An Attackers Guide to Hiding Your Back-end APIs

Mathias Gam-Pedersen
ReTest Security
Copenhagen, Denmark
mgp@retest.dk

***Abstract*—When it is possible for an attacker to obtain a lot of detailed information about a web application, it speeds up the time needed to find any potential vulnerabilities in said application. In this paper, we discuss how you can "hide" information about your back-end application through various methods, to hinder an attacker to be able to gather useful information through reconnaissance and to reduce your publicly accessible attack surface. The methods described are proxying, having the front-end fetch information and proxy requests, and using serverless functions to pass requests to the back-end. We, furthermore, discuss some of the security implications of opting for one of these solutions, along with added security measures that could further increase the security of the back-end APIs.**

***Index Terms*—Attack Surface Reduction, Security Engineering, IT Reconnaissance**

I. Introduction

If it is possible to obtain a lot of information about a system through reconnaissance, it makes the job of attacking said system easier. In this paper, we will discuss approaches that can be used to reduce the amount of information possible to obtain from a back-end. Furthermore, we will cover the motivation and security considerations related to using these approaches.

II. Background

In this section, we will provide a high-level background to the primary concepts of this paper. First, we will cover one of the common architectural patterns of smaller web applications. Next, we will give a brief introduction to attacker methodologies, focusing on the initial stages. Finally, we will briefly discuss how the back-end could be isolated.

A. Common Web Application Design Pattern

In this section we will provide a brief introduction to the Model/View/Controller *(MVC)* design pattern [1], used for many smaller web applications. In Figure 1 we have visualized the flow requests follow through a typical MVC application and the user.

Most applications store and handle data, which we refer to as the *model*. Often data is stored in a database. To provide the functionality to perform create, read, update, and delete *CRUD* operations on the model, and various other functionality, most applications have a back-end application, which we refer to as the *controller*. To present the user with a visual view of the data, most applications contain a front-end graphical application, referred to as the *view*.

For this paper, we only consider web-based MVC applications. Therefore, we define a controller to be a back-end API and a view to being a front-end web application.

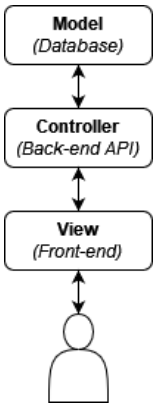

Fig. 1. Visualization of the Model/View/Controller *(MVC)* design pattern.

B. Attacker Methodologies

Most attacker methodologies and generalized approaches, such as MITRE ATT&CK® [2], the Cyber Kill Chain® [3], and OWASP Web Security Testing Guide *(WSTG)* [4], follow the same generalized approach: Perform reconnaissance to gather information about the target system, discover vulnerabilities, exploit a vulnerability, and finally using the vulnerable system in an unintended manner.

The earlier system owners can stop or slow down the progress of an attacker, the better. If it is not possible to perform detailed reconnaissance of a system, then the task of attacking a system typically becomes much more time-consuming. The reason is that if an attacker does not know which technologies are used for a given system, it would be required to test for potential vulnerabilities specific to most common technologies and programming languages. This would mean that an attacker has to send a larger amount of requests to the systems with various payloads, increasing the chance of the various detection systems noticing that the system is being attacked. Instead, if it would be possible to determine the exact technology used for a given service, it narrows down the potential attacks, thereby reducing the number of payloads needed to test and the time investment.

C. Isolation of Your Back-end APIs

We define isolation of the back-end APIs as when it is not possible to interact directly with them. Instead, only a few systems can communicate with the APIs. The most secure approach to achieving this would be via network isolation, where no unauthorized entity can interact with the back-end service(s). Following this approach, it would not be possible to interact directly with any back-end service.

In some scenarios, isolation via network isolation might not be possible or feasible. For such situations, a simple approach is to require an API key for all API endpoints in the back-end, which is only known to the authorized services. Although this does not remove the opportunity to interact directly with the system, it removes the option of being able to directly access any functionality of the system.

III. Hiding Your Back-end APIs

In this section, we will discuss three different approaches, which could be utilized to provide front-end user access to an isolated back-end, without disclosing any information about the core back-end application(s). Before discussing the various approaches we will first cover the motivation behind choosing to use one of the approaches.

A. Motivation

The primary motivation for hiding back-end APIs is to reduce the attack surface of an application. By limiting the number of services that are publicly available you reduce the chance of any one system being compromised by a potential zero-day exploit. Furthermore, it allows companies with limited time available for maintaining security a way of prioritizing which systems are most important to keep updated and secure.

In addition to attack surface reduction, hiding information about back-end APIs greatly reduces the amount of information that is possible for an attacker to obtain through reconnaissance. The consequence of this is that the attacker has to spend more time to find any potential vulnerabilities in the back-end, and would have to attempt many different exploits which would not work on the technology a given back-end is using. If isolation is coupled with the detection of malicious activity, this would likely help increase the chance of malicious activity being detected.

B. Approach A: Proxy

One approach to hiding APIs is through the use of a proxy, which removes any information from the headers which is not strictly needed. All requests between the front-end and back-end would go through this proxy.

Following this approach, the back-end functionality would still be accessible in the same capacity as it would be if exposed directly to the internet. The main benefit is the reduction of information available, and it may make it easier to implement other security checks in connection with the proxy.

C. Approach B: Server-side Fetching

Using this approach the application would only have the front-end server publicly available. Whenever the application needs to fetch data from the back-end, the front-end server performs this action before sending the website data to the user. For the functionality that the user needs to trigger, which does not involve requesting a new web resource, the front-end server would implement simple API endpoints that pass the request to the back-end server.

By utilizing this approach it may be possible to limit access for some endpoints in such a manner that the users will not be able to provide any input which might lead to vulnerabilities. If the API in the front-end only sends back limited information about the status of the request to the back-end, it would remove the ability for an attacker to use error messages to gather useful information about the back-end.

D. Approach C: Serverless Functions as a "Proxy"

Another approach is to make use of serverless technology to have all requests to a back-end go through serverless functions, which then send the request to the back-end. The benefit of this approach is that a potential bottleneck is removed in terms of the number of active users the system used to hide the back-end can handle, as long as the serverless functions are permitted to scale enough to meet the demand. Furthermore, since the functions are short-lived and single-request there is no ability to interfere or read other requests, removing the ability to man-in-the-middle upon a compromise of the "hiding mechanism".

IV. Security Considerations

In this section, we will cover some of the security considerations which need to be considered before choosing to hide your back-end APIs. Furthermore, we will provide suggestions on how the security could be further improved, if You choose to hide Your back-end APIs.

An important thing to consider related to the various approaches is related to what the impact would be of a total compromise of the "hiding mechanism". If it is possible to compromise the system(s) running approach A and B, it would likely lead to the ability to see all traffic flowing through the system. Furthermore, it would allow for an attacker to perform man-in-the-middle attacks, where traffic is manipulated in transit to achieve malicious actions. However, the impact of a full compromise of the said system has to be weighed against the impact of a full compromise of a back-end server. Typically these servers contain credentials to be able to gain full access to databases and other critical services, which could prove more critical to have compromised than the impact of a man-in-the-middle attack.

A. Further Improving Security

To further improve the security of the web application, we would recommend implementing rudimentary input validation and input cleaning into the ”hiding mechanism”. Implementing this would help remove most malicious input before the input being handled by the back-end.

The impact of a compromise of the ”hiding mechanism” can be greatly reduced by limiting the mechanisms’ ability to manipulate data. This can be achieved by sending the data as a signed JSON Web Token *JWT*, which would provide integrity to the request through the mechanism. If confidentiality is important, JSON Web Encryption *(JWE)* could be used to send the data in an encrypted manner through the mechanism.

To further increase security it would be recommended to include various other security services as a part of the chosen ”hiding mechanism”. One of the highest impact mechanisms is to use a trusted web application firewall *(WAF)*, which would analyze all traffic to spot and block any request which is deemed to be malicious. Adding logging mechanisms would also help to be able to analyze the traffic of the applications, which can be useful in finding bugs or a potential incidence response scenario.

V. Conclusion

By adding a ”hiding mechanism” between the front-end and back-end of a web application the system owners will be able to greatly reduce the amount of information that is possible to obtain about the front-end. Furthermore, by opting to use a ”hiding mechanism” it might be possible to limit access to some API endpoints, which reduces the available attack surface of the application. Combined this would require an attacker to invest more time into attempting to find vulnerabilities in the application, along with increasing the number of malicious requests said the attacker has to send, increasing the chance of attacks being detected.

References

[1] Reenskaug, Trygve Mikjel H. ”The original MVC reports.” (1979).

[2] Strom, Blake E., et al. ”MITRE ATT&CK®: Design and Philosophy.” (2020).

[3] Hutchins, Eric M., Michael J. Cloppert, and Rohan M. Amin. ”Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains.” Leading Issues in Information Warfare & Security Research 1.1 (2011)

[4] Saad, Eric, et al. ”OWASP Web Security Testing Guide” (2022)