API Design Principles & Security Best Practices – Accelerate your business without compromising security

– Anil Lamba
CISA, CDPPM, practice lead Cyber security, EXL Service Inc., NJ, USA
dranillamba@outlook.com

APIs are a strategic necessity to give your business the agility, innovation and speed needed to succeed in today's business environment. However, the financial incentive associated with this agility is often tempered with the fear of undue exposure of the valuable information that these APIs expose.

Introduction

With the rise of APIs also comes the potential for more security holes, meaning coders need to understand the risk to keep corporate and customer data safe. According to Gartner, by 2022, API abuses will be the most–frequent attack vector for enterprise web applications data breaches. It is no wonder that many IT decision makers today are concerned about API security.

In 2018 itself, there have been more than a half dozen headlines of data breaches where APIs were listed as the exploited mechanism to illegally extract data. Hackers are sophisticated and are constantly looking into new ways to break down defenses and access valuable data. Taking the appropriate security measures throughout the design process can ensure that your API is used properly by those you allow to interact with your application.

In this article, we'll take a look at API security best practices, and discuss strategies for securing APIs.

1. **API Security Best Practices –** If you are building an API for public use or even only for your internal services, the following needs to be considered before augmenting any additional security layer or technology:

- **Authentication** –API authentication means determining that the client application has an identity that is allowed to use the API. API must be able to authenticate itself to the Apps which consume it. Likewise, when your API interacts with Servers, they must authenticate themselves to the API. Tokens should expire regularly to protect against replay attacks. Most enterprises will use an internal database or LDAP authentication store, though Oath may be a better option for highly public APIs.

- **Multi–factor Authentication (MFA) –** Multi–factor Authentication (MFA) requires a user to use a one–time usage token they receive after authenticating with her credentials. The User may also have a digital key which is a token that the App can validate. When the App receives the token which it validates with the MFA Provider, it proceeds to consume your API. Tokens are usually issued with an expiration period and can be revoked.

- **Authorization –** Authorization is determining the scope of interaction allowed – that is, what actions and data the authenticated application has access to when using the API. This is typically best handled by application logic for e.g. using an access control framework, such as OAuth. However, it is best to augment this functionality using an API gateway. The following two ways are also used for defining level of authorization required by the user -

- **Role–based Access Control (RBAC) –** Static assignment of roles to Users based on the organizational groups to which they belong. Groups are role and App agnostic, they are purely business–level decisions. In RBAC an App uses roles to assign degrees of access to groups of Users which the role represents.

- **Attribute–based access control (ABAC) –** Attribute–based Access Control (ABAC) aims to facilitate the dynamic determination of access control based on some sort of circumstantial information available at the time of the API call.

- **Make security the number one priority –** Developers often have a feature–driven mindset, where functionality has taken precedence over security. Unfortunately, in today's security landscape, vulnerabilities and threats lurk at every corner and have ever–growing consequences, so we have to turn this on its head.

- **Encryption –** SSL/TLS encryption is mainstream and should be used for both public and internal APIs to protect against man in the middle attacks, replay attacks, and snooping. For external APIs the web server can handle this directly or a reverse proxy can be employed. A service mesh can be used for internal APIs libraries to add automatic encryption on top of service discovery and routing.

- **Protecting your API –** Developers must ensure that the API properly validates all input from the user to prevent XSS and SQL Injection. There are many ways to protect against these types of vulnerabilities including but not limited to "cleaning user input to prevent XSS", as well as preventing SQL Injection by "preparing statements with bind variables".

- **Block Large Requests and Responses –** Some attackers may try to overwhelm the API or trigger a buffer overflow vulnerability with large requests. These may be in the form of a large JSON body or even unusually large individual JSON parameters within the request. Abnormally large response may also be indicator of data theft. Create custom rules to track and block these suspicious requests. A web application firewall can automatically detect and blocks this type of input abuse.

- **Throttling your API –** Throttling is a means of controlling or limiting a client's access to your data. There are two key API throttles that will provide you with additional control & security for your APIs:

- **IP–based throttling –** Using IP–based throttling you can restrict the number of API calls made by a particular IP address. In addition, you could ensure that your API can only be accessed by a particular set of IP addresses.

- **Rate–limit throttling –** Rate–limit throttling allows API requests to be made until a certain limit has been reached for a specific time period. By utilizing rate–limit throttling within your API you can help to ensure that the database isn't overwhelmed by one particular client who may be misusing your interface.

- **Building tests that don't represent real functional use –** Performing tests without considering how the APIs will be consumed may be quicker in the short–term. However, in doing so, you won't be testing across concerns, which could prevent you from uncovering and debugging potentially serious API issues.

- **Custom API Rules – Build your own business logic rules for security, for e.g.** a simple protection might be to identify your authentication token (in the HTTP header or in the JSON body) and require it to always

be present to block and log any unauthenticated attempts. Another example would be to enforce the Content–Type header to be what is expected for your API (e.g. application/json) or block unused or non–public HTTP methods (e.g. PUT and DELETE) to further lock down the API.

- **Testing APIs in a vacuum –** Building API tests can be a bit of a solo act, but the minute a test is in your workflow, it requires the attention of different teams in your organization. If you set up test failure notifications to go to just you, you're adding time, effort and headaches to your workflow.

- **Keys in URI –** For some use cases, implementing API keys for authentication and authorization is good enough. However, sending the key as part of the Uniform Resource Identifier (URI) can lead to the key being compromised. As explained in IETF RFC 6819, It's safer to send API keys is in the message authorization header, which is not logged by network elements. As a rule of thumb, the use of the HTTP POST method with payload carrying sensitive information is recommended.

- **Geofencing –** If your API is public, it might make sense to either block users from countries you don't do business with, or at least raise the risk score of entities that come from those countries.

- **API Fuzzing Protection –** You may have a combination of documented and undocumented features in your APIs. Attackers may attempt to map and exploit the undocumented features by iterating or fuzzing the endpoints. Install a web application firewall for application profiling and behavior tracking.

- **L7 DOS Protection –** You have protected the front–end of the API with rate–limiting, but the back–end services can still be exposed to Layer 7 denial of service. Customize a web application firewall to ensure long–running queries gets tarpitted and eventually blocked automatically.

- **Use Auditing and Logging –** Auditing should never be skipped. Logging should be systematic and independent, and resistant to log injection attacks. Auditing should be used as a tool for detecting and proactively preventing attacks.

- **Monitor add–on software carefully –** One popular use of the interfaces is to allow third parties to write add–on apps for a platform. A potential monster is such interfaces often give developers a high level of authorization rights. Hackers covet those privileges and will voraciously try to dig out such system vulnerabilities.

- **Secure the exit gateways –** Businesses need to set up another checkpoint on the way out of the network. Even If a hacker worms into the system and accesses confidential information, it has value only if the data can be moved out to their own systems. In other words, if you miss a crook on the way in, you still can thwart him on the way out.

- **Stack Trace –** Many API developers become comfortable using 200 for all success requests, 404 for all failures, 500 for some internal server errors, and, in some extreme cases, 200 with a failure message in the body, on top of a detailed stack trace. A stack trace can potentially become an information leak which attackers can exploit by submitting crafted URL requests. It's a good practice to return a "balanced" error object, with the right HTTP status code, with minimum required error message(s) and "no stack trace" during error conditions. This will improve error handling and protect API implementation details from an attacker.

- **Consider Adding Timestamp in Request –** Along with other request parameters, you may add a request timestamp as HTTP custom header in API request. The server will compare the current timestamp to the request timestamp, and only accepts the request if it is within a reasonable timeframe (1–2 minutes, perhaps). This will prevent very basic replay attacks from people who are trying to brute force your system without changing this timestamp.

2) **Design Guidelines for Developers –** DevOps has made allocating resources simpler and faster, but at the same time, being under pressure to deliver new releases ASAP, well intentioned, responsible programmers sometimes hurry and make mistakes. Here are

design guidelines for developing secured APIs: –

- **Drop Basic Authentication –** Basic Authentication is the simplest form of HTTP authentication. With each request, users submit their credentials as plain and potentially unencrypted HTTP fields. Instead, use a more secure method such as JWT or OAuth.

- **Don't ship a home–grown solution –** Never try to implement your own authentication, token generation, or password storage methods. Depending on your application's language or framework, chances are there are existing solutions with proven security.

Never expose information or URLs – Usernames, passwords, session tokens, and API keys should not appear in the URL, as this can be captured in web server logs, which makes them easily exploitable. For example, below mentioned URL exposes API key So, never use this form of security. *"https://api.domain.com/user–management/users/{id}/someAction?apiKey=abcd123456789 //Very BAD !!"*

- **Implement Max Retry and Jail safety mechanisms –** Attackers will try to authenticate using a variety of credential combinations. Setting a maximum number of retries blocks users who fail too many authentication attempts in a certain amount of time. Users who exceed the number of max retries are placed in a "jail" which prevents further login attempts from their IP address until a certain amount of time passes.

- **Encrypt Everything –** Always encrypt data before transmission and at rest. Intercepting and reading plain HTTP is trivial for an attacker located anywhere between you and your users. Encryption makes it exponentially harder for credentials and other important information to be compromised. Secure HTTP (HTTPS) encrypts data between clients and servers, preventing bad actors from reading this data.

- **Limit Requests –** One of the most common attacks on the Internet is a Denial of Service (DoS) attack, which involves sending a large number of requests to a server. The server tries to respond to each request and eventually runs out of resources. Rate limit requests to mitigate DoS attacks by throttling or blocking IP addresses.

- **Enforce HTTP Methods –** Each of your API's endpoints should have a list of valid HTTP methods such as GET, POST, PUT, and DELETE. These methods should correlate to the action the user is attempting to perform (for example, GET should always return a resource, and DELETE should always delete a resource). Any operations that don't match those methods should return 405 Method Not Allowed. This prevents users from accidentally (or intentionally) performing the wrong action by using the wrong method.

- **Validate User–Submitted Content –** Validate all data to prevent application layer attacks, such as, Cross–site scripting, Code injection, Remote Code Execution Business logic, Cross–Site Scripting (XSS)Parameter pollution attacks. You can mitigate these attacks by scrubbing user input of HTML tags, JavaScript tags, and SQL statements before processing it on the server.

- **Remove Components with Vulnerabilities –** Remove unused dependencies, unnecessary features, components, files, and documentation. Continuously check the versions of your dependencies for known security flaws. When picking new dependencies only add code from official sources over secure links. Consider monitoring for libraries and components that are unmaintained or do not create security patches for older versions.

- **Protect Sensitive Endpoints –** Make sure that all endpoints with access to sensitive data require authentication. This prevents unauthenticated users from accessing secure areas of the application and perform actions as anonymous users.

- **Avoid Using Auto–Incrementing IDs –** Auto–incrementing IDs make it trivial for attackers to guess the URL of resources they may not have access to. Instead, use universally unique identifiers (UUID) to identify resources.

- **Apply strict input validation –** Restrict parameter values to a whitelist of expected values, validate posted structures data against a formal schema language in order to restrict the content

and structure and blacklist risky content, such as SQL schema manipulation statements.

- **Use Password Hash –** Passwords must always be hashed to protect the system (or minimize the damage) even if it is compromised in some hacking attempt. There are many such hashing algorithms which can prove really effective for password security e.g. MD5, SHA, PBKDF2, bcrypt and scrypt algorithms.

- **Turn Debug Mode Off –** While it may seem obvious, make sure your application is set to production mode before deployment. Running a debug API in production could result in performance issues, unintended operations such as test endpoints and backdoors, and expose data sensitive to your organization or development team.

- **Logging & Monitoring –** Ensure all login, access control failures, and server–side input validation failures can be logged with sufficient user context to identify suspicious activity or malicious accounts and retained for sufficient time to allow delayed forensic analysis. Logging all API access is essential to assist resolving any issues, and potentially discover any patterns or excessive usage activity.

Conclusion:

API usage is rising and empowering businesses to build more dynamic applications. However, as they take advantage of these capabilities, organizations need to be aware of the potential security holes, close them timely and ensure that security is the number one priority.

An unsecured API / application endpoint can serve as a gateway to the data centre by which attackers can effectively attack the backend and there is no silver bullet when it comes to its security.

Goal of this article is to make developers understand design principles and security best practices, to protect their APIs from malicious activity. Potential threats can often be avoided by thinking critically about these practices and applying them to avoid breaches and help your business maximize its potential.

Dr Anil Lamba is a notable industry speaker, researcher, an innovator, and an influencer with proven success in spearheading Strategic Information Security Initiativesand Large-scale IT Infrastructure projects across industry verticals. He is Ph.D. Cyber Security, CISA ® and hold various other impressive industry credentials*. He has helped bring about a profound shift in cybersecurity defense. Throughout his career, Anil Lamba has parlayed his extensive background in security and a deep knowledge to help organizations build and implement strategic cybersecurity solutions.

Industry Credentials*: Ph.D. Cyber Security, M.B.A. – Strategic Project Management, CISA ®, CISSP, CDCP, CPD, CFE, PMP, Amazon Web Services (AWS) Certified Architect, AZURE Certified, Prince2, ITIL Expert, ISO 27001 Lead, Auditor, MCSE, 6 Sigma Green Belt, CEH and CCNA.

He has performed various IT Security & Governance initiatives viz. Cloud Security Audits, Secure SDLC Audits, Penetration tests, Tabletop exercises, Data Warehouse (i.e. Enterprise Data Lake & Grid) Audits, Regulatory as well as Industry standard assessments and Pre-Mergers & Acquisitions assessment projects across industry verticals. Anil has gained his expert reputation by staying at the leading edge of security research and mentoring security teams, developers and audit experts across various industries. Anil has strived to change the way organizations approach security, placing an emphasis on making informed decisions regarding products and services to best protect the organization, employees, and customers. He has spent much of his career meeting with CISOs and CIOs to advise and educate them on threats, required policies, processes, and expertise.

Anil has given more than 29 researches and 6 conference papers to information & cyber security world. His career has focused on high impact research in Cyber Securityas well as its applied practice in real world and bringing it to everyone's use and education for free. His researches has educated the industry and the general public on the evolving threats to our interconnected world. He serves on the board member, reviewer& editor of several computer science related research journals.

Per Anil, "Effective cyber security requires science, engineering, business, policy and people skills. My goal has been and is to instill this culture in the discipline and provide leadership in all elements."

By Career, He is Practice Lead – Cyber Security for a Consulting company named EXL. As a security professional, Anil has spent more than 15 years in cybersecurity operations leadership and influencing policy level decisions in multiple client organizations and helping clients understand cyber security challenges,. Some of these organizations included some of the largest pharmaceutical, Fortune 500 banking & financial and telecommunications companies in the U.S.

Anil leverages his skills and pervasive industry experience to help customers understand risks in their systems and develops programs to mitigate those risks. He has automated various cyber security, privacy and data security audit programs which has in-turn saved a lot of time for our clients as well as improved the practice. He has been a trusted advisor to the legal team to ensure that the right security clauses are built into supplier and customer contracts. He has also enabled engineering development teams of our clients with secure code practices and have performed reviews and quality assurance tests for functional and security verification.

He has always volunteered himself to act as a cybersecurity adviser for undergraduate college students conducting independent researches. He has presented research and other topics at many conferences over the years. His published researches and conference papers since 2014 has led to many thought provoking examples for augmenting better security. His Ph.D. thesis on Cyber Crime became a famous literature for inspiring cyber security and law students.

dranillamba@outlook.com