Chaining JSON Blocks for Secure Transmission of Mobile Data Sequences

John Yoon
1Math/CS Dept, Mercy College, New York, U.S.A.

Abstract - JavaScript Object Notation (JSON) has evolved as a de-facto standard data format between servers and clients. As mobile applications send/receive large volumes of data to/from remote data sources over extended periods of time, the security of JSON data becomes compromised and vulnerable to cyberattacks. Several approaches have been proposed, e.g., JSON files encryption. However, typical JSON encryption techniques alone may not be satisfactory in part because they can’t handle long-duration JSON data and because once sniffed, the entire file is compromised. This paper proposes a cryptographic hash technique by chaining JSON blocks to ensure JSON data transmission for mobile devices. The contribution of this paper explores 1) the integrity of JSON block sequence transmission; and 2) detection of JSON block data spoofing.

Keywords: JSON, JSON Spoofing, Secure Data Transmission, Swift Decodable, Cryptographic Hash

1 Introduction

JavaScript Object Notation (JSON) is a de-facto standard data format [1] for a platform-independent web API. Mobile applications can receive data in a JSON format from remote data sources with additional encryption for data sensitivity. For example, suppose that a mobile application sends a bank account and password, requesting the balance state of the bank account. When a bank balance statement is transmitted to a mobile application, the bank account may be encrypted or only show partial data (e.g., the last 4 digits). Together, the JSON data consists of both sensitive data and plaintext data.

Typically, sensitive data elements are encrypted and embedded in a JSON data format for transmission. This JSON data may be sniffed and compromised, or it may be spoofed to be modified and resent. Recall the example above, when a bank account and password is transmitted, there’s a chance the data may be sniffed and compromised. Or worse, the request for the bank balance statement may be spoofed and modified to wire-transfer to another bank account.

As 5G mobile technology becomes more available, the size of JSON data will be large. This means that JSON data may contain multiple sensitive data elements or that a single transaction consists of one or more JSON data, each of which may contain sensitive elements. Such complex JSON data can be transmitted by smaller blocks at multiple times. For example, in a bank money transfer, the entire transaction is a complex JSON data, which starts with a bank account connected from a mobile device, an authentication from a bank, money transfer request from the mobile device, authorization of money transfer from the bank, money transfer from bank to bank, and the confirmation ticket to the mobile device. Along the trail of a bank money transfer, each data block can be formed in a JSON format and can be transmitted between a bank server and a mobile device, or across multiple bank servers. Similar examples can be found in many applications including sharing property deed documents, legal dossiers, loan applications, or train or theater tickets to be sent to mobile devices, etc.

In a sequence of JSON blocks, each such block carries significantly sensitive data. However, a sequence of JSON blocks has more vulnerabilities since each JSON block may be sniffed or spoofed independently and separately from the rest. Typical cryptographical approaches [2] do not comprehensively prevent such a sequence of JSON blocks from any cyberattacks or cyber threats. For example, a home refinance application contains multiple documents including current loan documents, and it should flow to multiple agencies including underwriters and attorneys. Since there are multiple steps along the hierarchy where the JSON data flows, not only are sensitive data elements encrypted to protect data from being compromise, but the encrypted elements are verified between a parent node and a child node along the hierarchy.

This paper proposes a novel technique to chain a JSON block sequence to ensure the secure transmission of JSON data when data spoofing takes place. A technique that this paper employs is a hash mechanism [3] in addition to the typical cryptographical approaches. The integrity of the current JSON block is determined by the hash code of the current and the chain of the previous blocks. The contribution of this paper includes 1) the integrity of JSON block sequence transmission; 2) detection of JSON block data spoofing.

The rest of this paper is organized as follows: Section 2 describes preliminaries and related publications; Section 3 introduces the JSON model and defines JSON block with its encodability and decodability; Section 4 describes a threat model that may take place on JSON block sequences; Section
5 describes the defender’s model where two ways of JSON block chaining are discussed; and Section 6 will conclude our findings.

2 Background and Related Work

One of the most popular data formats used in REST (Representational State Transfer) architecture is JSON (JavaScript Object Notation). JSON, in a text format, is a lightweight data-interchange format. It is a collection of key/value pairs (similar to dictionaries), and the collection can be realized as an array of dictionaries [4].

EXAMPLE 2.1: An example for home refinance applications can be written in a JSON format as follows, according to OWASP JSON Web Tokens (JWT) [5,6].

Header:

```json
{
   "type": "loan",
   "algo": "HS256"
}
```

Payload:

```json
[
   {
      "applicant": {
         "name": "John",
         "ssn": 1234,
         "addr": "main street, new york",
         "co-name": "Mary",
         "co-ssn": 9876
      },
      "property": {
         "addr": "wall street, new york",
         "price": 123
      },
      "requested loan amount": 567,
      "lender": {
         "bank": "b-bank",
         "loan officer id": 234,
         "loan manager id": 345,
         "underwriter id": 998
      },
      "documents": [
         {
            "w2": 333,
            "net-income": 4455
         },
         {
            "tax year": 2018,
            "tax report": 3344
         },
         {
            "bank": "c-bank",
            "balance amount": 32
         },
         {
            "bank": "d-bank",
            "balance amount": 321
         }
      ]
   }
]
```

Signature:

```plaintext
HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), KEY )
```

The payload of this JSON data consists of key names: applicant, property, requested amount, lender, and documents at the top level of the hierarchy.

Fig.1 shows a typical architecture for mobile apps to access data from remote web servers where there is a data source. As a mobile app needs data to be transmitted from a remote source, the app can use the web method “POST” to access a webserver, e.g., a PHP program. Then, the web server (or a PHP program) can access a backend database, e.g., MySQL, to execute an SQL expression. The result returned from the database server is converted into a JSON data by a PHP program at the web server. The mobile app receives the JSON data to provide to a mobile user [7,8].

One of the important security issues for mobile devices is the initial authentication. The properties of applications running on mobile devices are considered for authentication [9,10] and depending on the mobile device, there may be different properties to use for authentication. For example, in XCode Swift [11], an UI graphic component, button, can be set to enable or disable for user’s credentials.

```swift
@IBOutlet var btnDBAccess: UIButton!
@IBAction func btnDBAccess(_ sender: Any) {

    if idPassVerifier(id, password) {
        btnDBAccess.isEnabled = true
    } else {
        btnDBAccess.isEnabled = false
    }
}
```

The above swift code shows that an UI button is define as btnDBAccess. This button is set to be clickable if the user’s id and password are matched; otherwise, it should be disabled. In this way, the user’s access can be effectively controlled.

Another approach is to encrypt JSON data. According to JSON Web Token (JWT) [5], JSON data has three elements: header, payload and signature [6]. The signature in a JSON data is a hash code of the triplet (header, payload, key). This approach ensures the integrity of the JSON payload as one entity. One time hashing may not work if a very big payload is transmitted or if the transaction is transmitted over a long duration [12], for example of a loan-review process that includes multiple agents. Our approach is to produce a sequence of hash codes of the triplet (header, payload_n, the previous hash code), where the previous hash code is in turn a hash code of the previous triplet (header, payload_n-1, the previous hash code). In this way, hash codes are chained to
ensure the current payload based on the previous.

A mobile operating system is proposed to integrate data contents within mobile applications [13]. The mobile OS claims there is a secure integration. However, the drawback is low flexibility and low dynamic connections to the world level data sources. Our approach preserves the high flexibility keeping active connection with the world level data sources.

To preserve the Bitcoin ledger’s integrity and detect JSON data spoofing, a technique of using hash functions and hash tables is employed [14,15].

3 JSON Model

This section introduces JSON trees (or JSON graphs), and JSON blocks.

3.1 JSON Trees

As JSON data (particularly, JSON payload) is hierarchically organized, it can be represented in a directed, edge-labeled tree or a graph [16,17]. A JSON tree employed in this paper is a cyclic directed graph. An edge links two nodes. A node with an incoming edge is a parent node to the node from which the edge goes out, while a node with an outgoing edge is a child node to the node for which the edge is incoming.

EXAMPLE 3.1: Recall EXAMPLE 2.1. The JSON payload is represented in the following JSON tree:

```
In the hierarchy shown above, the parent of the node (loan officer id) is (lender), which is a child node to the node (home loan refinance application).
```

3.2 JSON Block

JSON data block is a sub-tree of the JSON tree, a minimum unit containing zero or more sensitive data elements, which can be transmitted securely one at a time. How can JSON data block be transmitted securely? Our answer will be described in Section 5. Instead, this section provides an overview of how a JSON data block is formed.

JSON data blocks are generated by the following steps:

1) A JSON data block begins by parsing the data from the beginning of the message or from the first data element after the end of the previous data block.
2) As a JSON data is parsed, when a sensitive data element is encountered, the block ends to include all child nodes of its parent node and stop its parsing.
3) The next data block begins after the end of the current parent node.
4) The above steps continue until the end of the given JSON data.

EXAMPLE 3.2: In EXAMPLE 2.1 and 3.1, the underlined key names indicate sensitive data. With this, JSON data blocks are obtained by following the steps below:

1) Parsing from the beginning, the first sensitive data element, “ssn”, is encountered. The first JSON data block includes all the child nodes of its parent node, which is “applicant,” and stop its parsing. So, the first JSON data block will be applicant.json, as shown below:

```
applicant.json
```

```
[
  "applicant": {
    "name": "John",
    "ssn": 1234,
    "addr": "main street, new york",
    "co-name": "Mary",
    "co-ssn": 9876
  }]
```

2) The next parsing begins after the node “applicant”, which means it begins from the node “property.” When a sensitive data element is encountered, which is “underwriter_id”, the parsing ends at the end of its parent node “lender.” So, the second JSON data block will be property_lender.json as shown below:

```
property_lender.json
```

```
"property": {
  "addr": "wall street, new york",
  "price": 123
},
"requested loan amount": 567,
"lender": {
  "bank": "b-bank",
  "loan officer id": 234,
  "loan manager id": 345,
  "underwriter id": 998
}
```

3) In the same way, finally JSON data block documents.json can be obtained.
4 Threat Model

In our threat model, the attacker’s goal is to sniff JSON data to exfiltrate messages while remaining stealthy on both sides, or spoof JSON data to modify messages while misinforming either the server or the client. To this end, we make the following assumptions.

- The attacker can sniff entire or partial JSON data, in which sensitive data may be breached.
- The attacker can request for JSON data with a key that is illegally obtained.
- The attacker can spoof entire or partial JSON data to modify data to continue to transmit.

Two sample threats, sniffing and spoofing attacks may take place over JSON data, which is illustrated in Fig.2. This paper focuses on detection of spoofing attacks more. The sniffing attacks can be detected or prevented by minor modification of the approach proposed in this paper.

5 Defender’s Model

This paper does not describe how to prevent JSON data from illegal exfiltration, but it aims at its detection. To protect JSON data transmission to mobile applications, the defender deploys JSON protectors on both server sided and client side. Assuming that an asymmetric key is shared between the server and client sides (e.g., key sharing in Diffie-Hellman cryptography [18]), the sender side sends one JSON data block at a time, and the receiver side receives it to verify the integrity of JSON data. The hash code received from the sender side should be the same as the hash code generated at the receiver side. In order to acknowledge, the receiver side sends a new hash code, which is generated from the message together with the receiver’s key (or MAC id), to the sender side. Then, the sender side checks and verifies the integrity of the data transmission loop between the sender and the receiver. This continues until the final block is completely transmitted.

Both sender and receiver sides have the keys, s1 and r1, respectively. In order to send JSON data blocks, jsb1, jsb2, ..., jsb_n to the receiver side, do the following:

1) At the sender side, let h1 be hash(jsb1+s1).

2) The sender side sends jsb1, h1 to the receiver side.

3) At the receiver side, let h2 be hash("ACK"+r1). Check hash(jsb1+hash(pair(r1))) = h1.

4) The receiver side acknowledge by sending h2 to the sender.

5) At sender side, let h3 be hash(jsb2+h1). Check hash("ACK"+pair(s1)) = h2.

6) Sender side sends jsb2, h3 to the receiver side.

7) Continue 3) to 6) until there are no more jsb left to send.

This can be formally implemented as the algorithms below: sendToServer() and sendToClient(). Two algorithms are calling each other. Let dd be the message to send to a client. Since dd can be decomposed into a series of JSON data blocks, it is iterable [19]. It means that the function next() returns the next block from dd, just for the simplicity purpose in the following algorithms. Let KEY be a symmetric key shared with the server:
Algorithm 1: sendToServer (data, chh)
// Assume dd, which is iterable

Compute shh = hash(data, KEY)

IF chh == shh {
    // data integrity is verified
data = next(dd)
    // get the next block
    Compute shh = hash(data, KEY)
    sendToClient (data, shh)
} ELSE {
    // data is compromised on the way from the client
}

Algorithm 2: sendToClient (data, shh)
// Let data be a control message, e.g., “ACK” for acknowledge:

Compute chh = hash(data, KEY)

IF shh == chh {
    // data integrity is verified
    Compute chh = hash("ACK", KEY)
    sendToServer("ACK", chh)
} ELSE {
    // data is compromised on the way from the client
}

As shown above, the above two algorithms are invoked each other and therefore, they are together are recursive. This recursion terminates as the JSON data block is sent to clients by taking the next block at each time of recursion being called.

With this procedure in mind, another issue is the data format for those data needs to be transmitted. The data transmitted from a mobile device to a server is based on URL format [20], while the data transmitted from a server to a mobile device is based on JSON.

The second block of the JSON data shipped from the sever is in Fig.3 (a), while the request from a mobile app is coded, for example, in swift in Fig.3(b).

EXAMPLE 4.1: Recall EXAMPLE 3.2. Consider Algorithms listed above. Fig.4 illustrates how a web server app, AppJSONsender.php, sends JSON data, and a mobile app, AppJSONreceiver.swift, receives it. Assume that the mobile application has a private key (called r1), and the server application has a public key (called s1). The mobile app requests data from a database. The mobile request reaches the server PHP application, which can access the backend database, as illustrated in Fig.1. The SQL-embedded PHP poses an SQL statement on MySQL. The SQL result is received by the PHP application (applicant.json), and it

Guard let url = URL(string: 
    "http://cysecure.org/xxx/swiftDBretriev.php") else
    {return}

var request = URLRequest(url: url)
let postData = "id="+id   //data to send to php
request.httpMethod = "POST"
request.httpBody = postData.data(using: String.Encoding.utf8)

URLSession.shared.dataTask(with: request) { (data, response, error) in
    if let response = response {
        print("Response: ", response)
    }
    if let data = data {
        do {
            self.mySQLresult = try
            JSONSerialization.jsonObject(with: data, options: []) as! [[String : String]]
        }
    }
}.resume()
is converted into a sequence of JSON data blocks.

Then, applicant.json sends a JSON data block together with the hash code to the mobile application (e.g., iphone swift app). Then the mobile application verifies the integrity of the message by checking “the hash code” received from the server with “the hash code” generated locally. In this way, spoofing attacks if any can be detected.

6 Conclusion

This paper described a new technology of JSON data transmission for mobile devices. A big JSON data can be transmitted in a series of multiple blocks. In order to transmit sensitive data elements in JSON data format, each JSON data block contains a sensitive data element and a hash code technique is employed. Along the JSON tree, our approach is to chain between the parent and child block nodes to ensure data integrity and security. Note that our approach is not going to replace the cryptographic approaches altogether, but rather it integrate with existing cryptographic approaches to manage JSON data blocks more securely.

The contribution proposed in this paper includes block chaining of JSON data sequences. Keeping the hash code of chains of JSON data blocks ensures the secure transmission of JSON data from servers to mobile applications. This approach can detect if there is any JSON data spoofing.

References


