx, etc. By doing this, I discovered the possibility to upload a malicious XML file through the customer complaint tab. I added a .zip file extension to my XML file (test.xml.zip), intercepted it with Burp Suite, and changed the filename back to .zip after it cleared the form submission checks. This vulnerability was possible because Juice Shop failed to validate all inputs on both the client and server side.

Vulnerability Category	Specific Vulnerability Tested
A06:2021 - Vulnerable and Outdated Components	Juice Shop contains a developer backup that I found in the vulnerable library challenge that contains a library *epilogue-js* referred to in *package.json*. The version of this library contains a typosquating issue. Juice Shop also contains an outdated white list that redirects me to cryptocurrency addresses that are no longer promoted. To find this, I loaded developer tools in my browser > debugger which lists the javascript files included in the website. I then clicked on the main-es2015.js file, I searched for a string called redirect, which previously redirected users to https://blockchain.info. Pasting this URL at the end of the Juice Shop URL {juiceshop_url}/redirect?to=https://blockchain.info/address/<cryptocurrency_wallet> worked.
A07:2021 - Identification and Authentication Failures	In the Injection vulnerability, we saw the login page is vulnerable to SQL Injection. Further authentication failures are in the same login page which makes it vulnerable to brute-force attacks. After executing Burp Suite Sniper against the login URL, I was able to discover that the password for admin is admin123.
A08:2021 - Software and Data Integrity Failures	Juice Shop does have a vulnerability for testing this. However, when Juice Shop is running from a Docker container or on a Heroku Dyno, which is how we had deployed Juice Shop for this research, this vulnerability was not exploitable for testing.
A09:2021 - Security Logging and Monitoring Failures	I identified a security misconfiguration that allowed me to view/list directories containing files (see A05:2021 above). A world readable directory was available under support/logs that contained log files being written by Juice Shop. The logging failure here was in Juice Shop not logging my access to this directory and files. My access to this directory and browsing through the logs was detected by Traceable. An administrator can then take the alerts and report it to administrators in a real life scenario to lock down the directory and/or block the access within Traceable.

Vulnerability Category	Specific Vulnerability Tested
A10:2021 - Server-Side Request Forgery	When accessing Juice Shop's profile page using the image upload function, I was able to provide my own URL instead of the URL used by Juice Shop to download the image exposing an SSRF Vulnerability in the "Link Gravatar" button. This SSRF attack was detected and stopped by Traceable as my risk score as a user had climbed above the threshold as a result of the previous attack attempts. While my score was unacceptably high as an attacker, I did confirm that the Traceable interface correctly identified it as a SSRF attack.

EVASION TECHNIQUES

In this section I detail an example set of the attacks I ran in an attempt to exploit the vulnerable apps and evade Traceable.

crAPI (OWASP API Security Top 10)

Evasion Techniques (Overall)
crAPI was, by design, vulnerable to all the OWASP API Security Top 10 vulnerabilities. Once Traceable was turned on, I attempted the same attacks again, which failed. I then tried evasion techniques like attempting to manipulate some of the fields that I believed Traceable was using for attack detection. This included modifying HTTP verbs and adding spaces into the API requests, which also resulted in failed attempts. Traceable detected each attempt and prevented the attack from succeeding.

NoSQL Injection
NoSQL Injection attacks are manifested in applications when the developer fails to sanitize user input, allowing the adversary to "inject" malicious input that executes a command on the server that the developer didn't anticipate. NoSQL Injection is used against applications using a NoSQL databases, unlike SQL Injection which is used against applications that use SQL databases such as Microsoft SQL, MySQL, PostgreSQL. In the case of crAPI, the NoSQL database is MongoDB.

Injection attacks can allow an adversary to execute unwanted code, enabling them to bypass authentication, exfiltrate sensitive data, modify data in the database, even compromise the database and underlying server hosting it.

MongDB supports the operator $ne, which stands for "not equal to". Because crAPI is vulnerable to NoSQL Injection, it was possible to use the $ne operator in my query to the API endpoint at {{url}}/community/api/v2/coupon/validate-coupon, using the body of `{"coupon_code": {"$ne": "TRAC075" },"amount": "10"}`.

Using the $ne operator for MongoDB with the coupon code parameter, I was able to successfully list working coupon codes that gave me different percentage discounts for my purchase. By using NoSQL injection, I was able to apply a 75% discount code (TRAC075) giving me 75% off my purchase.

Traceable was enabled after the attack initially succeeded. When attempting to send the NoSQL Injection attack in my API request, Traceable detected the MongoDB function $ne and blocked the request. I also attempted to try other HTTP verbs (referred to as HTTP verb tampering), such as POST, PUT and DELETE and inserting spaces (e.g. %2500) into the paths in the body in an attempt to evade Traceable. Other attempts were made to get around Traceable's detection mechanisms including using URL encoding. An example would be `GET /%69%6e%64%65%78%2e%68%74%6d%6c` as an alternative to GET /index.html. These efforts also failed and raised my threat score within the system making me unable to get through for further attack attempts.

Broken Object Level Authorization (BOLA)
Broken Object Level Authorization or "BOLA" attacks are the most common vulnerability I find in APIs. Developers oft-remember to authenticate requests but frequently fail to remember to authorize them, leading to sensitive data exposure. BOLA enables an adversary to directly access resources that they shouldn't be able to because the developer exposed an object and failed to define necessary limitations on who's authorized to request it.

crAPI is a vehicle management and service application. It offers a function vulnerable to a BOLA attack that allows users to monitor their vehicle's current location. In the scenario I exploited, I logged into crAPI using my adversary account. I then navigated to the Community forum tab which displayed all of the other users' posts. While the web page didn't display the vehicle UUIDs, I thought perhaps the API did in its response. I used Burp Suite to capture the response from the API when I browsed to this page, which did in fact contain the vehicle UUID of each user's post. I then used this UUID in the Dashboard vehicle locator instead of my own, which gave me the location of that user's vehicle.

Because of Traceable's user context awareness, it was successful in detecting my attempts at exploiting this vulnerability. I found out later that detecting my BOLA attack led to an alert and my user being blocked. No other evasion efforts to bypass Traceable were possible in this exercise. In an attempt to get around this scoring system that was being applied to my traffic as an offending user, I attempted to log out of the application, delete my cookies, and even took a new JWT token thinking Traceable used JWT tokens to identify or score specific user activity, which assumptions failed leading me to believe Traceable uses multiple variables tied together to identify a unique user including my IP address and other identifying factors.

Mass Assignment

Developers will often write their applications to automatically map user input in HTTP requests into internal objects, which can have disastrous affects if used by an adversary to introduce a parameter in the request that was never intended by the developer.

crAPI has an eCommerce page that allows users to order parts for their vehicles. The user is able to accrue a credit balance when items they've purchased are returned. When attempting to perform a return of a previously purchased item, the application will generate a unique QR code (similar to that of Amazon), allowing the user to take the item and QR code to their nearby UPS store to return it back to the store. A Mass Assignment vulnerability in this return function allowed me to get a credit for items I didn't actually return back to the store.

After capturing the response from the API when looking at past orders, the server responded with a list of supported HTTP verbs, one of them being PUT. The PUT request method creates a new resource or replaces a representation of the target resource with the request payload. This meant that I could modify the fields in the database with new information. By sending a PUT to {url}/workshop/api/shop/orders/<#> with a body containing status:"returned" I could force the API to update the database with a return status for items I never returned, thus increasing my available credit to buy more items I don't actually have to pay for.

In an attempt to bypass Traceable, I attempted other mass assignment attacks that crAPI was vulnerable to. Another mass assignment vulnerability exists in the workshop/api/shop/orders URL. By capturing the POST request sent by /workshop/api/shop/orders, the credit is reduced by $10. By forwarding it to Repeater in Burp Suite, I was able to change the value of the quantity field in the request body to a negative value, which increases the balance beyond the original value. Unfortunately, this did nothing in the way of evading Traceable's Mass Assignment vulnerability detection indicating a new URL in my attempt made no different to attempt to get around the high threat score my activity had summed up to at this point in the testing.

JUICE SHOP (OWASP TOP 10)

By design, Juice Shop is vulnerable to all categories in the OWASP Top 10 list. As expected, without Traceable protecting it, I was able to succeed in exploiting every vulnerability I tested except for the OWASP vulnerabilities marked N/A. However, once Juice Shop was instrumented with Traceable, the tables turned in favor of Juice Shop.

SQL Injection

I started my testing with a SQL Injection (A03:2021) against the login screen of the USERNAME field using the "' OR 1=1—" insertion. This easily gave me administrative access to Juice Shop.

Once Juice Shop was instrumented with Traceable, this same attempt was blocked. I attempted different mutations of this to try and bypass Traceable's detection, such as using "' OR TRUE". I also attempted "admin' or '1'='1'" as another SQL Injection mutation attempt. Traceable caught them all.

Another SQL injection vulnerability exists in Juice Shop, specifically in the checkout endpoint allowing an attacker to list all available coupons accepted by the system then apply multiple coupons to the order.

Unauthenticated Access

Juice Shop has an unauthenticated access vulnerability, which is categorized under OWASP Top 10 A01:2021 – Broken Access Control. There is a vulnerability in the BasketItems function when a form POST contains the BasketId parameter twice and the ID of two separate users' baskets will add the item to those users' baskets.

```
POST http://k8s-lab1-ingressk-
54200f1b6e-1969228450.us-west-
2.elb.amazonaws.com/api/BasketItems/
```

With a Body of:

```
{"ProductId":3,"BasketId":"6","Baske
tId":"7","quantity":1}
```

This unauthenticated access vulnerability is a failure by the application to enforce policy to prevent users from acting outside of their assigned permissions, in this case, adding checkout items to other users' baskets. In this testing, I was actually quite surprised that Traceable was able to detect my usage of multiple BasketIds in the variable to detect my attempt to specify other users' baskets. In an attempt to bypass Traceable's attempt to detect this, I added spaces between the numbers and quotes as well as attempted URL encoding, all of which failed.

NOSQL Injection

I then attempted to check Juice Shop for vulnerability to NOSQL injection on the product review page. I needed to determine how the NOSQL query needed to be formatted so I looked in the Burp Suite proxy history tab to see the captured packet from my initial query. I then sent the following query to the repeater within Burp Suite and modified the ID field to:

```
{ "id": { "$ne": -1 }, "message":
"NoSQL Injection!" }
```

I then created a new request to Juice Shop using the PATCH method to `{{URL}}/rest/products/reviews`.

What this request does is select ALL items in MongoDB in the products review tab that are not equal to -1 (all) and change the product review (because of the PATCH verb) to "NoSQL Injection!"

The HTTP PATCH verb is basically a set of instructions on how to update or modify a resource.

When Traceable began blocking my attempts to use the $ne operator here for MongoDB and detecting it as a NoSQL Injection attack, I attempted multiple mutations of $ne supported by MongoDB ($not and $nin), which also failed to evade Traceable's detection.

Broken Access Control
This vulnerability allows an unauthorized user access to resources they shouldn't have access to. Attackers are able to circumvent authentication and authorization controls governing access to sensitive data. Juice Shop is vulnerable to Broken Access Control in its view cart function for users.

This vulnerability is exploitable when a request is sent to the API endpoint `{juiceshop_url}/rest/basket/<#>` containing the basket ID of another user. Due to a failure by Juice Shop to authorize the request is coming from the actual basket owner, it allows me to view the basket contents of other users. Chaining this vulnerability with an account takeover, it was possible to then submit the basket contents as an order costing the victim money for the ordered items.

Thinking Traceable supported detection of this one type of Broken Access Control, I then attempted other forms of exploitation techniques for this vulnerability that chains other types of attacks together, including cross site request forgery (CSRF) that Juice Shop was vulnerable to hoping to get around Traceable's detection. These other vulnerabilities include being able to change the name of another user, delete all 5-star customer feedback, and accessing the administration section of the store. Unfortunately these attempts were also detected and stopped by Traceable, indicating that Traceable was able to detect and block multiple types of Broken Access Control attacks.

Conclusion

CONCLUSION

After hammering against an API security solution marketed as a distributed tracing approach to API threat management, I can conclude without contestation that it was successful in stopping me. Each and every attack was carefully crafted against the vulnerable APIs in an attempt at evasion, insertion, or denial of service. With Traceable protecting the apps, all the previously successful attacks failed.

An effective API security strategy needs to:

- ensure observability into network telemetry of all ingress and egress traffic of your APIs
- ensure a documented and regularly maintained patch and vulnerability management strategy
- know which APIs are riskier than others predicated on the type of data they're serving and their overall risk factors
- maintain a regularly updated asset management inventory of the APIs in every environment

After attempting to evade Traceable, it became quickly evident to me that their claimed user and application context awareness is critical in API security solutions, and that their unique approach of using distributed tracing is indeed effective in understanding the context and logic of the application it's protecting.

We've all trusted OWASP for over a decade on how we properly secure web applications. If OWASP believes APIs necessitated its own separate list of threats with the OWASP API Security Top 10, then we as defenders should recognize the need to secure them differently as well. Simply because APIs speak the HTTP protocol doesn't mean we should secure them like a traditional web server. The types of attacks that are levied against APIs, such as authentication or authorization vulnerabilities can't be detected with a rules-based solution like a WAF, thus necessitating a pureplay API security solution.

The empirical evidence presented as an outcome of this research and live fire exercises against two vulnerable APIs instrumented with Traceable's API security solution speaks volumes. Plainly and simply, Traceable was very effective at stopping my repeated exploit attempts against the known soft targets.

BIBLIOGRAPHY

Elliott, R. (2017, May 3). Web Application Firewalls and the Future of Website Security. Section. https://www.section.io/blog/web-application-firewall-definition-website-security/#:%7E:text=Types%20of%20Firewall%20and%20Web,still%20in%20high%20use%20today.

Wikipedia contributors. (2022, April 18). Web application firewall. Wikipedia. Retrieved May 23, 2022, from https://en.wikipedia.org/wiki/Web_application_firewall

Lane, K. (2021, April 8). Intro to APIs: History of APIs. Postman Blog. Retrieved May 23, 2022, from https://blog.postman.com/intro-to-apis-history-of-apis/#:%7E:text=Salesforce%20%E2%80%93%20Salesforce%20officially%20launched%20its,did%20business%20from%20day%20one.

Malik, N. (2021, June 22). The Difference Between REST and SOAP APIs. Dzone.Com. Retrieved May 23, 2022, from https://dzone.com/articles/difference-between-rest-and-soap-api#:%7E:text=REST%20APIs%20uses%20multiple%20standards,in%20the%20large%20sized%20file.

January 02, S. (n.d.). SOAP vs REST. What's the Difference? SmartBear.Com. Retrieved May 23, 2022, from https://smartbear.com/blog/soap-vs-rest-whats-the-difference/

Livens, J. (2022, November 11). What is distributed tracing and why does it matter? Dynatrace News. https://www.dynatrace.com/news/blog/what-is-distributed-tracing/

Schreyer, T. (2022, May 31). *Securing APIs With Observability & Distributed Tracing | Traceable App & API Security*. Traceable API Security. https://www.traceable.ai/blog-post/security-observability-why-tracing

OWASP Top Ten | OWASP Foundation. (n.d.). https://owasp.org/www-project-top-ten/

OWASP API Security Top 10. (n.d.). https://apisecurity.io/encyclopedia/content/owasp/owasp-api-security-top-10.htm

PATCH - HTTP | MDN. (2022, September 15). https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH

$ne — MongoDB Manual. (n.d.). https://www.mongodb.com/docs/manual/reference/operator/query/ne/

crAPI: Mass Assignment. (2022, June 22). levo.ai. https://levo.ai/crapi-mass-assignment/

BIBLIOGRAPHY

Hacking OWASP Juice Shop. (n.d.). Curiosity Kills Colby. Retrieved January 20, 2023, from https://curiositykillscolby.com/

juice-shop/SOLUTIONS.md at master · juice-shop/juice-shop. (n.d.). GitHub. https://github.com/juice-shop/juice-shop/blob/master/SOLUTIONS.md

EC Council. (2022, October 12). What Is Broken Access Control Vulnerability? Cybersecurity Exchange. https://www.eccouncil.org/cybersecurity-exchange/web-application-hacking/broken-access-control-vulnerability/

OWASP. (n.d.). API-Security/2019/en/src at master · OWASP/API-Security. GitHub. https://github.com/OWASP/API-Security/tree/master/2019/en/src

ABOUT KNIGHT INK

Firm Overview

Knight Ink is a content strategy, creation, and influencer marketing agency founded for category leaders and challenger brands in cybersecurity to fill current gaps in content and community management. We help vendors create and distribute their stories to the market in the form of written and visual storytelling drawn from 20+ years of experience working with global brands in cybersecurity. Knight Ink balances pragmatism with thought leadership and community management that amplifies a brand's reach, breeds customer delight and loyalty, and delivers creative experiences in written and visual content in cybersecurity.

Amid a sea of monotony, we help cybersecurity vendors unfurl, ascertain, and unfetter truly distinct positioning that drives accretive growth through amplified reach and customer loyalty using written and visual experiences.

Knight Ink delivers written and visual content through a blue ocean strategy tailored to specific brands. Whether it's a firewall, network threat analytics solutions, endpoint detection and response, or any other technology, every brand must swim out of a red sea of competition clawing at each other for market share using commoditized features. We help our clients navigate to blue ocean where the lowest price or most features don't matter.

We work with our customers to create a content strategy built around their blue ocean then perform the tactical steps necessary to execute on that strategy through the creation of written and visual content assets unique to the company and its story for the individual customer personas created in the strategy setting.

Contact Us

Web: www.knightgroup.co
Phone: (702) 637-8297
Address: 10845 Griffith Peak Drive, Suite 2, Las Vegas, NV 89135

ABOUT TRACEABLE

Overview

Traceable is the industry's leading API security platform that identifies all your APIs, tests your APIs before production, continuously evaluates API risk posture, stops known and unknown API attacks, and provides deep analytics for threat hunting and forensic research.

Traceable's platform applies the power of distributed tracing and machine learning models for API security across the entire development lifecycle. Traceable's explorable data lake provides insight into user and API behaviors to understand anomalies and block API attacks, enabling organizations to be more secure and resilient.

Traceable offers the industry's most flexible deployment capabilities to fit into any environment, scenario, and organization. Data can be collected through mirroring, native edge infrastructure, serverless, language-based, and other cloud-native environments.

Learn more at https://traceable.ai

Appendices

APPENDIX A

OWASP API SECURITY TOP 10

Realizing a need for its own separate project, the OWASP community came together and launched a new project dedicated to the threats APIs face outside of the original OWASP Top 10.

Broken Object Level Authorization (API1:2019). Often referred to as BOLA and previously referred to as Insecure Direct Object Reference (IDOR), BOLA vulnerabilities exist when a developer exposes an object in the API request allowing the attacker to substitute their own ID pointing to a different resource not associated to that authenticated user. An example of a BOLA attack would be an attacker replacing an API request for /api/patients/100 with a new ID of /api/patients/101.

Broken Authentication (API2:2019). Broken authentication vulnerabilities allow an adversary to assume the identity of another legitimate user. These can include not only weak hashed, plain text, or default passwords but also no JSON Web Token (JWT) token validation, or API keys that have no lifetime expiration. Attacks against Broken Authentication commonly involve brute force attacks and credential stuffing.

Excessive Data Exposure (API3:2019). Excessive Data Exposure vulnerabilities exist when an API is poorly hardened, providing far more sensitive data than what the API consumer requires or what the user needs to see relying on the API client to filter out only that necessary information. Developers will sometimes do this for forward compatibility should the business require more data in the future.

Lack of Resources and Rate Limiting (API4:2019). These vulnerabilities are present when an API doesn't do proper checking of payload sizes being sent in the API request or when an excessive number of API requests can be received and processed creating a Denial of Service (DoS) condition.

Broken Function Level Authorization (API5:2019). BFLA can be easily explained as a vulnerability allowing a regular user with no elevated privileges to access sensitive data outside of what they're authorized for or perform actions reserved only to administrative users or users granted a higher level of privileges.

Mass Assignment (API6:2019). These vulnerabilities exist when specific parameters and payloads aren't whitelisted that a user is authorized to change in the backend. This vulnerability enables an adversary to send a POST to the backend server and write new data to specific fields if allowed. An example here would be if an adversary was allowed to change their email address with their bank but was able to also specify a new balance for their checking account, changing it to a much larger number than what's there.

Security Misconfiguration (API7:2019). These vulnerabilities affect the API server itself, from application misconfigurations to the API server itself, even the underlying operating system. These can include missing patches, a lack of vulnerability and patch management, missing or outdated TLS encryption, and more.

Injection (API8:2019). With injection vulnerabilities, adversaries are able to insert executable code or SQL, NoSQL, LDAP, operating system, or other commands into their request. This occurs because of a failure by the API to sanitize user input and employ other forms of prevention against injection causing the application to inadvertently execute the code that was inserted into the request giving the adversary access to backend resources, administrative access, OS-level access, or more.

Improper Asset Management (API9:2019). I say this all the time. You can't protect what you don't know you have. These vulnerabilities appear when non-production (dev) APIs are left reachable, for example, dev or staging APIs that are not secured. These can also include shadow APIs that the organization is unaware of so are missing from patch and vulnerability management programs and are reachable by the adversary.

Insufficient Logging and Monitoring (API10:2019). This appears when the API either fails to or lacks verbosity in its logging and/or there is a failure by the organization to effectively monitor or alert on attacks affecting their APIs. This can include detection and response initiated by a Security Information and Event Management (SIEM) platform or SOAR (Security Orchestration Automation and Response).

OWASP Top 10

The OWASP Top 10 is a community-led project that maintains the most up-to-date threats to web applications so it can be used to test and harden web apps against vulnerabilities. It represents a broad consensus across its contributors of what constitutes risks to web applications.

The OWASP Top 10 vulnerabilities were last updated in 2021 with the following prioritized vulnerabilities to web apps:

Broken Access Control (A01:2021). Broken access control allows users to access objects or data outside their allowed permissions. Think of this vulnerability as authentication and authorization vulnerabilities.

Cryptographic Failures (A02:2021). This class of vulnerability causes the unintended exposure of sensitive data where encryption of that data fails allowing unintended recipients to view or modify it.

Injection (A03:2021). This vulnerability is introduced when user-supplied input is not properly sanity-checked, allowing the insertion of executable code into the query, which is inadvertently executed by the application. An example of injection vulnerabilities is escaping the sql statement in a login form with a ‘ and adding another SQL statement at the end causing the backend database to execute it.

Insecure Design (A04:2021). These vulnerabilities are introduced by missing or ineffective control design and create a flaw in the pre-code design stage of ”shift left” in security that enables the principle of “secure by design. Insecure by design is a flaw in the initial design of the app before a single line of code is even written.

Security Misconfiguration (A05:2021). **These vulnerabilities** can include multiple types, such as missing or relaxed permissions or lack of hardening. These vulnerabilities can also include unnecessary features, pages, ports, and even account permissions and can also include default passwords on accounts, such as guest accounts or administrator logins. This category of vulnerabilities can also include an application server or service that isn’t being regularly patched or part of a formal vulnerability or patch management program.

Vulnerable and Outdated Components (A06.2021). This class of vulnerabilities includes software, which includes both the application server, application, APIs, database management server (DBMS), and even third-party libraries being relied upon and the operating system (OS) being out of date, vulnerable, unpatched, or just aren’t aware of what version they are. This category can even include the configuration of the components.

Identification and authentication failures (A07:2021). These vulnerabilities can include a failure of the application to properly authenticate a user’s identity or even properly manage session states. Types of attacks against apps failing to identify and authenticate users and sessions can include credential stuffing and manual or automated brute forcing tools. If the app is missing or has improperly configured multifactor authentication; or uses plain-text or weakly hashed passwords.

Software and Data Integrity Failures (A08:2021) This class of vulnerability highlights failures in integrity violations of code and infrastructure, such as reliance on third-party code, plugins, software repositories, etc., and even content delivery networks. These vulnerabilities can also include failure to validate the integrity of auto-update servers where applications poll updates from a central software repository.

Security Logging and Monitoring Failures (A09:2021). When effective monitoring is missing of a web app, an organization fails to detect and respond to breach attempts or keep the mean time to detection/mean time to response (MTTD/MTTR) low since there are no detective controls to alert them to a successful breach event. This category of vulnerabilities can even include logging that is enabled but stored locally on the same server leaving it vulnerable to being deleted by adversaries once a compromise occurs.

Server-side Request Forgery (A10:2021). SSRF vulnerabilities occur when user-supplied input asks a web app to call an external URL that isn't sanitized by the app to ensure it isn't a malicious request.

All of these encompass the latest versions of the OWASP Top 10 and OWASP API Security Top 10 as they relate to the most commonly seen vulnerabilities in the wild and are updated regularly to reflect changes to that priority.

I can attest to the order in priority of the OWASP API Security Top 10 as BOLA and Broken Authentication vulnerabilities are among the most common vulnerabilities I find and exploit in tests.

APPENDIX B

API1:2019 Broken Object Level Authorization	"In testing this vulnerability, I discovered an unauthenticated access vulnerability in the vehicle locator of crAPI to demonstrate a BOLA attack in tracking the location of other drivers.." Within Burp Proxy tab + using Chromium (disable intercept) **Step 1:** Authenticate as **Victim 1** **Step 2:** Navigate to **Community** **Step 3:** Note the other vehicle's **UUIDs** **Step 4**: Navigate back to main **dashboard** **Step 5:** Replay dashboard request and replace vehicle **UUID** with another user's /identity/api/v2/vehicle/**8b9edbde-d74d-4773-8c9f-adb65c6056fc**/location
API2:2019 Broken User Authentication	Within Burp Proxy tab + using Chromium (disable intercept) Step 1: Navigate to **Forgot Password** Step 2: Input victim's **email address** Step 3: Change password with invalid **OTP** Step 4: Within Burp, **fuzz the OTP field** in the request using 0000-9999. This will trigger rate limiting. Step 5: Change URL to /identity/api/auth/**v2**/check-otp which has no rate limiting set
API3:2019 Excessive Data Exposure	Within Burp Proxy tab + using Chromium (disable intercept) Step 1: Authenticate as Victim 2 Step 2: Navigate to **Community** page Step 3: Add **Post** Step 4: Note other users' email addresses in response GET /community/api/v2/community/posts/recent

API4:2019 Lack of Resources & Rate Limiting	Within Burp Proxy tab + using Chromium (disable intercept) Step 1: Navigate to **Forgot Password** Step 2: Input victim's **email address** Step 3: Change password with invalid **OTP** Step 4: Within Burp, **fuzz the OTP field** in the request using 0000-9999. This will trigger rate limiting. Step 5: Change URL to /identity/api/auth/**v2**/check-otp which has no rate limiting set
API5:2019 Broken Function Level Authorization	Step 1: Send a DELETE method to {url}//identity/api/v2/admin/videos/{{video_id}} and it will allow a regular user to delete the videos of other users.

API6:2019 Mass Assignment	Within Burp Proxy tab + using Chromium (disable intercept)
	
	Step 1: Login to crAPI with a valid username/password
	Step 2: Navigate to SHOPS
	Step 3: Notice the credit/balance of $100
	Step 4: Navigate to PAST ORDERS
	Step 5: Find this API request in the HTTP HISTORY tab in Burp Suite
	Step 6: Notice the response from this request shows the supported HTTP METHODS (including PUT) and the STATUS parameter for the past orders of "returned."
	Step 7: Forward the {url}/workshop/api/shop/orders/3 request to REPEATER in Burp Suite
	Step 8: Add a new field to the BODY containing the STATUS parameter, which we'll set the value for of "returned".
	
	The body now looks like this:
	{
	"product_id": "1",
	"quantity": "2",
	"Status": "returned"
	}
	
	Step 7: Reload the SHOP page and notice the credit balance has now increased by $10 to $110.

API8:2019 Injection	Within Burp Proxy tab + using Chromium (disable intercept) Step 1: Log into crAPI with a valid username/password Step 2: Click on + ADD COUPONS Step 3: Note that the application prevents the use of random coupon codes Step 4: In the HTTP HISTORY tab of Burp Suite, click on the recent request attempt to use an invalid coupon code and note the 500 status code response from the API Step 5: Modify the API request with a new body by inserting the $ne (MongoDB not equal to function) and set the value to null. Your new body should look like this: {“coupon_Code”:{“$ne”: null}} Step 6: Send the request to the API, noting the response from the API containing a valid coupon code of TRAC075. Step 7: Send the same API request but replace null with TRAC075 and note the API response containing a brand new coupon code of TRAC065 allowing an attacker to stack multiple discounts.

> The API attack surface is the greatest existential threat nations and organizations face today.
>
> -ALISSA KNIGHT

About The Author

Over the last decade, Alissa quickly gained notoriety as an API hacker after publishing several vulnerability research reports, hacking 55 banks through their APIs in less than a week; millions of patient records after she hacked 30 mHealth and FHIR APIs in less than a week; and demonstrated the ability to remote control of any law enforcement vehicle through the automaker's APIs.

Over the last decade, Alissa has published numerous vulnerability research reports into hacking different market segments through their APIs. 2019 was the first report, where she was targeting financial services and FinTech mobile applications and APIs. In 2020, following the start of the COVID-19 pandemic, it was the overnight boom of mobile health apps (mHealth) and APIs giving patients remote access to their healthcare providers from home. That same year, she published evidence on how she was able to take remote control of law enforcement vehicles through the automaker's APIs. Then in 2021, she published her research on how she hacked millions of patient records through Fast Healthcare Interoperability Resources (FHIR) APIs of healthcare providers and payers. This year, in 2022, she has published research on hacking banks and cryptocurrency exchanges through their APIs.

Alissa Knight is a 22-year veteran of the cybersecurity industry as a white hat hacker, published author, filmmaker, and serial entrepreneur who sold two cybersecurity companies in M&A transactions to public companies.

Knight Ink
+1 702 766 6362
10845 Griffith Peak Drive
Suite 2
Las Vegas, NV 89135