On the Usage and Vulnerabilities of API Systems

Conner D. Yu
William & Mary

On the Usage and Vulnerabilities of API Systems

Connor D. Yu

William & Mary

Contents

I. Introduction

To some, Application Programming Interface (API) is one of many buzzwords that seem to be blanketed in obscurity because not many people are overly familiar with this term. This obscurity is unfortunate, as APIs play a crucial role in today's modern infrastructure by serving as one of the most fundamental communication methods for web services. Many businesses use APIs in some capacity, but one often overlooked aspect is cybersecurity. This aspect is most evident in the 2018 misuse case by Facebook, which led to the leakage of 50 million users' records.[1] During the 2018 Facebook data breach incident, threat actors used Facebook developer APIs to obtain the personal information of Facebook users over the span of a year. This incident raised many concerns due to it potentially violating users' privacy. This entire third-party data harvesting incident might not have occurred if Facebook had a more proactive API security and management system. This example was a very considerable data breach, but a similar attack could happen to any business that does not correctly understand and implement different security requirements based on the paradigm shift that APIs present. This raises the issue of how to properly secure APIs in a world where they can be misused, with Gartner Research stating that APIs will be a major attack vector in the next few years due to their widespread use.[2] To tackle this problem, this research paper sets to discuss the nature of APIs and their security vulnerabilities. Then, we will go into possible preventative measures and design decisions to secure APIs depending on the usage context. This paper aims to offer a blueprint on security best practices to secure API systems across various use cases.

[1]Bernard Harguindeguy, *Facebook Data Breach Highlights API Vulnerabilities*, https://www.pingidentity.com/en/company/blog/posts/2018/facebook-data-breach-highlights-api-vulnerabilities.html
[2]Mark O' Neill, *How to Build an Effective API Security Strategy*, https://www.gartner.com/en/documents/3834704

II. Background Information

The National Institute of Standards and Technology (NIST) defines an API as "[a] system access point or library function that has a well-defined syntax and is accessible from application programs or user code to provide well-defined functionality."[3]

In the pre-2000s era, the internet mainly involved a person or a group of individuals, usually the website owner, updating the information displayed on the website. This style of static webpages, which focuses on displaying information, constituted what scholars now deem Web 1.0.[4] It was not impossible to get information from other websites as web scraping existed since 1993, only four years after the debut of the world wide web.[5] However, the developer would have to take time to build the web scraping script, and if the format of the website was changed, the scraper would no longer work. Other developers manually scraped content, copying the information themselves, which was inefficient. Over time, the layout of the internet gradually changed, with many users increasingly interacting with websites dynamically, while complex web services emerged by utilizing APIs to communicate with other web services. [6]

APIs are a way to pass instructions and information back and forth between web services in an understandable way, which allows any system to embed itself into the internet and make use of the endless amount of organized data the web provides through a standardized communication protocol. This standardization allows programmers to interface complex programs and services

[3]*Application Programming Interface*, https://csrc.nist.gov/glossary/term/Application_Programming_Interface
[4]*Funding a Revolution: Government Support for Computing Research*, https://www.nap.edu/read/6323/chapter/9#181. *Web 1.0 to Web 2.0: an observational study and empirical evidence for the historical r(evolution) of the social web*, https://pdfs.semanticscholar.org/36be/5c171e42d63f412b1d7aa85e7bb76b8bde9a.pdf
[5]*History of Search Engines: From 1945 to Google Today*, http://www.searchenginehistory.com/
[6]*Web 1.0 to Web 2.0: an observational study and empirical evidence for the historical r(evolution) of the social web*, https://pdfs.semanticscholar.org/36be/5c171e42d63f412b1d7aa85e7bb76b8bde9a.pdf

together to provide a much more featureful product to the end-user without having to create and refine all the individual components from scratch.

A. Common Use Cases

There are various application usages of APIs, but this paper introduces three common API usages to illustrate the critical impact APIs have on people relying on web services for their daily lives.

i. Mobile Apps

Smartphones are now ubiquitous as most people use mobile applications for various essential daily tasks. Many commonly used mobile applications utilize API calls to facilitate critical functions.

For example, Uber, which offers mobility-as-a-service for ridesharing and food delivery, uses the Google Maps API to help drivers navigate to their destination.[7] This API is a critical component of Uber's business model, and the company is dependent on Google Maps API to display information to delivery workers and app users. Building on top of a service provided by a separate company, Uber avoided creating their own global mapping service from scratch. Apps, such as Uber, could probably not have existed in the pre-2000's era, even if Google Maps existed, due to the nature of how web services communicated then. There was no way to easily share data, and Uber would have had to either build their own system or build a complicated web scraper to parse the geolocation information. Developing a web scraper likely would have taken considerable time and resources and could have been broken each time Google Maps was modified. Now, with the Google Maps API, Uber is easily able to integrate third-party mapping features into its services.

[7] *Uber S-1*, https://www.sec.gov/Archives/edgar/data/1543151/000119312519103850/d647752ds1.html

ii. Web Services

Many web services[8] use APIs to rely on third-party software that they do not want to build from the ground up. This not only decreases time-to-market, but it also improves the user experience by giving the users access to services the company does not have the time or resources to create.

An example of this is third-party authentication. Many websites now allow you to log in via various social media platforms, such as Facebook, Twitter, or Google. Every time the application loads, the website checks whether the user is already logged into the corresponding social media platform, offering them an option to log in if they are not. When it is confirmed, the API gives the website the credential information.[9] Not only does this allow users to easily log in, but it also reduces the impact of a data leak. If less information is stored on various websites, less information is subject to compromise when a breach against a website does occur. However, third-party authentication also raises privacy concerns, as those authenticators most likely are collecting information on user activities when they login to use web services.

iii. Internet of Things

NIST defines the Internet of Things (IoT) as "a rapidly evolving and expanding collection of diverse technologies that interact with the physical world."[10] IoT is a product of the combination of large technological advances in various computing fields, such as embedded systems,[11] low-price software, mobile computing, and more.[12] IoT devices connect and exchange information with

[8] *What is a Web Service?*, https://www.ibm.com/docs/en/cics-ts/5.2?topic=services-what-is-web-service
[9] *Facebook Login*, https://developers.facebook.com/docs/facebook-login
[10] Katie Boeckl, *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks*, https://nvlpubs.nist.gov/nistpubs/ir/2019/nist.ir.8228.pdf
[11] A computer system within a larger system. https://archive.org/details/embeddedsystemsd0000heat/page/n5/mode/2up
[12] Katie Boeckl, *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks*, https://nvlpubs.nist.gov/nistpubs/ir/2019/nist.ir.8228.pdf

other IoT devices and web services from the internet, and a lot of that communication is done through APIs.

An example of how IoT devices use APIs is the automation platform home assistant. It is the central control panel for home automation, connecting locally to IoT devices in the user's house.[13] Through this, it is possible to connect devices ranging from a doorbell to an Amazon Alexa, a voice-based smart virtual assistant. The home assistant device would communicate with these devices through API calls, so if the user instructs the home assistant to turn off the lights, it can send a request to the smart light bulb to shut itself off.

While IoT devices are extremely useful, they do carry severe security risks. Due to the limited computational resources of these devices, IoT devices normally have limited abilities to apply new security updates. This means that any vulnerabilities that are discovered and subsequently patched in one version of the device will not be applied to existing devices. Furthermore, due to the lightweight nature of the devices, they do not have a lot of computational power, meaning a lot of communications through the network use weak encryption or do not use encryption at all, leaving them vulnerable to man-in-the-middle attacks.[14] Furthermore, many IoT devices connect to the internet, leaving them exposed to various threats from anywhere in the world if an external firewall is not present.[15]

One such attack is Mirai, a malware that continuously scanned the internet for vulnerable IoT devices that used common or default passwords. If one is detected, the device was infected to

[13]*Home Assistant*, https://www.home-assistant.io/
[14]Kerry A. McKay, *Report on Lightweight Cryptography*, https://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf
[15]*Securely connect IoT devices to the Cloud*, https://docs.microsoft.com/en-us/learn/paths/securely-connect-iot-devices/

be added to a large botnet, which was used to perform various attacks, mainly distributed denial of service attacks.

iv. Overall Observations

APIs are everywhere, and they play a critical role nowadays, even though they may not be the most visible technology. They are similar to the modern-day car assembly method, where instead of all the car parts being created in one location, car manufacturers assemble various car parts already manufactured from around the world.

APIs allow developers to use advanced functionality without having to reinvent the wheel themselves. Allowing disparate services to interface, APIs decrease development time and improve the end-user experience. Although they are incredibly convenient, convenience introduces risk.

III. Issues Related to APIs

APIs offer a variety of benefits to developers and end-users alike, which has led to their widespread adoption. Unfortunately, this ubiquity leads to a large potential attack surface[16] for threat attackers to target, leading to eventual exploitation as attackers will gravitate towards commonly used technology. Gartner Research predicts that APIs will be the primary attack vector[17] that attackers will utilize.[18]

While APIs are very useful, they introduce their own security vulnerabilities. At their core, APIs are simply a standardized communication protocol, and, like every form of electronic

[16]*The boundary of a system that an attacker can try to enter*, https://csrc.nist.gov/glossary/term/attack_surface
[17]The path by which an attacker gains access to the system, http://gauss.ececs.uc.edu/Courses/c6055/pdf/attackvectors.pdf
[18]*API Security Gartner*, https://www.l7defense.com/cyber-security/api-security-gartner/

communication, they have inherent vulnerabilities to certain kinds of attacks. These attacks are not inherently dangerous because of the nature of APIs, and some of these attack methods have been around for decades. They pose a risk due to the widespread usage of APIs, as so many systems and servers use APIs as their primary form of communication. Attackers will then prioritize developing attack methods for APIs. Due to that widespread usage, they can minimize their effort while maximizing their potential reward. Developers should carefully plan out how they are going to secure their systems and look at how their protocols could be compromised.

The next section will review common cyber-attacks that can affect APIs, and the subsequent section will examine some common bad practices that can disrupt APIs and potentially the organization's IT infrastructure.

A. Common Cyber Attacks

As mentioned, this section will describe some of the common cyber-attacks that threat actors use to infiltrate or bring down API systems.

i. Denial of Service (DoS)

Denial of Service (DoS) attacks involve the prevention or delaying of access to resources.[19] This can be done in several ways, such as spamming a server enough times with one machine to increase traffic to a level the servers cannot bear or sending a malicious request that the server is not properly configured to handle. An example of this was when a vulnerability from an improperly

[19]Michael Nieles, *An Introduction to Information Security*, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf

validated Cisco API let attackers disconnect all participants from a call whenever the attacker sent a malicious API request.[20]

An iteration of this type of attack pattern is a Distributed Denial of Service Attack (DDoS). Like DoS attacks, DDoS involves using numerous hosts to perform the task instead of just one host.[21] DDoS attacks have been around for decades, have found a new exploitation method in APIs.[22] A DDoS attack appears like thousands of malicious hosts making billions of API requests to a point more that the server can handle and bringing down the targeted web service for a period of time.

ii. Injection Attack

An injection attack involves an attacker injecting code or malware into a program or query into a system so that they can execute remote commands to read or modify a database or change some data on a website. There are many different forms of injection attacks, such as CRSF injection,[23] cross-site-scripting,[24] and SQL injection attacks[25] (which are the most widely known type of injection attacks).

[20]*Cisco Meeting Server API Denial of Service Vulnerability*, https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-meetingserver-dos-NzVWMMQT
[21]Andrew Regenscheid, *Security Best Practices for the Electronic Transmission of Election Materials for UOCAVA Voters*, https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7711.pdf
[22]P J Criscuolo, *Distributed Denial of Service Tools*, https://www.osti.gov/biblio/792253-EbRKPo/native/
[23]According to NIST, CSRF (Cross-site Request Forgery) is when a valid connection is hijacked by another connection, causing an unwanted action in the valid connection. https://csrc.nist.gov/glossary/term/Cross_site_Request_Forgery
[24]According to NIST, Cross-Site Scripting is a vulnerability that allows attackers to inject malicious code into an otherwise benign website. https://csrc.nist.gov/glossary/term/Cross_site_Scripting
[25]According to NIST, SQL Injection is an attack that looks for websites that pass insufficiently-process user input to database back-ends. https://csrc.nist.gov/glossary/term/SQL_injection

One of the new forms of injection attacks is an API injection attack. This involves the attackers inserting SQL queries [26] into input fields through the SQL database [27] underlying the system. This can lead to major problems if the user can access the database directly and the input is not directly sanitized. Damages caused by API injection attacks can vary based on how badly the database is configured and how much confidential information is in the database. The consequences of an API injection attack could involve the attacker gaining access to confidential information and, in the worst-case scenario, deleting or modifying the database.

iii. Man-in-the-Middle (MITM)

Like wiretapping, Man-in-the-Middle (MITM) attacks involve the attacker intercepting communicated data and passing it along to the valid entity and, in the worst cases, modifying the data. This could let the attacker obtain sensitive information, such as API keys, granting them access to the user's account and all the resources and API provides. An example of what the consequences of a MITM attack could look like involve the end-user having their confidential data leaked.

iv. Parameter Tampering

Parameter tampering involves the manipulation of parameters exchanged between client and server to modify application data, such as user credentials and permissions. It usually takes advantage of programmers relying on hidden or fixed fields as the only security measure for certain operations. Attackers are then able to gain increased application functionality and control when such fields are subverted.[28] This can allow the attacker to control any of the resources that the API

[26]According to Microsoft, statements to modify data in a database. https://docs.microsoft.com/en-us/sql/t-sql/queries/queries?view=sql-server-ver15
[27]According to Microsoft, a database that stores a specific type of structured data. https://docs.microsoft.com/en-us/sql/relational-databases/databases/databases?view=sql-server-ver15
[28]*Parameter Tampering*, https://www.imperva.com/learn/application-security/parameter-tampering/

is responsible for. Attackers can perform this attack directly to exploit the application itself or attach a user through a man-in-the-middle attack.[29] An example of this attack was when, in 2019, there was a PHP [30] script that targeted a website that relied on an API to validate its payments. Through parameter tampering, the attacker could modify the selected payment amount.[31]

B. Common Bad Practice

Development is a balancing act between cybersecurity and development time. Many times, developers focus on simply creating the system and ensuring its functionality, inherently assuming that it's secure. On the other hand, if the developers were to ensure that the system is *completely* secure, development time would stretch far too long, and the firm may ship an obsolete system. Instead, developers must strike a balance, implementing reasonably secure practices to dissuade threat actors from attacking a system. At the very least, developers should avoid a few common bad practices that can cause system disruption, sometimes without attackers even having to be present.

i. Unencrypted Communications

One of the most basic protection measures is the implementation of encryption of communicated data in API requests. Encryption prevents unauthorized third parties from snooping or modifying communications between devices, thwarting attacks such as the Man-in-the-Middle attack. Developers commonly use Transport Layer Security (TLS) protocols to protect the communication stream.

ii. Organizational IT Awareness

[29] *Web Parameter Tampering*, https://owasp.org/www-community/attacks/Web_Parameter_Tampering
[30] According to the PHP website, a widely-used programming language mainly used in web development. https://www.php.net/manual/en/intro-whatis.php
[31] *CVE-2019-9063 Detail*, https://nvd.nist.gov/vuln/detail/CVE-2019-9063

In many organizations, employees leave any technical issues to the IT team.[32] This is understandable, as learning how to deal with those issues can take a lot of time and disrupt the workflow of those employees. However, this ensures that the technologically unsavvy members of the organization never learn how to properly use their machines, leaving the firm vulnerable to social engineering or other simple attacks.

To counteract this, all businesses should have a program to ensure that all employees have some working knowledge of common attack methods and how to counteract them. This should decrease the number of social engineering attacks [33] and issues caused by unaware workers. Many organizations should prepare an incident response plan to increase awareness and mitigate damage caused by potential cyber incidents.

IV. API Management Proposal

This section presents various techniques to enhance API security and how to implement each of them. It should be noted that, in the real world, many third-party cybersecurity vendors bundle these features together for a consolidated solution.

A. Three types of APIs

To reasonably manage security concerns while ensuring the streamlined development of their product, developers should tailor their security needs to their product, implement the security measures that are reasonable for the platform, and balance development speed and security measures.

[32]Mark Wilson, *Building an Information Technology Security Awareness and Training Program*, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-50.pdf

[33]According to NIST, an attempt to trick someone into revealing information to attack systems or networks. https://csrc.nist.gov/glossary/term/social_engineering

There is a vast range of APIs available, and each will require a different level of security. In this paper, we define three different kinds of APIs based on the contextual use of each system, which alters security requirements:

- Public: APIs that rely on publicly available information, requiring relatively less security.
- Protected: APIs that offer resources limited or exclusive to a particular group of users, most often paying customers.
- Private: APIs that handle sensitive information that has dire privacy and legal consequences should it be compromised.

While all three are discrete entities, they all fall on a continuous range of required security measures. A given API may fall between public and protected security levels, in which case the developer should modify the recommended security practices to best suit their needs.

Public API Example

Google Maps API: Uses the API to communicate geolocation information, such as the best route to get to a destination. The API uses information and resources that Google has already made publically available.

Protected API Example

Twilio API: This API allows the user to make a request to send a text message to a designated number, charging a small fee for every request. As it charges a fee for its services, it should be more heavily secured.

Private API Example

Fitness tracker APIs: How fitness trackers get information to your phone. As it handles health data, which is protected under HIPPA, it should be protected rigorously, as data leaks can incur massive fines.

B. Suggested Security Practices

In the following section, we go over various security practices that developers can implement. The first few security practices we present should not require a lot of modification to the system, but as we move on, some practices involve the fundamental reorganization of the system's architecture, which would take more time and resources to implement.

i. Encryption

As discussed earlier, this involves the cryptographic transformation of data such that a human cannot access the original data.[34] Obviously, these can be deciphered upon arrival at their intended destination, provided the recipient has the intended 'key' to decrypt the files. This has the added benefit that any data that is intercepted by an attacker would be very difficult to decipher, helping prevent something like a Man-in-the-Middle attack. With the massive amount of data that any organization outputs, encryption makes it very hard for attackers to find, intercept, and decipher important information. This does have a few added costs, though. These costs include a small amount of computational resources and handling key management. If a key is leaked, then attackers would be able to decipher any data that is intended for the user of the leaked key. Overall, encryption is a very important aspect of securing data.

ii. User Registration

Should a business choose to host an API, they should consider vetting their potential users by controlling their access to the API resources. This means that if a developer wants to use an API, they should first have to register with the service and request an API token. This ensures that only registered users have access to the service. This means that developers know who exactly is

[34]Keith Stouffer, *Guide to Industrial Control Systems (ICS) Security*, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf

making requests to the system, and they can deny requests to certain tokens if there is any nefarious activity detected, preventing further threats. This can limit potential DDoS attacks with billions, or trillions of requests sent from servers across the world and gives clarity as to who made what requests, helping developers trace any nefarious activity. This does place a small barrier to entry, and potential users may not think the service is worth the hassle of signing up.

iii. Resource Management

Another measure that developers should consider involves ensuring their users do not abuse their access to the resources once they have been registered. This involves rate-limiting, which is when the user has a cap on the number of calls they can make per day or week, or they are charged on each request (when funds run out, they are unable to make requests). Measures such as this could also help prevent DDoS attacks. This would be highly effective if an API is hosted on a cloud computing platform. Due to the nature of the elastic allocation of computing resources, every call in a DDoS attack could be theoretically handled, but it would cost the business a large amount of money. If the business used rate-limiting, they could limit the amount each user can make at any given point in time or how many calls are allowed to be processed, all to avoid the resources being used maliciously.

iv. API Endpoint Isolation[35]

APIs can allow for communication within an organization or take in requests from the general public. Any request that comes in from the general public has the possibility of being subject to an injection attack. So, to the extent possible, API endpoints should be deployed on

[35] *API Endpoints*, https://docs.openstack.org/security-guide/api-endpoints.html

separate hosts, isolated from the system. Furthermore, private APIs do still have the possibility of being compromised, so they should still be isolated to the point that makes sense within the system.

This isolation could be done through hardware isolation, with the API residing on a separate machine altogether, or it could be isolated virtually through something like a virtual machine (VMs).[36] VMs do have their own vulnerabilities, including a possibility to enter a VM on the host machine through a separate VM. But, overall, those types of attacks are very difficult to pull off and impractical in the real world. While hardware isolation is the most optimal solution, VMs are highly secure. Technology has been around for decades, and most modern-day solutions are very safe. API endpoint isolation has massively positive security implications, as it significantly decreases the risks of injection attacks, but they have the cost of increased system complexity, and sometimes isolation is impossible given the architecture of the system.

v. Zero Trust Architecture Implementation[37]

Zero trust architecture is a paradigm that shifts defenses from static, network-based perimeters, such as firewalls, to users, assets, and resources.[38] There is no implicit trust granted to any machines or users based on a network location or asset ownership. Implementing this practice involves the fundamental restructuring of the system's architecture. Utilizing network segmentation and preventing lateral movement better helps defend modern digital environments. This significantly decreases the likelihood of a breach. However, it does lead to increased development time, as developers will have to authenticate themselves constantly, and request permissions for every resource they need.

[36]According to NIST: Software that allows a single host to run one or more guest operating systems. https://csrc.nist.gov/glossary/term/Virtual_Machine
[37]*What is a Zero Trust Architecture*, https://www.paloaltonetworks.com/cyberpedia/what-is-a-zero-trust-architecture
[38]Scott W. Rose, *Zero Trust Architecture*, https://www.nist.gov/publications/zero-trust-architecture

vi. Microservice Architecture Implementation[39]

Microservice architecture involves breaking down larger applications into smaller independent parts, communicating a large amount of the time through APIs.[40] Each microservice implements a specific end-to-end domain or business capability within a certain context boundary, and then each piece must be independently deployed.[41] Its smaller codebase makes it faster to develop. Also, an attacker compromising a single component of a system does not mean they have access to the entire system.[42] This approach would most likely be most effective when building new systems, as breaking down legacy systems into microservices architecture takes a large amount of time and resources.

C. Proposal

There are a ton of other security practices and software architectures that can better help businesses protect their APIs and systems, but businesses need to carefully balance their security measures with their development speed and overall user experience. If they spend too long securing their systems, not only could their products be obsolete by the time they make it to market, but the overall user experience could also suffer if they must jump through too many security measures to access the business's resources. On the other hand, if the business puts no effort into security and focuses solely on the short-term development of their products, their systems will be very insecure, making their data vulnerable to attackers and jeopardizing the end-user. Instead, businesses should carefully evaluate the sensitivity of their data and the legal and business

[39]Thomas Bush, *Which is More Secure: Monolith or Microservices*, https://nordicapis.com/which-is-more-secure-monolith-or-microservices/ Ramaswamy Chandramouli, *Security Strategies for Microservices-based Application Systems*, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf
[40]*What is Microservices Architecture*, https://cloud.google.com/learn/what-is-microservices-architecture
[41]Nish Anil, *Microservices Architecture*, https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture
[42]Ramaswamy Chandramouli, *Security Strategies for Microservices-based Application Systems*, https://csrc.nist.gov/publications/detail/sp/800-204/final

consequences of any breaches. They should carefully evaluate how sensitive their data is, and properly work to make sure that their systems are secured accordingly.

i. Public

Observing our classification of public APIs, which are said to be APIs that rely on publicly available information, we propose that public APIs should encrypt their API calls and either user registration or resource management in place.

For the rationale, from a user perspective, we want as little user data exposed as possible. While some information may seem innocuous, attackers may be able to take advantage of the aggregation of data to gain a better understanding of their target. This, combined with other gathered information, could allow a threat actor to take advantage of consumers. In the end, it's much better to secure everything, as any amount of information, when aggregated, can lead to potential damages to the end-users. A counterargument to encryption is that it does add extra complexity through key management and computational processing. This should be carefully considered, but in the end, adding encryption vastly improves data protection compared to the effort required.

The other recommendation is using either user registration or resource management at a minimum. User registration allows the service or system to know who is sending in each request, which can greatly assist developers when they try to identify nefarious activities. However, it does add a bit of a barrier to use. Developers may not want to go through the process of setting up an account for a service, especially if they deem the functionality trivial. An alternative could be rate-limiting. This could make sure that the company servers are not damaged by over-processing, or if they are using cloud computing, it can help ensure that they are being charged for API calls that are not made in good faith.

For APIs without user management, resource management should allow for company resources, whether they are on-premises or on the cloud, to be protected from many malicious attacks. While both user management and resource management work well separately, they can be even more effective when combined, as the developers can easily shut down or put a cap on requests from a certain user, usually identified by an IP address.

ii. Protected

Protected API refers to APIs whose resources are only offered to a limited group of people, usually paying customers. As API resources are a bit more valued, especially within the intellectual property context, they should be more protected than public APIs, which usually use publicly available information or resources. For the protected APIs, they should still follow the recommended security practices of the public APIs, but they should also isolate their endpoints as much as possible.

The point of API endpoint isolation is to make sure that threat actors are not able to easily inject code into API requests and compromise the system. In general, these APIs are very enticing to attackers, as compromising these systems leads to greater access to highly valued resources. To circumvent this, developers should implement API endpoint isolation. This means, whenever possible and to the greatest ability of the company, developers should isolate their public APIs and carefully monitor and process the requests that come in. Only when they have been vetted should the systems let the request into the system to be processed. Separating the processes as much as possible and checking the requests as much as possible help prevent injection attacks, as the attacker would have to know a lot about the inner workings of the system to bypass the vetting of the isolated endpoint.

iii. Private

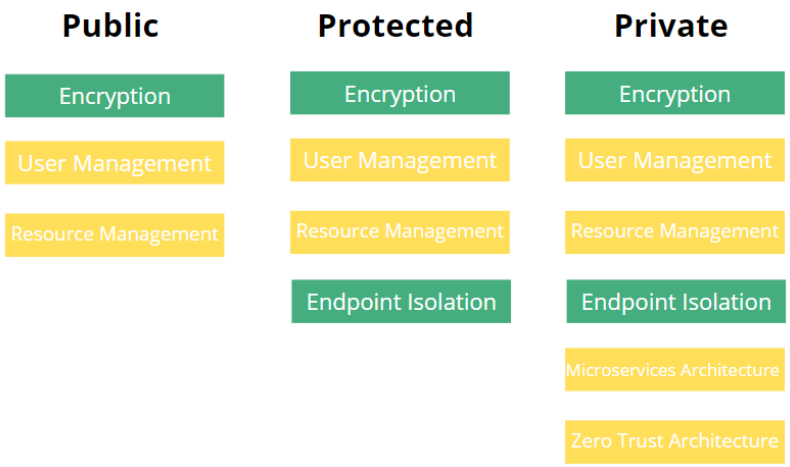
As discussed earlier, private APIs handle sensitive information that may have dire privacy and legal consequences should it be compromised, unlike protected APIs, which handle important resources. This classification includes APIs handling health or financial information. This is especially true due to the increasing interest in the protection of consumer data. In March 2021, the Virginia governor signed the Consumer Data Protection Act. The act bolstered consumers' legal rights over their data, requiring data holders to confirm whether their data is being processed and allowing them to opt out of the collection process. [43] This act also comes with steep penalties, enforced by the Attorney General, and penalties can add up to $7,500 per violation. Currently, this legislation will be in force on January 1, 2023.

This act especially has consequences for private API providers, requiring them to not only stringently protect the data but also make sure that any individual records can be securely retrieved or deleted. Not only does this act threaten providers with steep fines, but it is also indicative of a shift towards transparency with the collection of personal information. Privacy is becoming more and more of a societal concern, and the slightest breach of sensitive information may cause massive damage to a firm's credibility and bottom line.

On top of all the measures recommended in the public and private APIs, we recommend that businesses fundamentally rework their systems to fit a more secure architecture, whether it be microservices or zero trust architecture. In general, legacy systems are out of date and easily compromised, and reworking these systems from the ground up will have massive security benefits and will prevent large attacks down the road that could cost the company a large amount of money.

[43] *Cybersecurity and Information Security Newsletter*, https://legaltechcenter.net/files/sites/159/2021/11/Cyber-Newsletter-Issue-07.pdf

It should be noted that it may be difficult to implement API endpoint isolation with microservices architecture, as it requires APIs to be tightly coupled with the endpoints, and many systems would have to be set up to allow the many APIs to be isolated.



Green: Strongly Suggested; Yellow: Suggested

D. Conclusion

Every system will inherently have flaws. Even should a business implement all the given security measures and more, there is no way to ensure that no one will be able to hack into the system. However, implementing security measures greatly reduces the chance of a system being compromised, and they will also lessen the impact of the attack.

V. Evaluation of Proposal

Earlier in the paper, we mentioned an attack on a Facebook API, when hackers used Facebook developer APIs to obtain personal information of Facebook users over the span of a year. This led to a lot of information leakage, illustrating the need for active security and management practices for API systems. These proposed practices do incur a unique cost, not to mention that each of them will take time to implement, time which could have been used on developing the system. On top of that, there is not a one-size-fits-all solution. Each API system will have to be

carefully vetted to tailor to the unique security needs of the business. All these changes may increase the cost to the point that developers should question if they should release the API, though if they cannot create a secure system, they should question the security implications of its release.

Although these proposed practices do incur costs, finding that solution can help mitigate unwanted and unforeseen circumstances and most likely would have helped protect Facebook from the information leakage. Like all things, security and productivity should be a balancing act. Each practice should be carefully considered before being implemented. For example, implementing a new architecture, such as microservices architecture, will require an almost complete overhaul of the system. Developers will not only have to rework the system, but they will also need to acquaint themselves with how microservices architecture works. Nevertheless, it poses many benefits, reducing the scope of potential attacks. Like this, each security practice should be carefully considered before being incorporated into the system.

VI. Conclusion

APIs are incredibly powerful tools. Originally, each online service or system would have to be completely self-contained or have to create custom tools to take information from other sources that would have to be redefined very often. Now, APIs give online services the ability to interact with each other, creating a web of systems that can rely on each other to bring powerful new solutions to end-users. An example of this would be the PayPal API. Instead of users having to register with their credit card information every time they log into an online store, they can pay with PayPal, which utilizes its API to send a request to PayPal, which authenticates their purchase. This reduces the amount of information that online stores must manage, significantly reducing their security risks.

In the end, APIs do come with a great host of benefits. Developers should carefully consider their security implications. Knowing potential ways that their systems can be attacked is very important and being informed about them will allow developers to carefully choose the best-suited countermeasure against these attacks.