Security Lessons from Building a Back-End Service for Data Collection

Halmon Lui
Department of Computer Engineering
Wentworth Institute of Technology
Boston, MA, USA
luih@wit.edu

Chen-Hsiang Yu
Department of Computer Science and Networking
Wentworth Institute of Technology
Boston, MA, USA
yuj6@wit.edu

Abstract—Both malware implants and Internet of Things (IoT) attacks have grown rapidly. The back-end services face different kinds of risks. It is important to enforce proper preventative and security assessments to help mitigate those risks. In this paper, we want to share security lessons learned from designing a back-end service for data collection used in a statistics class in higher education. We organize our learned security lessons into three categories and they are easily targeted by an attacker. In this paper, we have three contributions: (1) the demonstration of building a secure and lightweight back-end service with Node.js, Express, MongoDB and Nginx, (2) the discussion of the vulnerabilities of a web application from a back-end service perspective, and (3) a presentation of security measurement to mitigate or prevent attacks on those vulnerabilities. The current findings are promising and are worth further exploration to help back-end developers create an efficient and secure web service.

Keywords—Back-end service, web application, application security, Node.js, MongoDB, Nginx.

I. INTRODUCTION

While technology becomes relevant in every field, malicious usage occurs more frequently than expected. According to the 2018 Symantec Internet Security Threat Report, malware implants and attacks on IoT have grown by 200% and 600% respectively in 2018 [3]. The 2018 Edgescan report also indicates that 20% of new vulnerabilities were considered high or critical [6]. All of these findings are concerning, and they must be addressed to protect valuable data stored in the database as well as the back-end server itself.

To support hands-on experience for a statistics class in higher education, we designed a system to collect real-time data from the users and reported our learned lessons in this paper. The real-time data are stored securely into the database. Because the real-time data sent by the students are related to their personal physiological information, the back-end service needs to be fast and efficient. In addition, since the system uses open source software to construct the back-end service, it is important to protect data from outside attacks.

The back-end service processes incoming traffic via Nginx [12]. From there, the traffic will be rerouted to the Node.js [14] server which will communicate with the MongoDB database [10]. Each of the three components, i.e. Nginx, Node.js and MongoDB, is highly scalable [4], [10] and widely used. However, because of security concerns, in addition to using existing technologies, security enhancement for such system was investigated.

In this paper, the security findings are organized into three categories: system software, connection forwarding, and data storage. For system software, Node.js and Express were used to implement a lightweight web server with RESTful APIs support. The connection forwarding is done through the use of Nginx to direct traffic for the server. MongoDB, in accordance with Node.js, is used to allow real-time data collection for data storage. Each component contains its own vulnerabilities from low to high risk. The vulnerability findings of each component are not a complete list of all weaknesses, but they are the ones we currently discovered and can be preventable. Each risk and the method to mitigate it are described in detail.

II. RELATED WORK

Back-end services, such as RESTful APIs, have played important roles to support different applications, but the use of technologies keep evolving. Since Node.js, MongoDB, and Nginx are rising tools, the efforts of improving security of these tools are increasing. For instance, Pfretzschner et al. [2] investigated JavaScript and Node.js dependency-based attacks on abusing global variables, patching, and the sharing of cache of the loaded modules.

With Node.js being a JavaScript runtime environment for the Web back-end development, there exists vulnerabilities which are shown in other researches [8], [7]. First, Davis et al. [8] studied the weaknesses of the event-driven architecture in Node.js. Due to the design of Node.js, the single threaded event-handling system is vulnerable to a type of denial of service called an Event Handler-Poisoning (EHP) attack. This attack would cause handlers to block indefinitely but Davis et al. designed steps to prevent this attack from happening. They extended this work to create Node.fz, [7] a testing aid for the server-sided event-driven applications which exposes known bugs such as the EHP vulnerability.

In addition to the weaknesses in Node.js, past researches also identified vulnerabilities in NoSQL databases [5], [9], [1]. For example, Hasija et al. [5] introduced compression and security for MongoDB without affecting its efficiency. The usage of authorization, granularity and encryption are highly encouraged to increase the safety of MongoDB. Hasija et al.’s work is furthered by Dissanayaka et al. [1] in their review of

MongoDB in regard to the Health Insurance Portability and Accountability Act (HIPAA) regulations. These regulations focus on four areas in data security: authentication, authorization, encryption, and auditing. System security for healthcare data must be compliant to those areas. On the other hand, Li et al. [9] broadened their spectrum to all NoSQL databases. Their goals included teaching real-world NoSQL database security threat analysis and defenses. Many modern information systems are targeted by cyber-attacks and the security awareness needs to be spread.

Countless volunteers in the Open Web Application Security Project (OWASP) [15] have assisted in adding different vulnerabilities to web applications. These include the Zed Attack Proxy that automatically finds security vulnerabilities in web applications as one is developing and testing the applications. Another notable resource is the Web Testing Environment which is a collection of easy-to-use application security tools and documentation in multiple formats. Both of them are good resources for the developers to reference for security check in their work.

III. BACK-END SERVICE FOR DATA COLLECTION

To collect the user’s vital sign in real time for statistical analysis, a back-end service that uses state-of-the-art back-end server techniques was designed and implemented. The service runs in the Node.js JavaScript run-time environment, along with Express.js and the database engine, MongoDB. Nginx, a forwarding proxy, sits in front of the entire service. A Python script applies real-time data analysis with the information derived from the users via the Node.js server. The information is then returned to the MongoDB database and the cycle repeats.

When the users send HTTP requests to the back-end service, which provides RESTful APIs, the reverse proxy (Nginx) handles the request and redirects it to the correct portion of the API in order to communicate with the database engine, MongoDB. By removing direct connections from the user to the database, the system is secure from tampering (Fig. 1).

![Diagram of the back-end service]

**Fig. 1 The architecture of the back-end service**

IV. SECURITY ISSUES AND SOLUTIONS

Because the back-end service processes the users’ personal vital sign data in real-time, system security is a priority. Although state-of-the-art technologies are already used, many security issues were discovered, and these issues were accounted for. In this section, security issues and solutions are summarized in three categories.

A. Connection Forwarding with Reverse Proxy Server

Nginx, known for connection forwarding, contains vulnerabilities which were prevalent in older versions. Some of these vulnerabilities included: integer overflow in the range filter, NULL pointer dereference while writing client request body, invalid pointer dereference in resolver, etc. These vulnerabilities have been identified and patched. In our back-end service design, the most updated version of Nginx was used to avoid these issues [13]. There does not exist any known vulnerability in the current version of Nginx, except a Denial of Service (DoS) or a Distributed Denial of Service (DDoS) attack.

B. Vulnerabilities in JavaScript Run-Time Environment

Node.js is a JavaScript run-time environment. The back-end service uses Express, a JavaScript framework, to implement a software server running in Node.js. The package manager for Node.js, named npm, generates security flaws as well. For example, when one uses npm to install a Node.js package, a large amount of code written by someone else is inserted into the program. Unfortunately, there is currently no security check for these added lines. Therefore, it is easy to take advantage of this design to insert malicious codes. For example, monkey-patching is the dynamic modification of classes and functions at run-time which can be placed inside of an installed dependency if left unchecked. Unless one has read through the dependency entirely, which is almost impossible for most users, there is no way to determine whether a function has been monkey-patched or not (Fig. 2). Therefore, an attacker can patch a function to maliciously tamper with the user’s program [2].

```javascript
function ClassName() {
    
    ClassName.prototype.someFunction = function () {
        //Here is the original implementation
    };

    function MonkeyPatchClassName() {
        var originalFunction = ClassName.prototype.someFunction;
        
        ClassName.prototype.someFunction = function () {
            //Here the monkey-patched version will be used instead
            //originalFunction will be invoked
        };
    }
}
```

**Fig. 2 Example of a basic monkey-patch. The function someFunction of class ClassName is monkey-patched from the function MonkeyPatchClassName**

Similar to Nginx, using the most updated package would mitigate most existing vulnerabilities, but new updates may also contain flaws which is why the users should constantly check new packages. Examples of such vulnerabilities include directory traversal, unchecked buffers which would
compromise an entire system, and recursive equations which can be used to break the server. Since most people are pressed for time and are unable to go through all of the code in a package, Snyk [17] can be used to detect and fix common vulnerabilities, such as content encoding in HTML.

C. DoS and DDoS Mitigation

In addition to the code being vulnerable, the Node.js server itself can be susceptible to DoS and DDoS attacks. Although Node.js has an asynchronous event-driven architecture, enough requests and connections will still crash the server. Therefore, a proxy server such as Nginx can act as a shield in front of the Node.js server. It takes requests without an increase in resource usage [10]. Not only does it absorb the initial shock from a DDoS attack, certain security measures are able to be taken to shield the server further. For instance, a proxy server can limit the number of incoming requests from a client, limit the amount of connections per IP address, close slow connections, blacklist or whitelist IP addresses, etc. [11]  

limit_req_zone $binary_remote_addr zone:one:1m rate=304/m;

server {
    # ... 
    location /homepage.html {
        limit_req zone:one; # Limit rates 30 per minute
        limit_conn addr 20; # Limit connections to 20
        client_body_timeout 4s; # Close slow connections
        client_header_timeout 4s; # Close slow connections
        allow 123.456.7.8/90; # Whitelist IP address
        deny all; # Block all other IP addresses
    # ... 
    }
}

Fig. 3 Example of available Nginx security settings

D. Vulnerabilities in Database Engine

The main security concerns of a database engine, such as MongoDB, are the attacks challenging the data integrity, such as NoSQL injections. To fend off such attacks, one could create a password for the server, but passwords can be brute forced. Instead, we found that a better way to keep the database engine safe is to make the server unavailable to the public and only accessible through the JavaScript client. In addition, to provide a second level of protection, it is safer to run the MongoDB on a different port than using the default port (Fig. 4), such that casual and beginner attackers doing a quick scan will not discover a possible vulnerability.

net:
port: 27017

Fig. 4 Default port listed inside of /etc/mongod.conf which is highly recommended to be changed

Additional security measures are available from standard MongoDB and should be utilized to maximize the server security [6], [16]. One of these security measures is the use of authentication and individual user authorization. This prevents illegal access from the users who do not have proper permissions. Another measure which should be utilized is network traffic encryption. If an attacker sets up a man-in-the-middle to intercept data or even a packet sniffer, the discovered data would be encrypted and extremely difficult to decrypt. Not only can traffic be encrypted, but the REST data can also be encrypted to make the decrypting process difficult for the attackers.

V. DISCUSSION

After being in the Node.js community, we found that most developers seem to be aware that there are security risks, but they are unsure of the ways to improve the security. Our back-end service uses different open source components, including Nginx, Node.js and MongoDB.

Nginx can be a great benefit to any system for its reverse proxying ability and its event-driven architecture. Not only does it perform efficiently, but it also adds a protective shield in front of the actual server. The RESTful APIs of the back-end service are created by using Node.js which contains several vulnerabilities, but most of them are preventable by the developer. Because the attacks targeted at Node.js can come from required dependencies, developers should always ensure that the used packages are secure.

The third component, MongoDB, stores all vital sign data from the users. A balance must be kept between security and efficiency. If the defensive measures are significantly slowing data writes or reads, other methods should be explored and adopted to enhance the security.

Cybersecurity has been gaining attention nowadays, but it is still in an early stage for higher education to be prepared to teach related topics in computing domains. Lessons learned and presented in this paper are suitable for instructors to use as an example in a classroom setting. The system architecture presented in this paper is light-weight and extremely suitable for laboratory exercises of different computing classes, such as Web App Development, Network Programming, Network Security, Digital Forensics, etc. We look forward to extending this project to support different security related classes in higher education.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, a light-weight back-end service with a goal of gathering the users’ physiological data for a statistics class in a higher education environment was presented. This system will rapidly and securely collect real-time data using limited free resources. Through a careful lens, the common vulnerabilities have been tackled and the service is safe from most attacks. As cybersecurity strengthens, attackers can find new weaknesses which need to be dealt with. Although the service may be secure at the moment, it is imperative for the system administrators to constantly peruse through the components in order to ensure maximum security.

There are three main contributions presented in this paper. First, we presented our back-end service that is secure and light-weight to collect real-time data from the users. The
service was created by using state-of-the-art technologies. Second, we summarized the vulnerabilities learned and found. Third, we discussed the security measurements to mitigate or prevent attacks from identified vulnerabilities.

For the future work, the back-end service design can be researched further. There may be newer methods available to speed up or make each individual component secure. For example, the Nginx component is still extremely new for many developers; there is lack of research in this component from security perspective. There are still many paths open for the exploration of Nginx. In addition, although npm provides a convenient way to install packages for Node.js application, it also generates security flaws, such as Monkey-Patching presented in Fig. 2. There is a need to design an automatic scanning for preventing security flaws in npm packages.

REFERENCES


