Web Security: Security Methodology for Integrated Website using RESTful Web Services

Dinh Toan Nguyen

University of Tampere

School of Information Sciences

Computer Science/Software Development

M.Sc. thesis

Supervisors: Eleni Berki, Sunil Chaudhary

December 2015

University of Tampere

School of Information Sciences

Computer Science/Software Development

Dinh Toan Nguyen: Web Security: Security Methodology for Integrated Website using RESTful Web Services

M.Sc. Thesis, 60 pages

December 2015

Abstract

Security is not only one of the most important feature in software development but also a key point to gain user's trust. The higher is security, the more trust users put on the application. Especially in Web services, security plays a key role in reaching success for the application. Therefore, this thesis will focus on this important field, that is security in RESTful Web service.

The first five chapters research and analyze the current security methods which are used by some popular API providers. Because there are various security methods in the market, the standard method named OAuth 2.0 and some methods which are required by API providers are selected and researched in this thesis. This part is also the theoretical foundation for chapter six.

In detail, Twitter and Stripe services are analyzed to determine how the important information can be secured through out API calls. This would describe the general standard security model on integrated websites.

The second part includes the descriptions of my own website and C# library which could help users to make an API calls easily and securely. This website is an ideal solution for small business in marketing and sale process. By exploiting the speed of social networks, it offers a huge chance to spread their products over the globe. Furthermore, users are able to develop their own website by using the C# library.

Keywords: Security Methodologies, RESTful Web services, REST API security, Oauth 2.0 3-legg authorization.

Acknowledgement

The thesis is a result of the collaboration and cooperation among many parties. It is not my own goal, therefore, I am honored to say thanks to:

- Antti Torronen: CEO of Kwork Innovations. Antti and his organization helped me significantly in this thesis by offering me a 3-month internship and instructing me to end up with the main idea for this thesis. Moreover, Antti guided me to go through all necessary technical challenges.
- Eleni Berki: My supervisor from University of Tampere, who always supports me kindly and conscientiously. Eleni has instructed me throughout the thesis cycle.
- Sunil Chaudhary: PhD Student from SIS of University of Tampere. He enthusiastically double checked my thesis word by word and gave me valuable suggestions to contribute my thesis.

Tampere, December 2015

Contents

1. Introduction

1.1. Background

According to the CSI Computer Crime and Security Survey for 2010/2011 , 92 percent of businesses have suffered from a successful application attack between July 2009 and June 2010. Even worse according to William Noonan [Richardson, 2010/2011], the frequency and complexity of cyber crimes have increased significantly. In more details, 69 percent of US respondents pointed out that the impact of cyber threats to their business has caught their attention (up from 49 percent from previous year). According to the US State of Cybercrime Survey, 59 percent of respondents confirmed that they have encountered cybersecurity issues in 2014 more than in the last year [PWC, 2014].

The aforementioned situation illustrates the importance of the security in business especially in Web Servers. In addition, more software products are built upon the web infrastructure than native application. Especially, a new page for web application development was turned when Roy Thomas Fielding introduced the new web architecture named REST (Representational State Transfer) in chapter 5 of his PhD dissertation titled "Architectural Styles and the Design of Network-based Software Architectures". This is a reason for the significant growth of APIs [Hughes Systique Coporation, 2006]. The new architecture can foster a better environment for website development, however, it also brings new challenges in security which needs to be adapted in the new infrastructure. Therefore, this thesis explores how security methods are working on Integrated Websites using RESTful Web Server.
This thesis aims to examine the following two main fields:

1. How Twitter and Stripe server can authenticate requests via API calls from various client's applications.
2. How the Integrated Website can exploit the API calls in order to foster better environment for e-commerce.

Based on the above mentioned two questions, this research is divided into two main parts. The first part (chapters 1- 5) focused on the first field. The second part (chapter 6) provides the solution and answers the second field by using a website and a C# library designed by the author, which can help users make the API calls in a simple way and maintain the integrity of the services.
Before proceeding to the next chapter, it is necessary to define the basic terms and concepts used in the thesis.

1.2. Definition of terms and concept

Integrated Website: The basic concept of the Integrated Website is to provide various features and services which use different technologies and are provided by many Web Servers. Basically by using an Integrated Website at the client side, users are able to interact, cooperate and consume many services from different servers.
API: It stands for "Application Program Interface" or "Application Programming Interface". An API is a set of tools, functions and protocols which the user can refer to and apply when developing their own software application [Wagner et al., 2007]. API is

not only convenient, but also easy for the user to build up their application using the third-party services.

Web security: It draws on the principles of defining how to protect the Web Server and Web Client and how to determine the system vulnerabilities through flaws in every stage of the processes such as development, requirement analysis, design and maintenance of the software application. According to CIA triad , a model designed to introduce problem areas and solution for information security, there are three main crucial components of security representing for three elements in a triangle [Hall , 2005]:

- **Integrity**: This is about the rightness of the information. Some unauthenticated accesses could cause unexpected modifications in data, for example, user data has been changed during transaction by a third party [Cryptome, 2013].
- **Confidentiality**: User data has been eavesdropped from both the server side and client side. It could lead to loss of user's privacy. Thus, every single access should be authorized carefully [Hall , 2005]. In case, fake users and data may go through all authentication processes and become valid. As a result, fake users are able to send wrong requests on behalf of victims. This is dangerous for both the services and the users; therefore, this thesis also analyzes some authentication processes which are used by API Providers .
- **Availability:** This refers to the accessibility of the data. This means that the information must be available when it is needed. As a result, authentication mechanisms, access channels and hardware must operate properly. This would work against malicious actions such as:
  - **Denial of Service:** Web Server could be attacked in many ways such as DoS Attack, filling up memory or sending a serial request. This kind of threat may cause an unexpected delay in transferring data between the client and the server or even worse it is able to damage the Web Server [Loukas and Oke, 2009].

Currently, there are many security models. However, all of them may belong to one or more of the following definitions:

- **Domain Based Security Model (DBSy)**: It was built up in the late 1990s by the Defence Evaluation and Research Agency (DERA) . This is a set of notations and techniques developed by QinetiQ specifically for the UK government. DBSy explains and assesses business-driven information security requirements for network architectures. In this model, the browser contains a list of verified domains which have rights to certain actions and the applications are determined by the server domain hosted [Hayat et al., 2005].
- **Certificate based security (Certificate-based Authentication)**: The application is verified by signing a valid certificate by a third party authority. In this case, digital certificate acts as the official identification card which decides the application rights [Microsoft, 2003]. The certificate could be provided by the third party authority named public key infrastructure (PKI) which will be analyzed later on in this thesis. Besides that, the application may generate its certificate which must follow some

rules and restrictions from the authority and this kind of certificate will be introduced also in chapter 5 “Online Payment Security Method”. This is also the most popular model which is widely used in various web servers. Because of its popularity, this thesis will focus on Certificate based security model as the foundation to research the current problems and develop my own practical work. Moreover, there are two sub kinds of certificate based security. This model could use one or both of them:

  - Perimeter Security: This kind of security method is the pre-signed application. Once installed, the application automatically has the right to access APIs whose certificate has been qualified [Taylor and Francis Group, 2003].
  - Least Privilege: This is a time restricted version of the certificate based security. According to which the certification author has signed before, the validation of the application to APIs could be terminated [Langford, 2003]. Currently, the general duration for a single certificate is two hours. It means that after two hours, the application must resign with new validated certificate to continue accessing APIs. Besides that, the application author may request for a long term certification: if the application is signed with this key, it has the right to access those APIs longer than 2 hours (depending on each API provider). For example, Google and Facebook allow long-term application access to their APIs for 3 months.

* **Prompting based security**: This kind of security model requires the application to go through the authentication process by entering its credentials. After getting permission, depending on API provider, that application may be granted in perpetuity since it is qualified or it could be requested to verify itself each time that application accesses any source [Allott et al., 2008]. This model matches well with the very high or very low frequency access. In case , if the application accesses, sends and gets data continuously from a server for a long time, then, the API Provider should grant a long-term access to that application. For example, some reporting and analytics APIs allow customers to send requests frequently to get the latest data. Otherwise single access could be a great choice to maintain the high integrity of the system since this would avoid unexpected modification from unauthenticated access.
* **No security**: Usually the application could not run effectively without high security process, just like the resident could not do anything legally without any official identification. However, there are some applications which do not limit the access or have no explicit security. They pass this burden to the third party authority, for instance, the anti virus application or even the integrated website if it is simple enough.

1.3. Problem Statement

As mentioned in the previous section, security threats could cause many problems for both the service provider and the client such as loss of privacy and data or broken web server. Especially in Integrated Websites, confidentiality plays the main role since

clients enter and send their sensitive information back and forth from the Client site to various Server sites. For example, Flickr, a popular photo sharing website, allows clients to login and use their services with a Yahoo account. By using a third party's account, Flickr not only simplifies their working process but also improves efficiency of their services. As a result, users are not required to register for an account since they are able to re-use an existing account. Furthermore, this also allows Flickr to scale down the storage size of database since thousands or even millions rows of customer details have been removed and provided by the third party. Using the third party's account to authenticate customers is named Social Login. This model becomes more and more popular recently.

Another case is Ebay. In this case, after finishing item selection, users are able to check out and make the transaction at Ebay by using the third party's services such as PayPal. It could bring many conveniences for both clients and service providers; however, this method also causes different challenges in security, especially in confidentiality. Because Ebay is not able to control the protective process of cardholder details.

This thesis will analyze and study those problems by answering these questions:

* How can customers use Social Login to securely login into Integrated Website? Which information of client will be transferred between the third party service and Integrated Website?
* How can the user be authenticated to send requests to various Web Servers?
* How can personal details such as bank account, bank card number and security number be protected while making an online payment?

Moreover, after going through the security process of the API Providers, I discovered some deficiencies which I am able to improve. All researched security processes are efficient and reliable; however, they seem to be too complicated and difficult for consumers. Some current authentication processes require users to have knowledge of programming. Therefore, going through these processes without any problems is not easy for all users, especially for those who do not have much experience in programming. For example, the returned result for each API call could be in JSON or XML (JavaScript Object Notation) or XML (Extensible Markup Language) format-based on the API Provider. Therefore, the user can read all values in the result, but if users need to get those values automatically for some purposes, such as to use as input for another API call or to display automatically on the website, then they are required to have knowledge of JSON and XML.

Further, each API call requires different arguments and the order of those arguments is important. If users make a small mistake in the order or misspelling in the argument value, their request cannot go through the authentication process of the Web Server. In addition, the returned errors from Web Service are quite ambiguous. For example, Twitter returns error code 400 (Bad Request) and their explanation for this case is "The request was invalid or cannot be otherwise served. An accompanying error message will explain further. In API v1.1, requests without authentication are considered invalid." However, there are thousand cases that could happen. Therefore, consumers need to double check their API call request line by line to find the root cause, maybe in some

cases the problem is missing a comma. Through mastering that foible, my own solution will be proposed in this thesis to overcome that ambiguous warning.

1.4. Research Methodology

This is a work-based thesis. Basically, this research will focus on the security methods in Integrated Websites and analyze the Twitter and Stripe workflow. Thus, in this thesis the their deficiencies are pointed out and my own solutions are proposed. After researching some methodologies, Design Science Research in Information Systems seems to be the perfect choice for this thesis since it fosters better environment to solve the technical issues.

The design-science paradigm originally came from engineering and the sciences of the artificial [Simon, 1996]. The key point of this research methodology is to encourage and motivate creations and innovations, which suggest the ideas, solutions, technical capabilities and products through every stage of the development process from the analysis, design, implementation, testing to management, in order to overcome all deficiencies and problems [Weber et al, 2012]. These problems are diverse and able to exist on various forms such as in software development model, formal logic, rigorous mathematics, programming algorithm and many more [Hevner et al, 2004].

The following Design Science Research Guidelines will clearly illustrate the principles for implementing and evaluating good design science research [Hevner and Chatterjee, 2010].

Table 1: Design Science Research Guidelines

Guidelines	Descriptions
Guideline 1: Design as an Artifact	The design needs to efficiently address the organizational problems and viable solutions.
Guideline 2: Problem Relevance	The objective of design-science research is to create, propose and foster the technology-based solutions which are able to solve and overcome the important existing business problem. This could be reached by innovating the process or changing the root cause of the problems.
Guideline 3: Design Evaluation	The quality of the design should be well qualified and rigorously applied by realistic and reliable approaches. A good design should be Observational (analyze intensively organizational structure, multiple projects), Analytical (take many aspects into account such as the complexity of the problems, the potentiality of the solution for organization), Experimental (high usability), Testing (high coverage of domains) and Descriptive (the design should be reliable by developing from existing theory and research).
Guideline 4: Research Contributions	An effective design should prove that its result is highly applicable and realistic to solve the existing

	problems or at least contribute reasonably in that area.
Guideline 5: Research Rigor	The research and its result needs to strictly follow on the nature of problems. The final aim is to build up the suitable solutions for the current deficiencies.
Guideline 6: Design as a Search Process	The methodologies used for the design should satisfy all requirements and restrictions for the organizational environment.
Guideline 7: Communication of Research	Design science research must stand for both technology-oriented and management-oriented sides. It means the design should take various aspects into consideration from technical problems, management usability to organizational restrictions.

1.5. Structure of the Thesis

This thesis researches web security in Integrated Websites, in particular the APIs transaction. The thesis is divided into two main parts. The first part includes five chapters. Chapter 1 provides a brief introduction and summary of the key concepts of this thesis and some definitions which are needed in the forthcoming sections in advance. Chapter 2 summarizes the RESTful Web Service and its security infrastructure and Chapter 3 introduces the standard security (OAuth 2.0). Chapters 4 and 5 analyze the security methods used by social networking application called Twitter and a online payment API Provider named Stripe. Chapter 6 proposes my own solution by building up a website and a C# library. The final chapter concludes the thesis and includes the limitations as well as some future enhancements of this thesis.

2. The RESTFul Web Services

Back in the time when REST definition was introduced by Roy Fielding in his academic dissertation "Architectural Styles and the Design of Network-based Software Architectures" in 2000 at the University of California, this theory did not get much attention, however, at the moment REST is a one the most popular term and plays a main role in Web Development [ProgrammableWeb, 2012]. It became the foundation for the rocketing of the APIs. The REST's frameworks have been appeared more and more in a vast majority of the existing APIs in the market [Slideshare, 2013].

According to the Programmable Web Research Center [Slideshare, 2013], from 2005 to 2013, the Web APIs experienced the stunning growth which have been stronger than ever (from only one API in 2005 to 10302 APIs on October 2013) as can be seen on the graph on Figure 1.

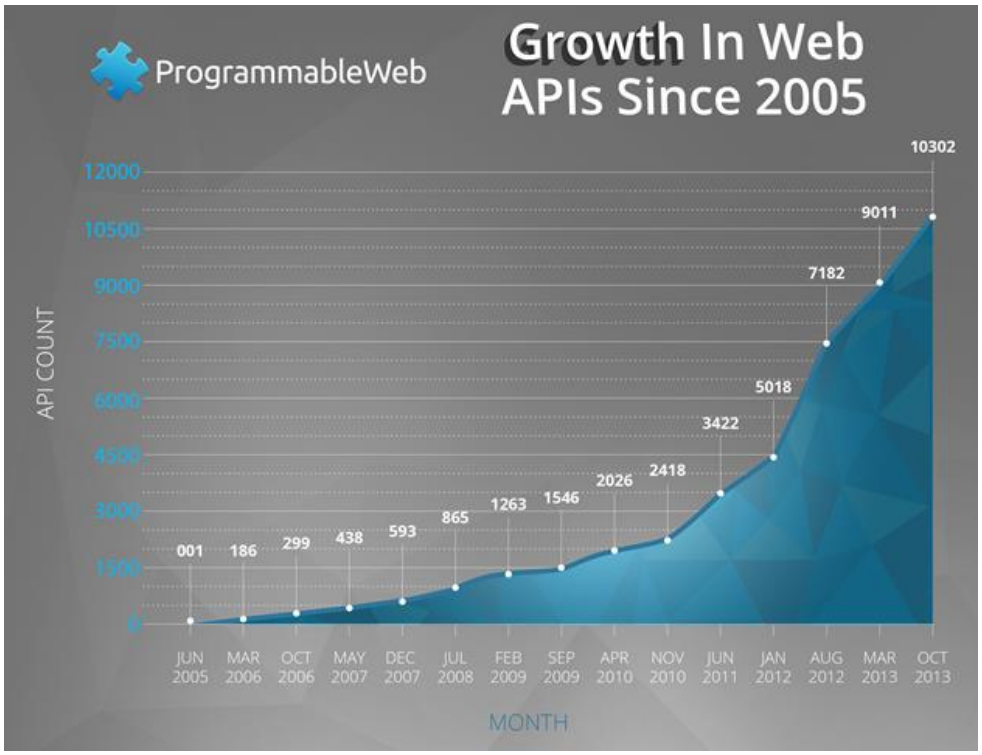


Figure 1: The Growth in Web APIs from 2005 to 2013 [Slideshare, 2013]

The question here is what could have caused this interesting growth ? The following chart, Figure 2, could partially answer that question. As has been stated in the Survey which was published on 11th Dec 2012 by the ProgrammableWeb [2012], over two third of selected statistic APIs have been developed by REST's framework. At this point, the impact of REST to the web development was drawn clearly.

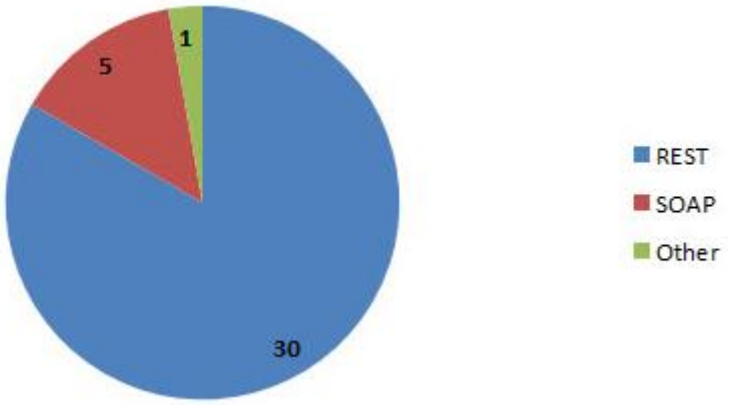


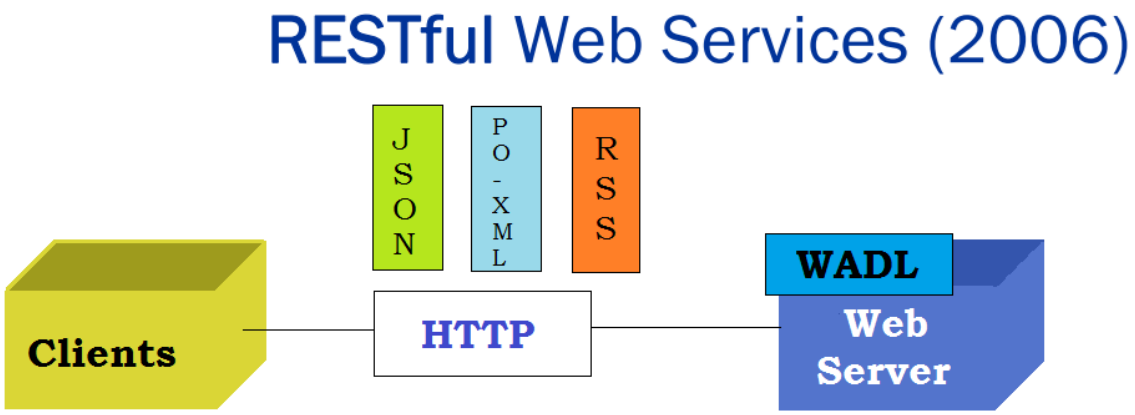
Figure 2: Protocols Used in APIs [ProgrammableWeb 2012]

The following sections will introduce further about the concept of RESTful Web Service and its work flow.

2.1. Overview of RESTful Web Service

REST stands for Representational State Transfer. Overall, the discipline of REST includes the stateless client-server, cacheable communications protocol.

According to the definition from IBM [2015], "*REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages*". This is also the most outstanding feature of the RESTful Web Service, the resource-oriented. This feature simplifies significantly the complexity of the web architecture by avoiding to use complicated mechanism such as CORBA, RPC or SOAP to communicate back and forth between client and server [Fielding, 2000]. The simple HTTP methods with some adjustments are replaced those algorithms to improve the services as can be seen on Figure 3 below.



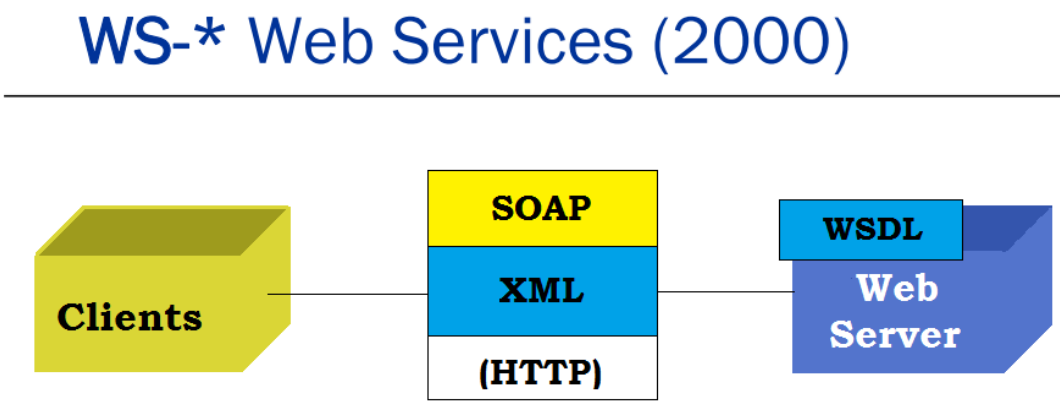


Figure 3: The comparison between RESTful Web Service and Web Service [Pautasso, 2008].

The following section will analyze more deeply what could make RESTful web service more powerful than the others and how RESTful style brings clarity on how to build highly scalable web applications.

2.2. The architectural specifications

2.2.1. Explicit HTTP methods

There are four main HTTP methods in version 1.1 named GET, PUT, POST, and DELETE (obviously there are also another methods such as HEAD, CONNECT and TRACE; however, this research will not focus on them since those methods do not play a big role and are not applied by many API providers). In RESTful Web Server, each method represents for one specific interaction and modification to the resource:

- **GET**: This method fetches the data from Web server, searches for the requested resources and collects data as the request. This method does not have the right to change or modify the resource.
- **POST**: POST method aims to create the new resource in the server. Because of that characteristic, the POST method is required to contain the body which clarifies all necessary values for that creation. As alike the GET method, this method also cannot modify the existing resources either.
- **PUT**: PUT method is used to change the state or update value of the resources. Different from GET and POST, this method is able to modify the existing resources. As the result, PUT method must contain the new value in order to modify the resources.
- **DELETE**: This method removes or deletes the resources. In some cases, the API providers do not support this kind of method. They do not allow user to delete the resources and those resources are just moved to a separate folder or changed the status to deleted. In those cases, DELETE and PUT method are quite similar.

By using separated methods for specific purposes, it not only improves the elucidation of the request but also reduces the workload for the server. Let's take a look at some HTTP methods in RESTful to witness how efficient and simple they are.

For example, there is a resource named People in the server. In case the client want to retrieve the information of a person name Peter in People resource, the HTTP GET request should be structured as below:

```
GET
http://example.com/v1/people/user_info.json?name=peter&count=1
```

The URL of the request should be readable and the arguments are separated by "&" character. In this case, the request asks to query a person name Peter and the returned result should be in JSON style and contains only one value if there are many people having the same name Peter. By the HTTP method, Web server is able to understand that the request is asking to collect and get information about Peter, not asking to create new people name Peter in the resource.

2.2.2. Resource-oriented RESTful URI

This feature describes the most clearly about resource-oriented in RESTful Web Service that is also the most outstanding function of this model. The URI will intuitively point to which resource will be interacted by that request. By reading the URI, the developers and users are able to have the awareness about what the request will modify in the resource. The REST style encourages having a separate URI for every single data item, like a photo or entry in a database. Let us examine the following URI to study deeper in the URI structure:

```
POST
http://www.example.com/bookstore/books/name=exambook&ISBN=ISBN12
3
```

As mentioned in section 2.2.1, by the HTTP method, this request is asking to create a new book with tittle "exambook" and its ISBN is ISBN123. Users and developers can understand this request by reading the URI, however, how web server could understand the request and execute its command. The answer is the structure of the URI. Basically, a complete URI contains three main parts which define apparently the aims of that request:

- The Host: This is the address of the Web Server. In this case, the host is "`www.example.com`". When receiving the request, firstly the web server will check the validity of the host of that request to ensure that request has been sent to the correct address.
- The resource: The interactive resource name. After checking the host, Web server will detect and approve the resource name. The resource name of the URI should follow structure of the resource. For instance, there is a resource named "book" in the "bookstore" resource and the resource name on the URI should be bookstore/book. Two values need to be separated by "/".

* The value: This is the clarification for the request. This part will supplement the requirements to narrow the scope of the interaction or specify more details for the action of that request. In the above URI, "`name=exambook&ISBN=ISBN123`" plays role as the value of the request. It clarifies that the request is asking to create new book with name "`exambook`" and `ISBN123`.

By applying the clear structure, the RESTful URI not only is intuitive to read, understand and resource-oriented (i.e., every single request has to specify the resource which it wishes to take action on inside) but also simplifies the request.
Let us compare the complexity of the request between Web Services using SOAP and RESTful Web Service [M. Elkstein, 2008].

```
<?xml version="1.0"?>
        <soap:Envelope
        xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
                <soap:body pb="http://www.acme.com/phonebook">
                        <pb:GetUserDetails>
                                <pb:UserID>12345</pb:UserID>
                        </pb:GetUserDetails>
                </soap:Body>
        </soap:Envelope>
```

Code Snippet 1: Request using SOAP Envelope

Code Snippet 1 illustrates a request using SOAP envelope. In this request, users are required to create a SOAP envelope having a body inside. In the body, the action of that request must be pointed out by using XML structure. Moreover, all extra information for that action has to be inserted inside the body. This structure will push the burden on both client and server side since the users need to create a complicated form for a simple request and the server needs to handle a complex XML to retrieve the necessary information. For the same purpose, in RESTful, the request will be short, simple and precise. It can be interpreted as below:

```
GET http://www.acme.com/phonebook/UserDetails/12345
```

Only one line can represent for the whole complex XML file and there is no request body. It is just a URL, which is sent by the HTTP GET method. Because of the simple structure, resource-oriented and user friendly, REST becomes the infrastructure of more and more APIs since the documentation is much easier to understand and implement.

2.2.3. Representation

As explained above in Figure 3, the difference between the RESTful Web Services and the Web Services in 2000 is that the REST is using the HTTP method combining the some standard readable formats such as JSON or XML. The JSON is also represented for the resource which is triggered in the request and it reflects the current state and properties of the resource. This JSON or XML is also the response of the request and included in the HTTP body.

Here is the sample returned JSON for the request to get the information of Facebook user as below:

```
GET /v2.5/{user-id} HTTP/1.1
Host: graph.facebook.com
```


```
{
 "id": "1388298271499685",
 "first_name": "Juho",
 "gender": "male",
 "last_name": "Ilmari",
 "link":
"https://www.facebook.com/app_scoped_user_id/1388298271499685/",
 "locale": "en_GB",
 "name": "Juho Ilmari",
 "timezone": 2,
 "updated_time": "2015-04-17T06:58:11+0000",
 "verified": true,
}
```


This result is not an actual resource. It is just a JSON file containing all necessary attributes for that resource, in this case, the returned properties are the Facebook Id, first name, gender and so on. For some properties which are heavy size such as Images and Videos, the link will be included in the JSON file and customers can access and download them later. This could help server to avoid the cumbersome transactions since the simple and readable text are sent only. This also could lead to reduce the dependency on the infrastructure of the platforms and devices, thus, these services can be used on various systems in spite of the programming languages.

2.2.4. Stateless

The communication in RESTful Web Service must be stateless. It means that every single request from the client side to the web server has to contain all the essential information to analyze and execute that request. This could lead to reducing the storage in the server since there is no need to save or store the context of those requests in the server side. Therefore, the session state is kept totally on the client side. One of the conditions to validate the request is the URL structure which was introduced above. Usually one request should contains three main parts, moreover, there is possibility that the API provider will have some extra compulsory arguments. For example, the AdWords API of Google also requires their clients to include the version of API in the request. The requests cannot go through the authentication process of Google if there is any missing argument.

Once again, this constraint will maintain the high performance of REST. Stateless shows its advantages in:

* **Visibility**: The requests are clear enough for the server to spend no more extra effort to take a further look beyond the request. As mentioned above, in stateless design, the request must be as much detailed as possible and because of this characteristic, the visibility of each request improves significantly.

* **Reliability**: In REST, the partial failure from the previous request could be recovered easily without duplicating or changing the state of the resource. Let us take a look at a failure of DELETE request as an example (Figure 4) [Envoisolutions, 2007].

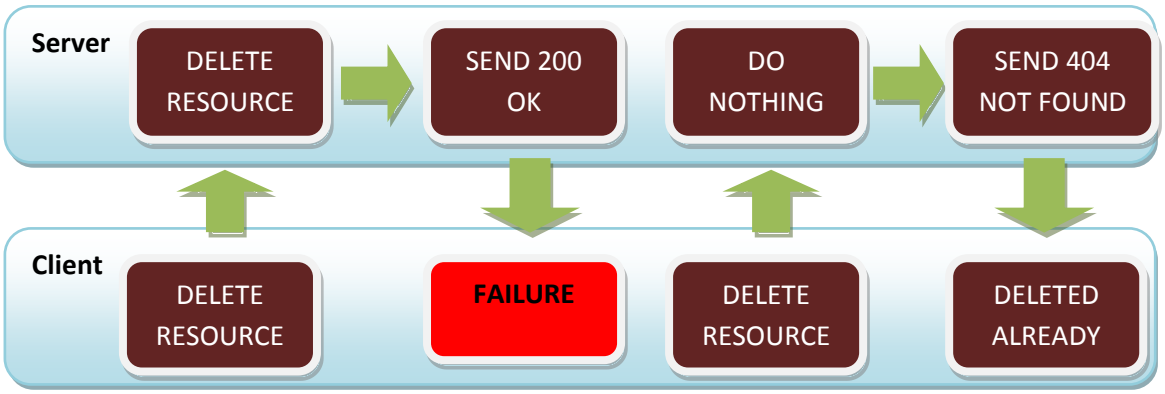


Figure 4: Reliability in DELETE request

In this case, the client side sends a delete request to remove one object in the resource to server. After qualifying the request, the server accepted and executed the order as requirement and sent back the confirmation. However, because of connection error, there is a failure at this stage and the confirmation cannot be sent to the client. As a result, the client will assume that the request has not been done and the new request will be sent again to server. At this time, server will take no action since that object was deleted already then the invalid confirmation will be sent back to client. Thus, the system integrity is maintained as highest as possible since there is no change or duplicate in the resource.

* **Scalability**: This is one of the outstanding features of the REST, the server is not required to prepare the space for the request result since all essential information of result was sent back to the client as the reply of that request. Therefore, the request should determine the kind of returned result in Content-Type inside every request. This will reduce latency, network traffic, and server load.

2.3. Security features in RESTful Web Service

So far, we have gone through many aspects of RESTful Web Services. In this section, one of the most important feature of REST and this thesis , the security in REST, will be introduced. The key point of security is validation. Qualify all inputs on server side to protect the server against SQL or NoSQL injection attacks. This kind of attack will exploit the security vulnerability in an application then query to get data and execute unexpected SQL query to modify and change the database. Preventing this attack will protect and maintain the integrity of the database.

There are several mechanisms available to secure the web service:

HTTP Basic Authentication: In this model, users are requested to verify themselves by user-ID and password for each request. The value of both ID and password should be opaque string which is unreadable by naked eyes. The server will only serve those requests which were validated already. The basic framework is illustrated as below:

```
credentials = "Basic" basic-credentials
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

In the example, the userid is "Aladdin" and password is "open sesame", however, those strings are not passed directly to the request. The opaque string, which is the result of based64 encoding, is used instead [The Internet Society, 1999].

Digest Access Authentication : Different from the HTTP Basic Authentication, Digest Access Authentication provides a no encryption of message content. The most interesting point of this model is using the nonce value. It means that every single request must contain a unique value. Whether, the request is executed or not, valid or not, authenticated or not, it must regenerate a new unique value for each request. As same as Basic Authentication, the server also authenticates the WWW-Authenticate header which is illustrated below:

```
credentials      = "Digest" digest-response
       digest-response  = 1#( username | realm | nonce | digest-uri
                        | response | [ algorithm ] | [cnonce] |
                        [opaque] | [message-qop] |
                            [nonce-count]  | [auth-param] )

       username         = "username" "=" username-value
       username-value   = quoted-string
       digest-uri       = "uri" "=" digest-uri-value
       digest-uri-value = request-uri   ; As specified by HTTP/1.1
       message-qop      = "qop" "=" qop-value
       cnonce           = "cnonce" "=" cnonce-value
       cnonce-value     = nonce-value
       nonce-count      = "nc" "=" nc-value
       nc-value         = 8LHEX
       response         = "response" "=" request-digest
       request-digest = <"> 32LHEX <">
```

Code Snippet 2: Digest Access Authentication framework example

In this example, digest-response, a string of 32 hex digits computed from the above values, represents for user password. The “qop” is represented for “quality of protection” and it is relevant to nonce, which prevents server from **chosen plaintext attacks**. This is also the improvement from HTTP Basic Authentication. In previous model, attacker can specify plaintext then encrypt or sign it to reveal the characteristics about the encrypting algorithm. For some cases especially in Public-key cryptography, this model is very feasible and the attackers are able to exploit this flaw to send many authentication requests as long as they know the public key. However understanding this deficiency, Digest Access Authentication model expects users to include a unique value in the request [The Internet Society, 1999].

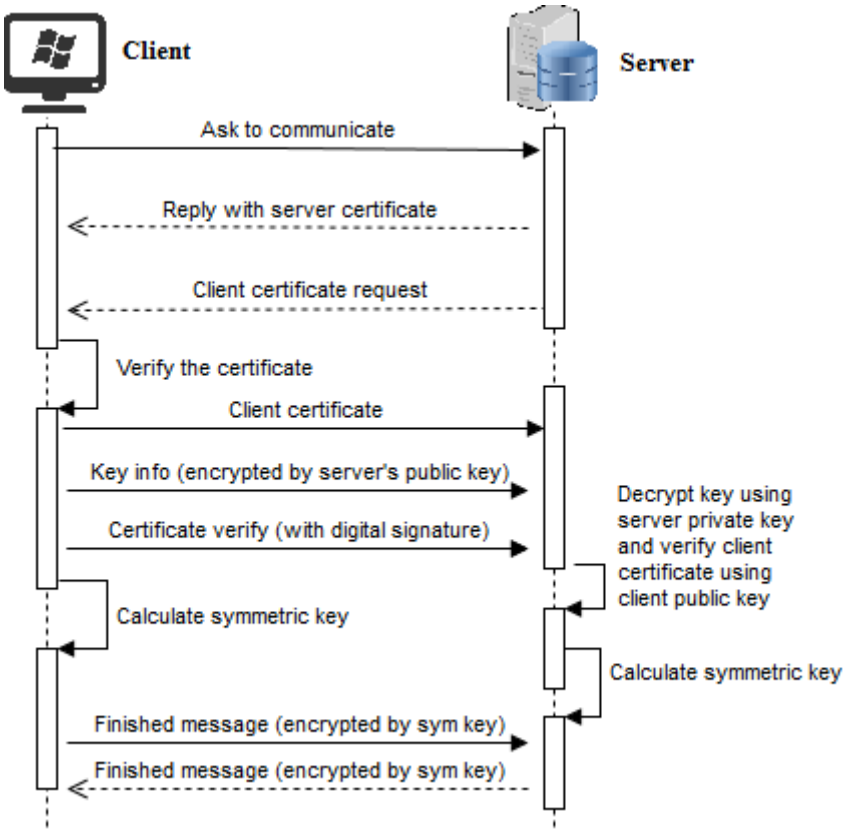


Figure 5: Client Authentication [Wagnon, 2013]

Client Authentication: The client side receives identity from the server side. In this model, the client is required to authenticate the server's identity, however, it is not compulsory for the opposite way. Thus different from models above, Client Authentication runs the authentication process on the client side and the all messages between client and server are encrypted [Wagnon, 2013].

As can be seen on Figure 5, after receiving request from the client, the server automatically sends the certificate twice. The first time announces client that it needs to make its certificate ready because the next message from server requires to verify client's identity. The next steps, client sends its certificate, keys to encrypt and digital signature to qualify itself. After going through the authentication process, the client sends request to sever which is encrypted by a symmetric key. At this time, server executes the request and sends back confirmation to client.

OAuth Authentication: This model becomes more and more popular which is applied in the vast majority of RESTful Web Services. This model is also chosen to be analyzed and researched further in the next chapter.

So far, this chapter summarized the nature of the RESTful Web Services to witness the differences from the Web Services in 2000. The stateless design (the most extraordinary feature) has affected gradually on how the security changed in RESTful Web Service. The explicit HTTP methods prevent accidental changes or modifications in the resource, the URI helps to scale down the interacted scope on the resource. Representation is to maintain the integrity of the system and resource since users are able to download the object only. Finally, stateless design reduces the negative abilities of resource errors.

In the next chapter, the standard authorization method named OAuth 2.0 will be clarified. The introduction and main concept of OAuth will also verified on Chapter 3.

This model is the infrastructure for many API Server, thus, by going through this standard model, the basic idea of further discussion is also summarized briefly.

3. The Standard Security Method in RESTful Web Services

3.1. Overview of OAuth 2.0

As mentioned in section 2.3, there are many security models which are used in RESTful Web Service, however, whenever considering about the most common model among of them, one of the first model that comes to mind is OAuth 2.0 since it is the most common model.

OAuth (Open Authorization) 2.0 is the open authorization protocol for native and web application to secure their services by using simple and standard method. It handles the income requests from various sources to authenticate who is communicating and its right to access to database which is saved by another application. Let us take Flockler (as link http://flockler.com/new-factory) website as an example, a social magazine application. It collects and displays many posts from various social resources such as Twitter, Facebook and Instagram on selected topics or from configured accounts. This application will send thousands of requests to OAuth server and require to verify itself for accessing to the databases of those social servers. After OAuth server accepted their access, the application is granted the right to retrieve data.

Before accessing the resource server, clients are required to verify themselves by sending a request to OAuth server containing user's credentials, in some cases those credentials are public key, private key or customer ID. OAuth Server qualifies that request and return a valid token which is represented for that customer and valid for 2 hours usually. Next, user needs to attach that token in request header and sends it to resource server with which he/she wishes to interact. The Resource server communicates with OAuth server to verify coming token after receiving the request. If the token is qualified successfully by the OAuth server, Resource server executes that request and returns successful response otherwise an error is sent back to the client.

The centered point of OAuth authentication process is the access token. Instead of relying on user's password as the master key for every access to server, OAuth generates a token based on the user credentials. This token stands for the access to single API on behalf of specific client.

The following diagram illustrates the work flow of OAuth authentication in general [Hughes Systique Coporation, 2006]:

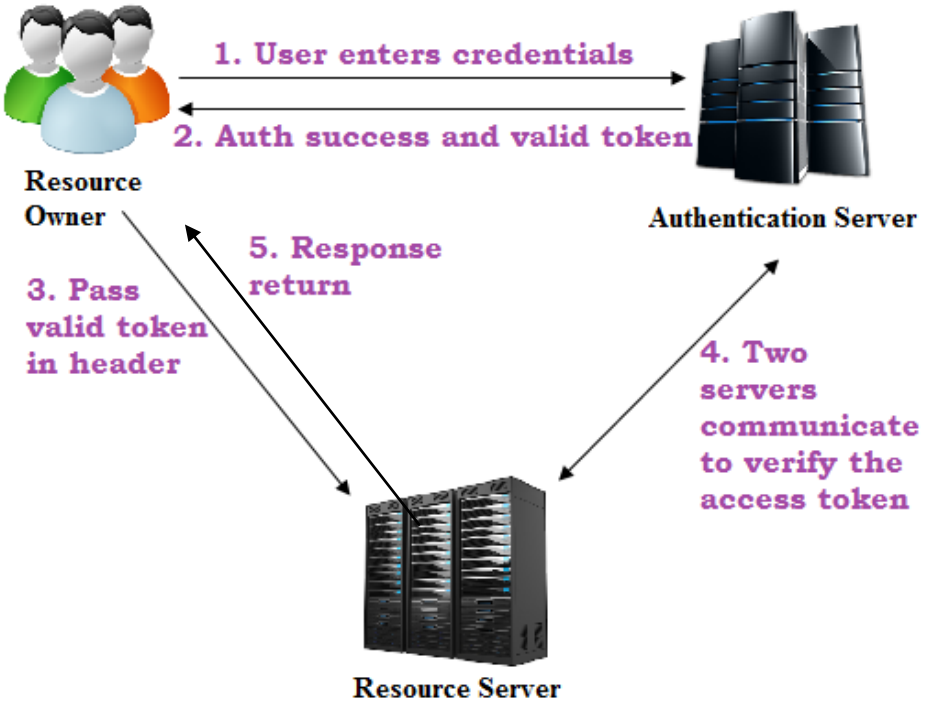


Figure 6: OAuth authentication general steps

By applying this model, both server and client could reduce much work load since they do not need to analyze and spend much effort on security. According to The Internet Society [1999], in HTTP Basic and Digest Authentication model, the server is responsible to verify all requests. In Client Authentication model this authentication process takes place in client side mostly. However in OAuth model, both client and server could take their hands off this process since there is a third party which takes care of it. This is not only reducing much work load for both client and server but also improving security level for the application. The application developers are not the specialist in security and from their point of view, they want to be hand free on protecting sensitive data. The final goal of the application developers is to implement all the functionalities with as least as possible bugs, therefore, security functions should be on another party's hand and in this case, the specialists of OAuth are glad to share this burden with application developers. This is also the answer why this model is applied widely by many large API Providers such as Facebook, Twitter and Google.

3.2. OAuth 2.0 Core

3.2.1. The OAuth 2.0 Authorization Framework

As mentioned in section 3.1, OAuth separates authentication process from resource server by using access token which is granted by OAuth Server with the approval of the resource owner and reflected for a registered client in limited time. Basically, there are four roles in OAuth [Hardt, 2012]:

* **Resource owner**: the entity grants access to secured resource. In case resource owner is people, usually it stands for the end-user.

- **Resource server**: the server hosts all resources which accepts the authenticated client to access data using access token.
- **Client**: the person, the application sends the request on behalf of the resource owner to retrieve data from Resource server.
- **Authorization Server**: the server grants access token to client after going through all authentication processes successfully.

Going back to Flockler case to explicit the roles in OAuth more intensively, Facebook, Twitter and Instagram play a role as Resource Server because these servers store all posts and photos for Flockler. Therefore, users of those social networks are the Resource owner since they own those posts and photos and Flockler itself is the Client. Flockler will send the request on behalf of the social users to get the access token and use that access token to collect all necessary posts and photos from those servers. Last but not least, OAuth stands for Authorization server. OAuth server will verify those clients and provide the access token for valid clients.

This section will focus on how the Authorization server can qualify the Client, how the Client is able to be represented for Resource owner. The answer for those question is Authorization Grant. This is a credential representing the resource owner and used by the client to generate access token. This authorization grant will be sent to Authorization server by Hypertext Transfer Protocol. OAuth defines four kinds of Authorization grant as below [Hardt, 2012]:

- Authorization Code: This is the returned result after navigating user-agent (the combination of the product tokens which are used to grant communicating applications to identify themselves by software name and version) of Resource owner to Authorization server by the client. The application uses this authorization code to communicate with authorization server to go through the authentication process. This authorization code proves the ability to authenticate client and the chance to pass the access token straight to Client without leading through Resource owner.
- Implicit: This is an uncomplicated method of Authorization Code. In Implicit flow, the intermediate credentials are not generated and the access token is issued directly to the client by using scripting language such as JavaScript. This kind of method is compatible for in-browser application since the Authorization server is able to check the access token in URI and some round steps are dropped off in this method.
- Resource Owner Password Credentials: The Resource owner's username and password can be used directly to obtain access token. The Client will verify itself by using these two credentials. However in my opinion, this could be the least secured option among four kinds of Authorization grant. Sending distinct username and password back and forth is never a high secured solution since the tracking application may listen to your message and reveal your username and password.

- Client Credentials: Instead of using the Resource owner credentials, Client is able to use its credentials to verify itself and obtain the access token, however, this method can be applied only when those credentials are qualified and Client's right is limited on modifying, changing or retrieving resources.

This thesis will focus on the Client Credentials method since this kind of Authorization Grant are applying widely by various large API Provider such Twitter. Therefore, let us take a further look on this method to have a general view of the work flow in the next section.

3.2.2. Client Authentication in the OAuth 2.0

Every registered client is granted a Client Identifier, which is a unique string and is not confidential, by the authentication server. Each time client request authentication server to obtain the access token, the client identifier is required in that request to prove the validity of that client. Client can establish a set of credentials for authenticating such as password, private-public key pair as explanation below:

- ***Client Password***

According to "Client Credentials" model in section 3.2.1, client is able to send its user-name and password to authentication server as authorization grant to generate access token. In OAuth 2.0, client could send its identifier as user-name by using HTTP Basic Authentication (which was introduced in section 2.3), however, the identifier must be encoded using the "application/x-www-form-urlencoded" encoding algorithm. Here is an example of access token request using Client Password:

```
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3
```

As can be seen, in the Authorization header, there is only one string which is represented for the client. This is the result after encoding the combination of username (identifier) and password. As a result, the authentication server must support the HTTP Basic Authentication.
Moreover, OAuth 2.0 also supports client to include its credentials in request-body using below arguments:

- Client_id: the client identifier after registration process.
- Client_secret: the confidential string is used only by specific client.

Basically, the client_id could be seen as client username and the client_secret is the password. Here is the example of the request using client_id and client_secret:

```
POST /token HTTP/1.1
     Host: server.example.com
     Content-Type: application/x-www-form-urlencoded


grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA&client_i
          d=s6BhdRkqt3&client_secret=7Fjfp0ZBr1KtDRbnfVdmIw
```

According to OAuth, this kind of Authorization Grant is not recommended and this is only option if client somehow does not have ability to use HTTP Basic Authentication. From my perspective, explicit client_id and client_secret value in the request could be the problem. By using HTTP Basic Authentication, client username and password are encoded and combined then sent only one encoding string to authentication server, however even though being encoded, client_id and client_secret are separated and readable in the request. Attackers could exploit this flaw to decrypt those two arguments by using chosen-plaintext attack. That could be the reason that client_id and client_secret have lower priority than HTTP Basic Authentication.

- **Public Key Infrastructure**

Public-key Infrastructure (PKI) is a cryptographic mechanism and this is the innovation of Symmetric-key cryptography. If Symmetric-key uses the same key for both encryption and decryption process, Public-key cryptography applies a key pair (a public and a private key) for this process and separates each key for each process. Public key is not confidential and distributed commonly, however, private key should be kept secret. Because of using specific key for specific process, the message, which was encrypted by public key, can be decrypted by private key only and vice versa the public key is able to decrypt the message if it was encrypted by private key [CGI Group , 2004].

Encryption Process:

In OAuth, every registered client is granted a pair of key, thus, each time client requests authentication server to generate the access token by using client credentials, client needs to include its public and private key in that request. As can be seen at Figure 3, at the beginning, stage 1a, the readable message is calculated digest, a unique representation (a bit like a sophisticated checksum) by using hashing technique. After that at stage 1b, the digest is encrypted by using client's private key to generate the signature, however, this signature also includes the client's public key. This could help the authorization server to verify and qualify that signature by using issuer public key. In the next stage 2a, the onetime symmetric encryption/decryption key is generated since asymmetric-keys are cumbersome for a long message.

In the encryption process, the message, the signature and the public-key will be encrypted once again by a symmetric key as stage 2b. This leads to including that symmetric key inside the message so that the recipient could decrypt later. However, there is one consideration here is that how to prevent another person from that symmetric key, it means that only recipient, who the request is sent to, is able to retrieve it and decrypt that message. The most secured way is to once again encrypt the symmetric key by the recipient public key at stage 2c. Finally, the whole package will be sent to server.

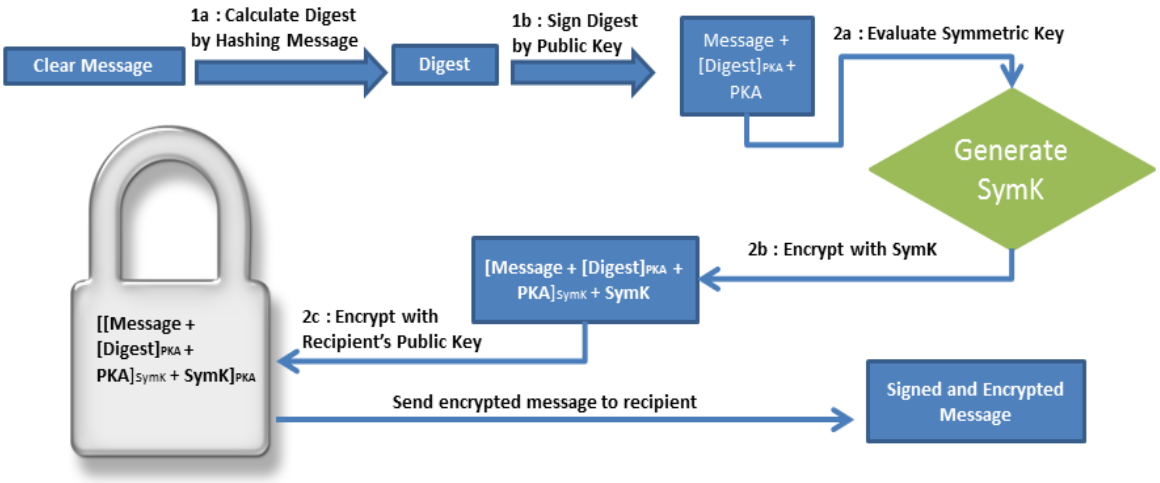


Figure 7: Signature and Encryption details with keys [CGI Group , 2004].

Decryption Process:

After encryption process, the recipient receives the whole package. Firstly, the recipient needs to decrypt that package by using its public key as description at step 1a. After unwrapping the package, the recipient has the encrypted message and the symmetric key. Next, that symmetric key is used to unbox the message as state 1b to collect the client public key and the signature.

In the verification stage, there are two parallel processes. The first process is to decrypt the signature by using the public key which was collected in the previous step 1b. The final result of the step 2a is to get the digest inside that signature. The other process once again evaluate the message by using the same hashing formula as client did since the hashing is one-way process. It means that server is not able decrypt the digest to retrieve that message. Therefore, the final result of step 2b is also a digest. At this moment, server compares two digests from step 2a and 2b. In case those digests are exactly the same, the message and signature are qualified and accepted.

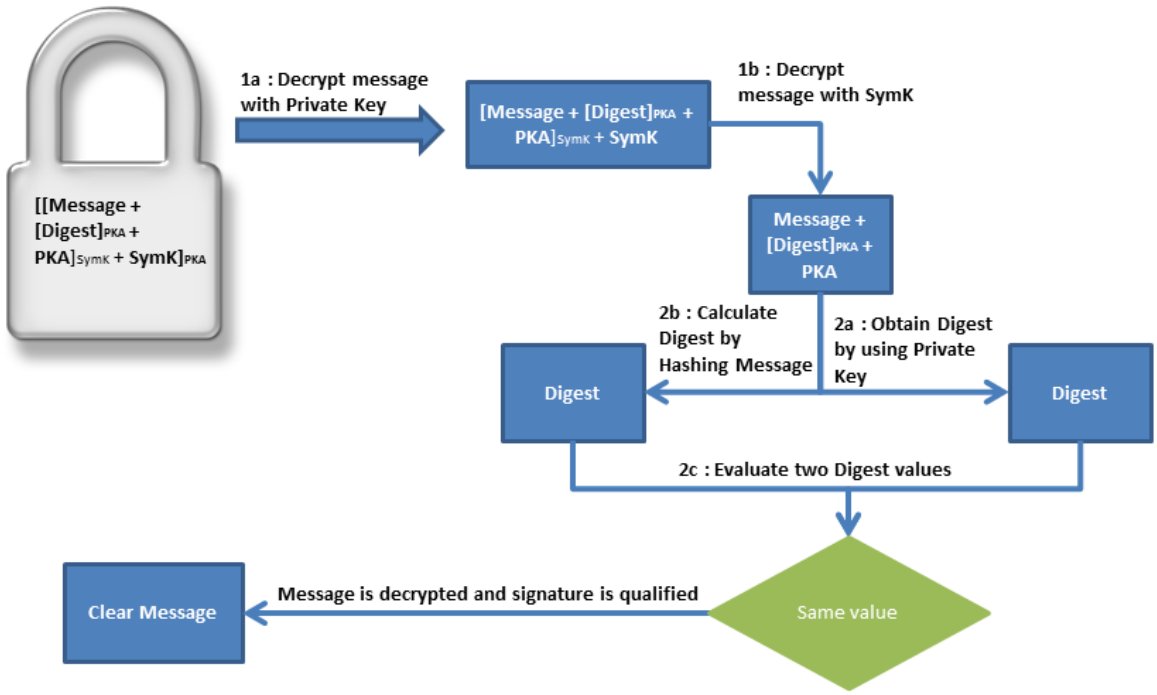


Figure 8: Decryption and verification details with keys

This authorization model is applied significantly by many API Provider which allow programmers to develop their own application to retrieve data automatically from resource server such Twitter, Facebook and many other social networks. This model will be analyzed in Twitter system in the next chapter.

4. Twitter Security Methods

Twitter, one of the biggest media API Provider which has 307 million monthly active clients [Statista, 2015], provides various methods to retrieve its data on behalf of Twitter user by using its application. By making authorized calls to Twitter's APIs, the application is able to access friends and followers of any account, retrieve any user information and access user's resources. By this convenience, the huge amount of integrated websites are using Twitter REST API every day and this trend has no end at least for the next few years. However, this would lead to a big risk and challenge to Twitter for securing its customer credentials. This chapter analyzes and focuses on how Twitter authorizes those applications to avoid leaking user's resources out unexpectedly.

4.1. Overview of Twitter Security Method (3-Legged Oauth 2.0)

Twitter verifies the application right by following the 3-legged OAuth 2.0 model. This model is the interaction among three roles that is why this model named 3-legged OAuth, the resource owner, the server and the client (application). The overall flow of this model is that the resource owner must grant access and restrict the interaction to client and the client communicates with server by the access token on behalf of that resource owner. By this way, the application is able to retrieve what user allows to collect. This reduces the risk of resource loss and modification. In order to send the valid request, the application needs to obtain the access token on behalf of a Twitter user. This is the compulsory and the first step of the authentication process. It aims to:

- Address that the application is legal to access a user account. User needs to grant the access to the application to her/his account manually to retrieve data. This not only proves that user is aware of the application existence but also supports user to manage the application right. Whenever netizen feels unsatisfied with that application, he/she is free to block the application access.
- Limit the application's interaction. The application is only able to implement what user have allowed it to do. It means that the application cannot interact on any resources which the resource owner does not grant access to. This helps to protect the resources and maintain the integrity of system by avoiding leaking out information or changing state of the resource accidentally.

Twitter asks user and the application to go through a main process to obtain access token:

Sign in with Twitter.

As can be seen on Figure 9, consumer is requested to grant access to application in order to retrieve the valid access token. Resource owner is able to issue the right to application by using "Sign in with Twitter" function. If user signed in successfully, Twitter will ask him/her to approve the selected application. Next, OAuth Server will issue the valid access token. Otherwise in case that user signed in unsuccessfully,

Twitter continues asking to provide valid username and password, then continue with above process.

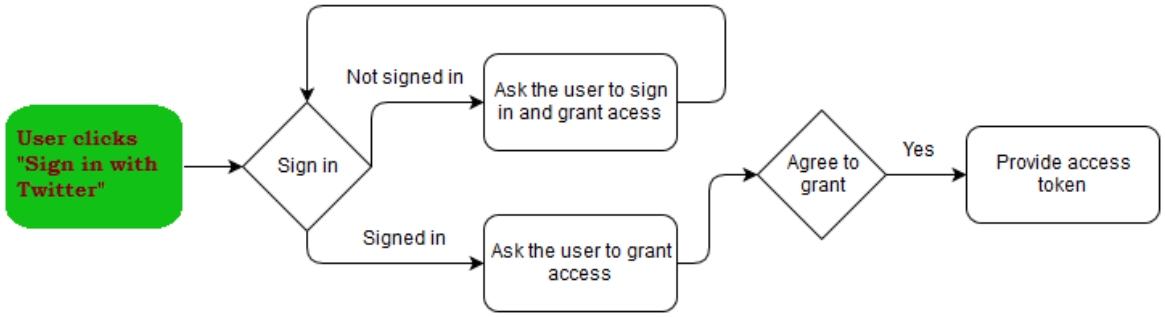


Figure 9: Signing in with Twitter

According to "OAuth FAQ" Document on Twitter [Twitter Documentation, 2015], currently Twitter does not expire the access token. It means that the access token will be valid as long as user is allowing its access. From my perspective, this would bring many benefits for both user and application. Firstly, the application is not required to send request to generate new access token frequently and as the result, the work load on both server and client side are reduced significantly. Secondly, client is not required to sign in each time the previous access token is expired.

Twitter's implementation, which is based on the Client Credentials Grant (in section 3.2.1) flow of the OAuth 2 specification, was illustrated in Figure 10. Now let's take a further look on technical flow in detail to witness how Twitter authenticate resource owner and application to generate access token.

Step 1: Obtaining a request token

At the beginning, the application needs to send the request to OAuth Server to generate a request token. This token is represented for one specific client and it is not an access token which means that that resource owner needs to grant the access to application through this request token.

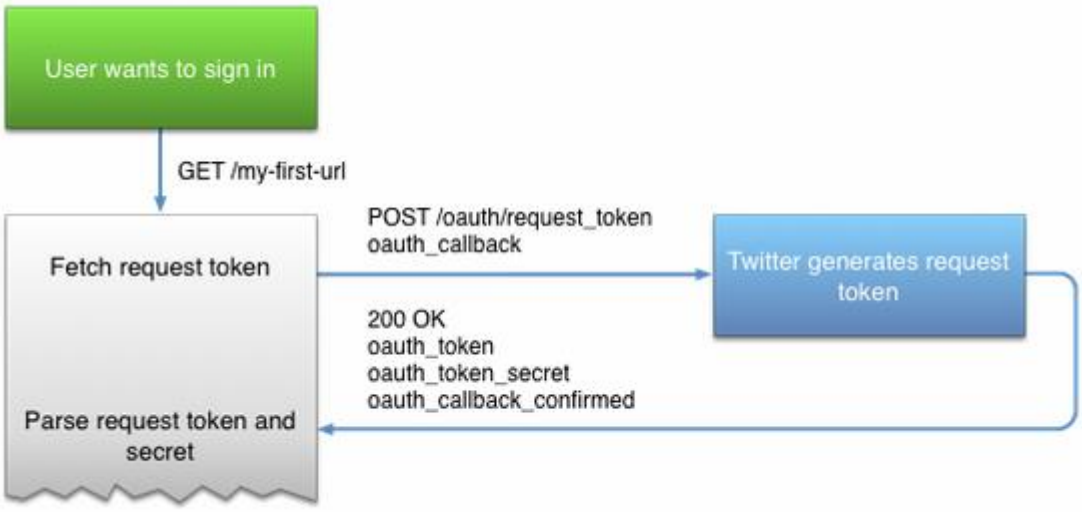


Figure 10: Generating Request Token

Figure 10 shows the overall flow for this step, the application has to send the POST request including its credential to server. After that, Twitter generates the request token and sends it back. Here is an example of the request:

```
POST /oauth/request_token HTTP/1.1
User-Agent: themattharris' HTTP Client
Host: api.twitter.com
Accept: */*
Authorization:
        OAuth oauth_callback="http%3A%2F%2Flocalhost%2Fsign-in-
with-twitter%2F",
              oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
              oauth_nonce="ea9ec8429b68d6b77cd5600adbbb0456",
              oauth_signature="F1Li3tvehgcraF8DMJ7OyxO4w9Y%3D",
              oauth_signature_method="HMAC-SHA1",
              oauth_timestamp="1318467427",
              oauth_version="1.0"
```

There are two most important arguments in this request. They are:

* oauth_consumer_key: The value represents for the client (application). This value aims to notify the authentication server about which client is requesting for the request token and avoid any mistake in mapping relation between client and resource owner.
* oauth_callback: This is a link that the authentication server will navigate to after signing in successfully. Twitter will include the access token and the token secret in the response and send them back to the application at this link. Therefore, at this step, the application must notify the server about the destination of those top-secret information to avoid leaking sensitive credentials out.

Step 2: Redirecting the user

After the first step, the authentication server is notified by client about the resource owner who is wishing to allow access to that application. Thus, at this step, the client should navigate resource owner to server in order to verify him/herself and grant access as Figure 11.

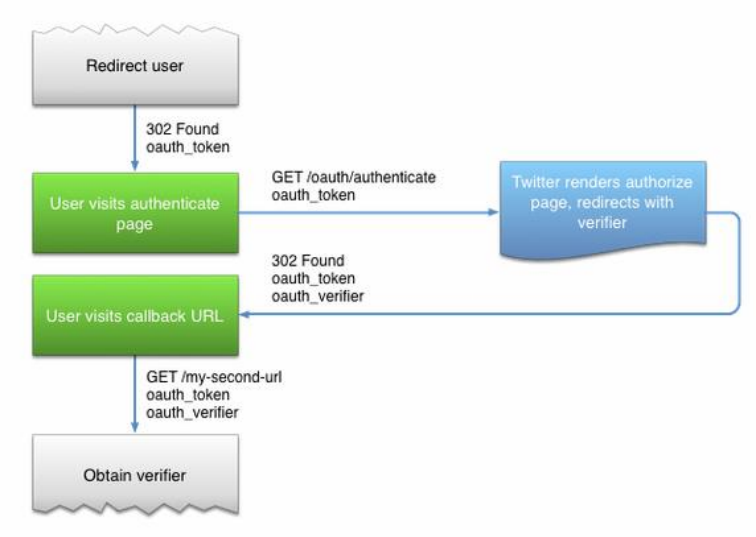


Figure 11: Redirecting the user

Firstly, the client must include the request token which is returned by server in the step 1 and navigate resource own to authenticate page (login page) by using HTTP 302 redirect. Next, the resource owner is required to provide valid username and password. According the resource owner's status, there are three possible cases:

- Signed in and approved. In case resource owner has signed in and approved the application already, this step will be skipped and the server will automatically return the valid access token to callback URL immediately.
- Signed in but not approved. If resource owner has signed in, however, he/she has not approved the application yet (as mentioned above, the resource owner is required to approve the client only one time), the authentication is displayed for user to grant and manage application's access. After that, the request token will be returned.
- Not signed in. For the new resource owner who has not signed in and approved the application, login page is prompted in order to enter the username and password. Next, the flow will be as same as "Signed in but not approved" status.

Here are the authenticate pages :

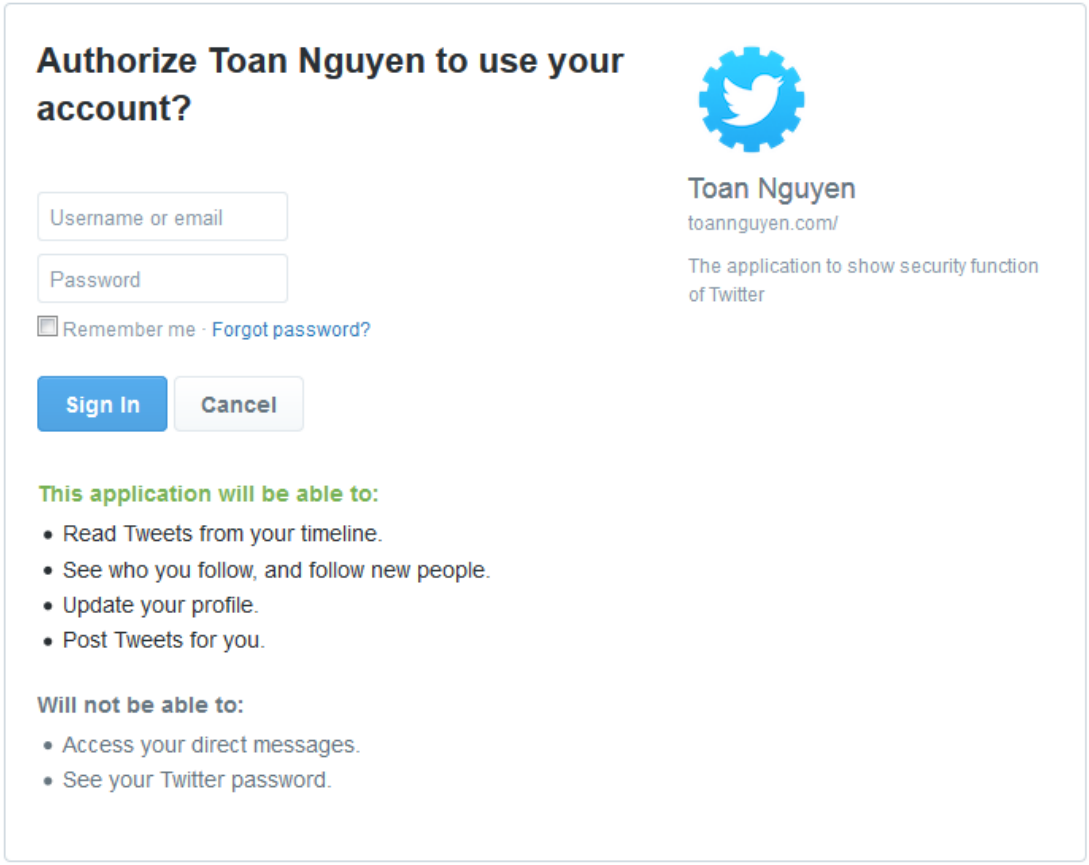


Figure 12 : Twitter authenticate page

Step 3: Converting the request token to an access token

After step 2, resource owner issued the right to application already. At this moment, the application is able to access to resource and retrieve data, however, there is one more step to finish. It is obtaining the access token by validating the request token. This step aims to notify the server that resource owner has allowed the application's access.
Here is the example request for access token:

```
POST /oauth/access_token HTTP/1.1
User-Agent: themattharris' HTTP Client
Host: api.twitter.com
Accept: */*
Authorization: OAuth oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
                     oauth_nonce="a9900fe68e2573b27a37f10fbad6a755",
                     oauth_signature="39cipBtIOHEEnybAR4sATQTpl2I%3D",
                     oauth_signature_method="HMAC-SHA1",
                     oauth_timestamp="1318467427",
                      oauth_version="1.0",

oauth_token="NPcudxy0yU5T3tBzho7iCotZ3cnetKwcTIRlX0”

Content-Length: 57
Content-Type: application/x-www-form-urlencoded

oauth_verifier=uw7NjWHT6OJ1MpJOXsHfNxoAhPKpgI8BlYDhxEjIBY
```

The application has to send a POST request to authentication server for upgrading request token. In this request, there are two arguments which once again illustrate high security of Twitter implementation:

- oauth_token: The request token which was returned in step 1. The application must clarify which token is using for communicating with server on behalf of resource owner. However, what if another application fakes the request token and sends it to server on behalf of the application, what if the server does not check the validity of that request token and send back the valid access token to the sophisticated application. Coincidentally, authentication server just reveals its user credential by itself. That is the reason why every single request must include the second argument.
- oauth_verifier: This is the returned value is step 2. This value aims to notify server about the validity of the request token. This value will be returned only after resource owner has accepted the application. The oauth_verifier in the request must match the oauth_verifier which was stored in the server after step 2. As a result, the sophisticated request cannot be authorized without the valid oauth_verifier.

After retrieving access token, the client can communicate with server by that token on behalf of resource owner to collect data, create object and even delete resource if possible as long as resource owner has accepted that client.

In general, the Twitter implementation about authenticating application was analyzed. Currently, Twitter is applying 3-Legged OAuth 2.0 which stands for the resource owner, the server and the client (application). In this model, the resource owner must qualify himself/herself with server by providing valid username and password. After that, he/she needs to empower the application to interact on his/her data. Finally, the server issues an access token to the application and it can be used on behalf of the resource owner. So far, we have clarified the relationship, dependencies among resource owner, server and the application. In the next section, we continue to research the conditions in communication between the application and server.

4.2. Authentication & Authorization in Twitter API calls

The application could call many API requests to Twitter server to retrieve, delete and create resources. On the one hand, this could bring Twitter closer to public because of the huge information resource and its flexibilities. On the other hand, this could be the biggest risk for Twitter since there could be million requests from million applications every day. Therefore, there are two outstanding burdens which Twitter are facing:

- How can Twitter authenticate the application and its requests?
- How can Twitter secure the sensitive including in the request such as access token, access token secret, customer key and customer secret ?

The key answer was introduced above in section 3.2.2. Twitter is applying Public-key Infrastructure partially to protect the sensitive data which is sent back and forth between the client and the server. In order to witness this function, let’s research which

steps are required to send and read the request in both client side and server side, in case the application sends the request to tweet a new tweet to Twitter server:

- ***The Client Side***

The raw data is not supposed to be sent from client to server since the other application is able to listen and catch those secrets. Therefore, before sending the request, the application is required to hide the clarity of the message by using hash algorithm. The overall flow can be explained by Figure 13.

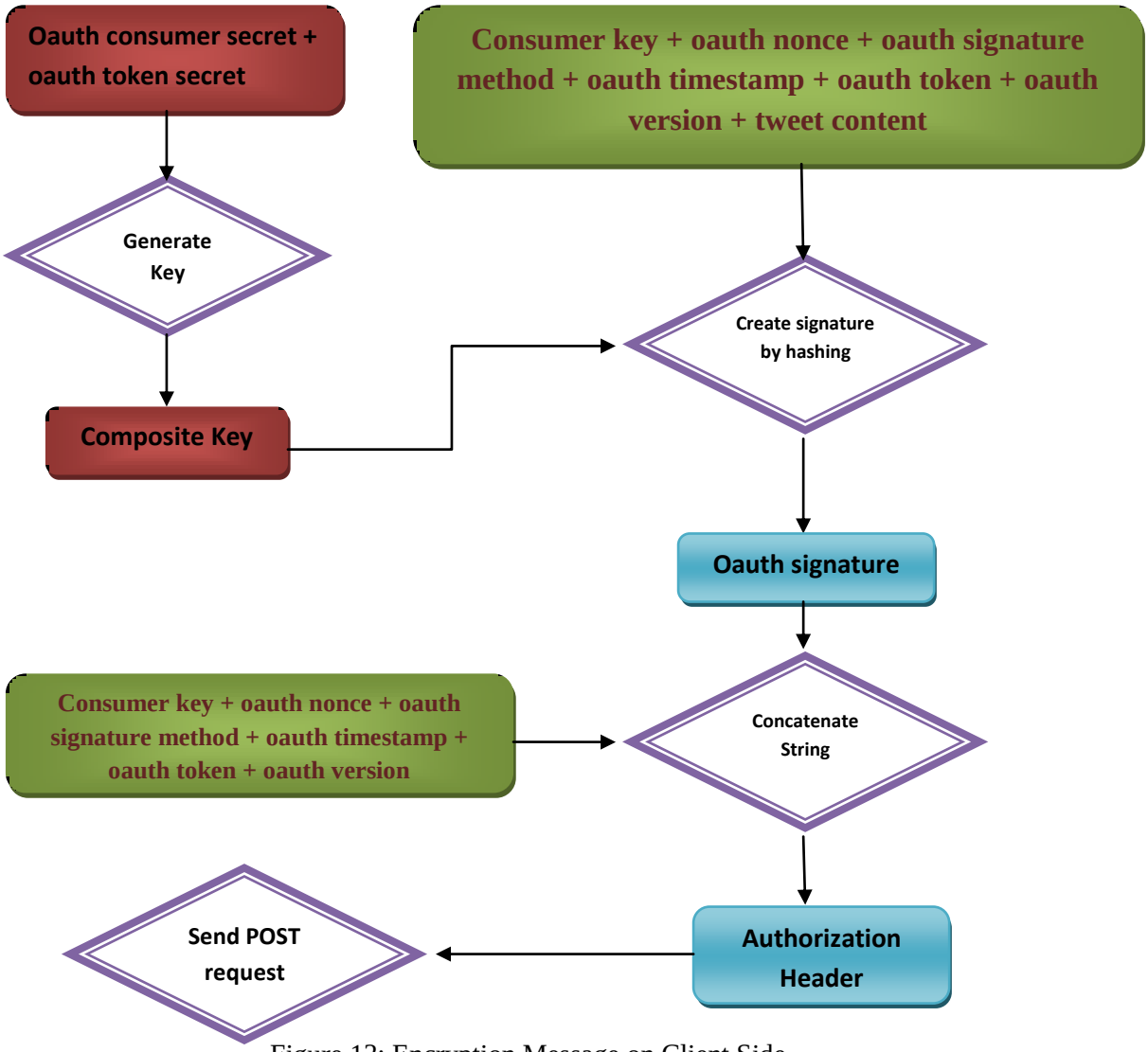


Figure 13: Encryption Message on Client Side

As can be seen on above figure, at the beginning, the application needs to calculate a composite key from two private keys, the OAuth consumer secret and OAuth token secret. Those two keys are supposed to be confidential. Next, the application needs to create the signature by hashing the message, which is concatenated from many arguments such as consumer key, oauth nonce, oauth signature method, oauth timestamp, oauth token, oauth version and the value of request (introduced in section 2.2.2) in this case is the tweet content. Finally, the oauth signature will be put in the Authorization Header in the HTTP POST request with other arguments and send to the Twitter server. Here is the example of the POST request from an application:

```
POST
/1.1/statuses/update.json?status=New%20Tweet%20from%20API&display_coordinates
=false HTTP/1.1

Authorization: OAuthoauth_consumer_key="DC0sePOBbQ8bYdC8r4Smg"
                oauth_signature_method="HMACSHA1"
                oauth_timestamp="1447763195"
                oauth_nonce="2757352841"
                oauth_version="1.0"
                oauth_token="4204335839-
        JKlJnO249TnMegiDsJaqDkxgDyDPizQQIi8nVO7"
                oauth_signature="zkKRtFwatfH8qjn%2FvxQnhnhyowY%3D"
Host: api.twitter.com
```

Code Snippet 3: Request Example of Tweeting new Tweet

There seems to be some duplicated arguments in this process such as consumer key, oauth nonce, oauth signature method and so on. However, in section 2.2.4 according to the Stateless, all essential information for that request must be included inside the request. Therefore, each argument plays a different role in decrypting and authentication process such as:

- oauth_consumer_key: notifies server which application is sending API call.
- oauth_signature_method: the hashing algorithm which was applied to compute the signature. The authentication must aware of this method since the server will calculate the signature once again by itself to validate the signature.
- oauth_timestamp, oauth_nonce, oauth_version: These three arguments was used to generate the signature, thus, the server needs to collect them. Moreover, as mentioned above, these arguments help to prevent the plaintext attack.
- oauth_token: the valid token which as represent for the application's right. The authentication server qualifies the API calls from the application by this token.
- oauth_signature: the encrypted message from client. This message illustrates what the application wishes the resource server to implement.

In this case, Twitter is applying the PKI model, thus, there is a pair of key which is using in authentication process. Despite of different name, the key (public key) and the secret (the private key), they still play the similar role. However, there is a major change in the encryption process. Instead of encrypting whole message, Twitter only requires the message is hashed by using HMACSHA1 algorithm using the key which was calculated from two secret keys. The main aims of this step are to:

- Hide the explicitness of the message. Obviously, sending the raw information back and forth between client and server is never a good solution since the hacker can listen to your call, thus, Twitter asks the application to encrypt those confidential information by secret keys.

- Address the validity of the request. As can be seen on the code snippet 3, some arguments such as oauth_token, oauth_consumer_key are passed directly and send straight to server without encrypting. Therefore, the signature plays a role as the certification for valid request. Because the signature was computed by the two secret keys, even though the oauth_token, oauth_consumer_key are distributed commonly, the other application is not able to fake the request without the valid signature. Because of this reason, the other arguments are not encrypted and transparent.

♣ ***The Server Side***

According the PKI model rules, after receiving the request from application, the server is going to validate the request by comparing two values of the signature. The first one is the original signature in the request and the second one is the result after calculating the signature by the server. This section will explain how the server calculates the signature by itself. This section will reuse the code snippet 3 as the example.

Firstly, let's take a look at the request URL:

POST
/1.1/statuses/update.json?status=New%20Tweet%20from%20API&display_coordinates=false

As defined in the section 2.2.2, the server separates the request value from URL. In this case, the value is the new tweet's content and it is "New Tweet from API". Back to Figure 13, in order to generate the signature, the server will take the tweet content from the URL and the other arguments in the request. At this stage, the server computes the key from two secret keys based on the oauth_consumer_key and the oauth_token in the request. The interesting point is that Twitter server saved all public and private keys of all applications, thus, the server is able to calculate the key without asking those secret keys from client. Finally, the server compares the signature from the request and the signature which was generated by itself. If the two signatures are the same, the server will automatically execute that request and reply the response to client, otherwise, the server does not follow that request and return an error message.

The authentication process on the server side aims to:

- *Prove the validity of the request*. By testing the signature, the server is able to double-check is the request sent from the authorized application.
- *Check the application's right*. By the oauth_token, HTTP method and the message, the server could check whether the application is allowed to trigger that request. If the application has not granted the access yet, the server will returne an error message without executing that request.
- *Check the reliability of the message*. The server needs to check the message to ensure it is the original message from client without any modifying or changing by any third party. Hash algorithm is one-way encryption and in this case the message was hashed by using the two secret keys, so, it is almost impossible for another application to fake the signature and the message at the same time. In

case the message was faked, when the server calculates the signature and compares two values then those two values cannot be the same and that request turns out to be invalid.

Overall, in order to protect the resource owner, Twitter expects their clients to issue the right to application by using 3-legged authentication. After that, the application passes the token, which is represented for its right, to the request and communicate with the server by that token. The application is able to work on those resources which the resource owner would allow it to interact on. In order to secure the sensitive information which is sent back and forth between client and server, the application must generate the signature from the message by hashing it with two secret keys. The obvious value of message should not be sent straight to server to avoid leaking the top secret information out.

In the next chapter, the security function of an API Provider will be carried out. Let take a further look on how the user details, such as bank account and security number, are protected while making transactions. The next chapter will use the Stripe, an online payment provider, and its services as an example.

5. Online Payment API Security Methods

On the one hand, this thesis has gone through quite many terms and definitions from RESTful Web Services, Oauthentication and Twitter security. On the other hand, there is one more emergent field, which gained much attention recently. It is e-payment. Because of the globalization and industrialization, people, especially the Youth, would rather like to spend more time on online purchasing than to shopping in the traditional way. This trend seems to have no end with the support from mobile devices and some online payment services such as PayPal, PayByWay and Stripe. With one click, user is able to make a transaction at anytime, anywhere with any device. However, this could be a burden for the service provider since they have to prove the ability of protecting the confidential information. This chapter will focus on Stripe, an online payment service, to witness how the transaction details are secured during payment process.

5.1. Stripe Security Introduction

Strip offers clients various ways to make a transaction by credit and debit card including Visa, MasterCard, American Express, Discover, and JCB. Same as other service providers, Stripe also follows Payment Card Industry Data Security Standard (PCI DSS), which was developed by Payment Card Industry Security Standards Council, to protect the clients. This is the set of requirements which play a role as the instruction for protection and security of their cardholder information. For all organizations that save, process and transmit the cardholder data, this model is compulsory in order to secure card transactions.

The main aims of this model are to avoid spreading customer details across the private, public networks and reduce the large amount of online theft, fraud and any financial damage to both clients and the provider as well.

In order to implement the framework of PCI DSS, there are twelve basic requirements which were introduced below [PCI, 2006]:

Table 2: Payment Card Industry Data Security Standard

No	Requirements	Category
1	Install and maintain a firewall configuration to protect cardholder data	Build and Maintain a Secure Network and Systems
2	Do not use vendor-supplied defaults for system passwords and other security parameter	
3	Protect stored cardholder data	Protect Cardholder Data
4	Encrypt transmission of cardholder data across open, public networks	
5	Protect all systems against malware and	

	regularly update anti-virus software or programs	Maintain a Vulnerability Management Program
6	Develop and maintain secure systems and applications	
7	Restrict access to cardholder data by business need to know	Implement Strong Access Control Measures
8	Identify and authenticate access to system components	
9	Restrict physical access to cardholder data	
10	Track and monitor all access to network resources and cardholder data	Regularly monitor and test networks
11	Regularly test security systems and processes	
12	Maintain a policy that addresses information security for all personnel	Maintain an Information Security Policy Maintain a policy that addresses information security for all personnel

However, this thesis focuses only on two requirements, number 3 and 4, in the "Protect Cardholder Data" category to analyze how cardholder details are protected.

PCI DSS Requirement 3, Protect stored cardholder data [PCI DSS Compliance Requirement 3, 2006]: This requirement proposes that every saved cardholder information should be protected at all levels. Below guideline instructs the provider to follow this concept:

- Maintain cardholder data storage to a minimum by applying data retention and disposal policies, procedures and processes that include at least the following for all cardholder data (CHD) storage (frequently check and delete invalid data).
- In order to avoid leaking out sensitive information, service provider should not save any sensitive authentication data after authorization even though it was encrypted.
- Hide the transparent PAN value when displayed (the first six and last four digits are the maximum number of digits to be displayed). Only the authenticated parties could see the full PAN value.
- In case the PAN is stored in server side, it should be encrypted by using hashing, truncation or strong cryptography with associated key-management processes and procedures.

PCI DSS Requirement 4: Encrypt transmission of cardholder data across open, public networks [PCI DSS Compliance Requirement 4, 2006] : this requirement aims to discuss the safe transmission of cardholder data across the public network, which has a low security. According to this requirement, the service provider should:

- Use effective cryptography and security protocols (for example, SSL/TLS, IPSEC, SSH, etc.) to safeguard sensitive cardholder data during transmission across open, public networks.
- Never reveal unprotected PANs by messaging technologies (for instance, e-mail, instant messaging, chat, etc.).
- Ensure that all security guidelines and operational procedures for ciphering transmissions of cardholder data are recorded, in use, and known to all affected parties.

Overall, Stripe is applying this framework to secure their client's credentials. In the next section, Let us experience how Stripe supports another applications, which are using its service to make transactions, to overcome the challenges of protecting cardholder details and follow the PCI guidelines.

5.2. Stripe Security Systems

The power of online payment becomes more and more powerful. Catching this trend, many Integrated Websites publish their services, which support clients to purchase and make a payment. However, according to PCI DSS Requirement 3 (Keeping cardholder data storage to a minimum), the Integrated Website should not store cardholder data on their side. Thus, Stripe will take care of the hardest parts of PCI compliance by providing processing user data service. The figure 14 illustrates this workflow [Stripe, 2015].

This model not only passes the burden from Integrated Website to Stripe server, reduces the workload for client side, but also significantly increase the security by following strictly PCI guideline since the Stripe gets involved in encrypting cardholder details.

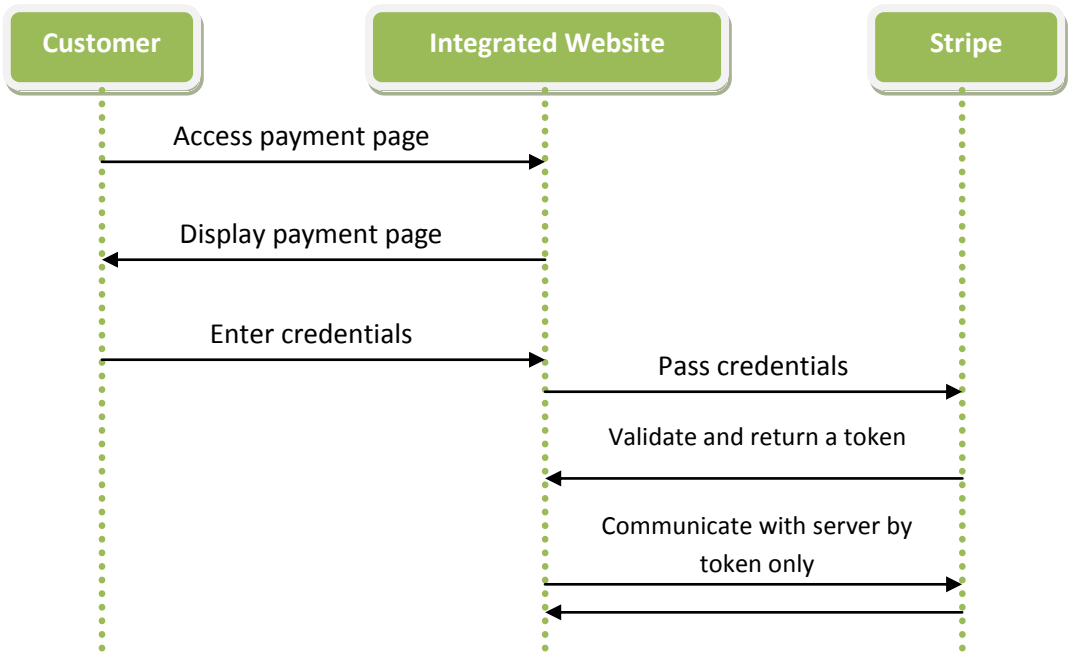


Figure 14: Stripe processing cardholder data workflow

As can be seen from figure 14, firstly, customers need to provide their card information to make a payment after accessing payment page and selecting products. Next, instead of storing those credentials and using them directly to make transaction later, the application should navigate them to Stripe server. At this stage, Stripe server validates the inputs and returns a valid token in case those credentials are valid, otherwise an error message will be returned. Finally, the application has to use this token to make API calls for charging on that account. By following this model, the cardholder details are secured properly because:

* The token is single-use token. According to PCI DSS Requirement 3, the service provider should not store any sensitive information even though it was encrypted, and in order to obey this rule, the token is able to use only once time. It means that saving this token in somewhere is not obviously high usability for long term.
* The application communicates with server via the token instead of the transparent cardholder details. These top secret credentials are always the final target of the hacker, especially, they are passed across the public network. Therefore, this information will be automatically convert to a token and this token is the only value which is transferred back and forth between client and server.

Currently, we have gone through how Stripe cooperates with another Integrated Website to follow the PCI compliance. In the next section, we will analyze how the request can qualify itself to the server by the token and how the server validate the application.

5.3. Stripe API Authentication

As explained from Figure 14, the Integrated Website communicates with Stripe server via a single-use token. Therefore, the application is able to verify itself by that token. Let us take a further look at the mechanism of generating token.

One token is for everything

Currently, Stripe is following the Public-Key Infrastructure (PKI) framework. It means that each of application is provided a pair of keys, after registering successfully. In order to generate the token, the application is required to include its public key to Stripe server. Here is an example of the request:

```
<form action="" method="POST">
  <script
    src="https://checkout.stripe.com/checkout.js" class="stripe-
button"
    data-key="pk_test_hh33zEPnHp5ZCsWOZowo3Jb4"
    data-amount="2000"
    data-name="Demo Site"
    data-description="2 widgets ($20.00)"
    data-image="/128x128.png"
    data-locale="auto">
  </script>
</form>
```

Besides of other cardholder details such as card number, cardholder name, security number, the public key of the sender, the data-key value in the request, should be sent to server. After validating the validity of the inputs, the server will generate the token for that request. It would lead to:

- *Dynamically generate token*. Because the token is formed by the public key, thus, this system allows multiple generating token applications and center processing token application.
- *Avoid the fake request*. According to the PKI framework (section 3.2.2), the message, which was encrypted by public key, can be decrypted by private key only and vice versa. In this case, the token is issued and encrypted partially by the public key. As a result, the only key, which is able to decrypt the token, is the private key of the application. This is blocking another application to sophisticates the request on the behalf of authenticated clients.

However, as discussed above, the public key is could be distributed widely and globally. Thus, anyone could obtain this key, send the request and use the token for charging. This sounds that this system is not secured and safe at all since anyone, any application could send the request to generate the valid token. The answer is yes, it is true that everybody, those who has the public key could ask Stripe server to issue the token. However, the centered point is that anyone could send the request to generate token but that token is valid only for the one application, which owns the private key. Let us examine an API call to create a charge to experience how the token can be used limitedly. Here is the example of an API request:

```
curl https://api.stripe.com/v1/charges \
   -u sk_test_BQokikJOvBiI2HlWgH4olfQ2: \
   -d amount=400 \
   -d currency=usd \
   -d source=tok_17BFnz2eZvKYlo2C7zOjlWeu \
   -d description="Charge for test@example.com"
```

Similarly, this example could be seen as the RESTful URL as this link:

POST
https://api.stripe.com/v1/charges?key=sk_test_BQokikJOvBiI2HlWgH4olfQ2:&amount=400&currency=usd&source=tok_17BFnz2eZvKYlo2C7zOjlWeu
&description=Charge for test@example.com

As introduced above in section , this could be interpreted that the request is sent from the application, which has the private key is "sk_test_BQokikJOvBiI2HlWgH4olfQ2:", in order to create the charge object in "api.stripe.com" server. This charge object costs 400 dollars and it will charge on the card, which was defined in "tok_17BFnz2eZvKYlo2C7zOjlWeu" token.
Each of single argument in the request plays the different roles. If the amount, currency and description supplement details for the request, there are two arguments are quite important in the security perspective:

- Source: A payment source to be charged, such as a credit card. In this case, the payment source refer to the valid single-use token. Just imagine that instead of sending transparent cardholder details, they were wrapped in the package with beautiful cover and send that package to recipient. That package is the token. By examining the token, the server is able to aware of which card this request wishes to charge on. The point is the package is encrypted now. It needs the key to unbox it and that key is the second argument in the request.
- Key: the secret key, the other key in a pair of keys which the public key was used to generate the token. This token not only supports server to decrypt the source to obtain the essential card information, but also plays a role as the certification for the request. However, the token could be requested from various machines and devices, so, how server can restrict some accesses from unauthenticated applications. The solution is this argument, by providing the correct private key, server can decrypt the token successfully and it means that this request was sent by the valid application. Different from another security systems, Stripe relies mostly on the encryption algorithm based on the PKI framework to secure the cardholder information. Thus, the private key plays a huge role overall.

In the general, Stripe server now is following the PIC guideline to build up the security system. Instead of sending transparent customer data directly to server, the information must be encrypted and wrapped into a token and this token is the only value, which is passed back and forth between application and server. By this way, user credentials are avoid to leak out accidentally.

So far, some security systems such as Twitter and Stripe have been gone through to witness how the API provider could protect their client data during the API calls. However, there are some challenges to bring this system closer to public because of following reason:

- *Security is never an easy field for everyone*. Basically, any current security system of API Provider is developed on the existing theory or framework, so, applying this system efficiently requires service consumer should be aware of those theories in advance. However, not every user has ability to receive and understand this kind of knowledge.
- *Long and complicated process*. As analyzed above, especially Twitter requires application to follow quite many steps from hashing a pair of keys to encryption the signature. After processing the main concept of security system, now user is struggled with encryption algorithm. Moreover, hashing process requires various arguments and each of them must be in correct format.
- *Technical skills*. Besides of having knowledge of the main concept, user needs to have sufficient technical skills such as sending HTTP Authentication request, JSON deserialization. The returned result for each API call could be in JSON or XML format based on API Provider. Therefore, user can read all values in the result by naked eyes but in case, API consumers need to get those values automatically for some purposes such as to use as input for another API calls or to display automatically on the website, then they are required to have knowledge at JSON and XML.
- *Ambiguous responses*. The returned errors from Web Service are quite obscure. For example, Twitter will return error code 400 (Bad Request) and their explanation for this case is “The request was invalid or cannot be otherwise served. An accompanying error message will explain further. In API v1.1, requests without authentication are considered invalid and will yield this response” [Twitter Developer, 2015]. However, there are thousand cases could be happened. Therefore, users need to double check their API call request line by line to find the root cause, maybe in some cases the problem is missing a comma.

Therefore, in the next chapter, chapter 6, my own website and a C# library will be proposed to overcome these challenges. The main aims of this website is to support clients those who do not have much knowledge and skills about the relevant fields as mentioned above. Mostly, the target audience is the manufacturers who wish to run their e-marketing services more stability and efficiently.

6. Implementation and Demo

Catching the challenges which were mentioned above, my own website, which intuitively illustrate what I have researched, is proposed in this chapter. It also assists users, especially the manufacturers, to efficiently and automatically publicize their marketing campaigns via social networks. Moreover, this website offers purchasers to make transactions for purchasing products which were published before. Here is the main workflow of the website:

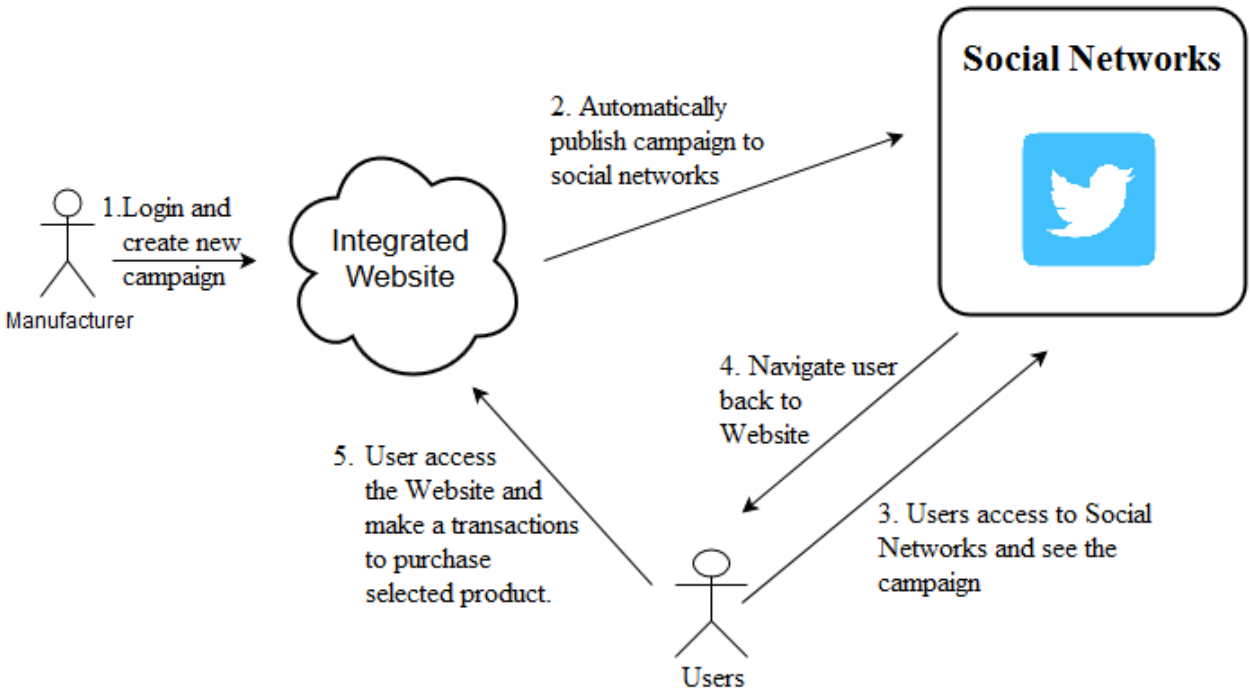


Figure 15: The main website's concept

The specific aims of this website are to help clients to:

- *Promote their products for round-the-clock*. By using this website, manufacturers are able to post a new tweet on their own page to introduce their new services, products anytime even though when they are not logged in their account or are not online.
- *Schedule their campaign automatically*. Nowadays, with the strength and speed of the social networks, the manufacturers have various choices to publish their campaign. A manufacturer could have several social pages to promote their services and product. However, this would lead to a challenge for both manufacturer and manager since they need to remember exactly when they should publish which campaign on which page. This problem could be solved by using this website. Manufacturers are able to schedule when their campaign will be published and terminated when creating the campaign. The integrated website will take care of the rest of work.

- *Overcome the technical issues*. The website is simple and easy to use. In manufacturer's perspective, understanding intensively API framework and background is not a top consideration, thus, this website is the bridge between user those who does not wish to spend any effort on technical skills and the API Providers who require users to follow their instructions strictly.
- *Manage campaign and social posts efficiently*. Because the application is able to access several social accounts, the various posts from different accounts could be managed at one central page, in this case it is my website.

In the next section, the technique and tool which were used to develop the application will be discussed.

6.1. Technical Introduction

Structure:

Currently, this website is divided into three main layers: The interface (MVC model), the services and the database which are illustrated in the figure below:

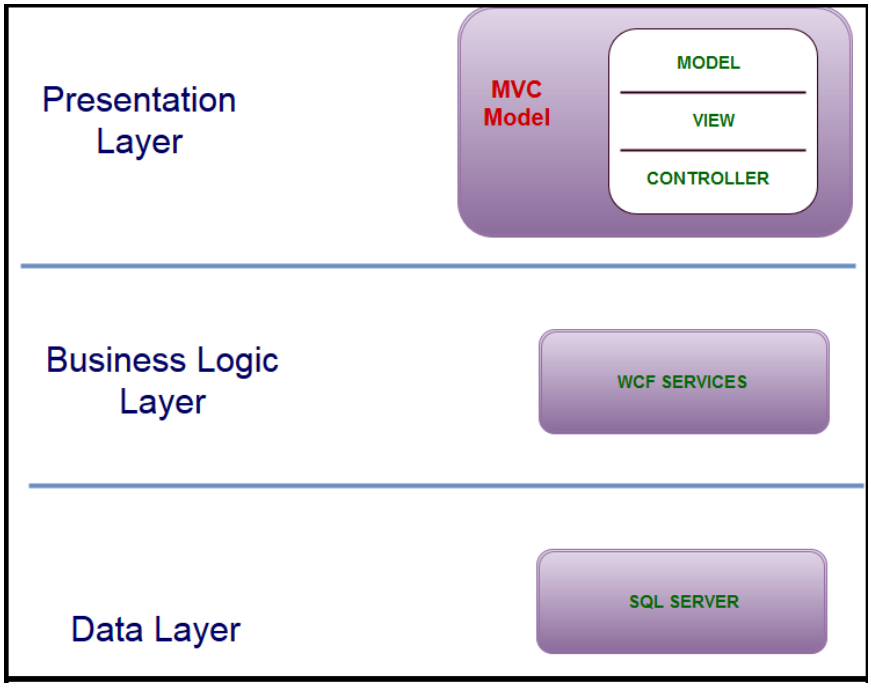


Figure 16: The application structure

MVC model: According to Krasner and Pope [1998], "*Model-View-Controller (MVC) programming is the application of this three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (the controller)".*

WCF Services: Windows Communication Foundation (WCF) is a framework for building service-oriented applications. WCF services assist to optimize sending data from one service endpoint to another service. This would make the communication in application easier. The message could be as simple as text, binary data or complicated objects [Microsoft, 2014].

Database:

Currently, this application is using SQL Server 2008 as relational database management system. The database is illustrated by the diagram below:

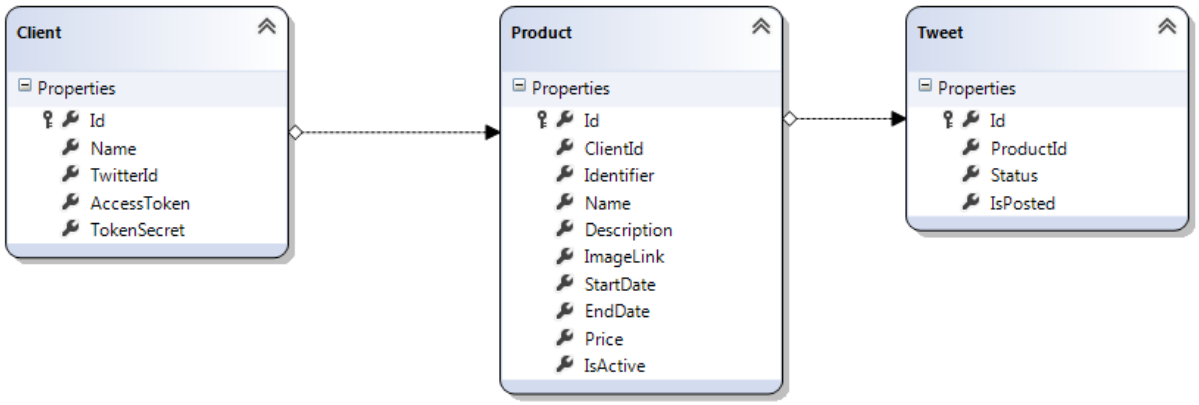


Figure 17: Database diagram

There are three main roles as introduce below:

Client: The manufacturer and Twitter clients. This website does not require user to register in order to use the services. Twitter account will be reused in this case. Therefore, user is required to have a Twitter account beforehand.
Product: The product which will be published in Twitter and purchased using the website. Each product is required to have a unique Identifier. This argument allows users to determine which product they wish to purchase.
Tweet: The Tweet for each product. As a part of the service, the manufacturer is able to promote their products using social networking applications, in this case it is Twitter. Thus, there is a need to have a single tweet for each Product.

Development Tool:

Microsoft Visual Studio Professional 2013 is used as the development environment.

6.2. Security Features Implementation

- ***Login***

Following the Social Login concept, this application re-uses Twitter account and this is also the outstanding feature of an integrated website. This function puts the burden of protecting user information out of the website since the user does not register and create any account in this website. Moreover, Twitter services, such as verifying users, protecting user credentials and high security features, are reused once again. It not only helps this website to be hand free from the security problems, but also increases its performance since this website does not need to spend much space for storing user's details. This feature is described in figure 18:

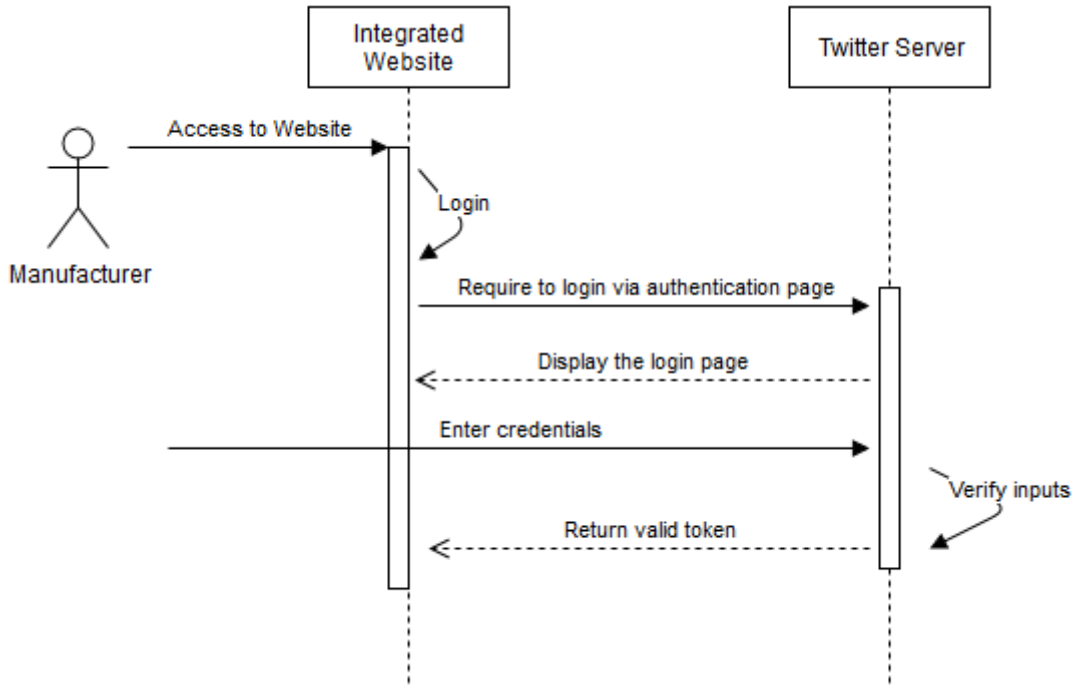


Figure 18: Login function

Although the result of this step is the valid token; however, the main aim is not to obtain that token but to verify the user. As noted above, after granting the valid token, that token could be valid as long as the resource owner has not revoked the access. The final argument at this stage is the Twitter ID and this is also the key to identify the user.

- ***Creating new campaign***

After obtaining the Twitter ID, the application queries all the existing products for the specific Twitter ID only. This would lead to avoid side effects for another customer's products and accidental modifications. The figure 19 briefly describes the workflow of the way how a manufacturer creates new campaign and the relationship between Integrated website and Twitter server.

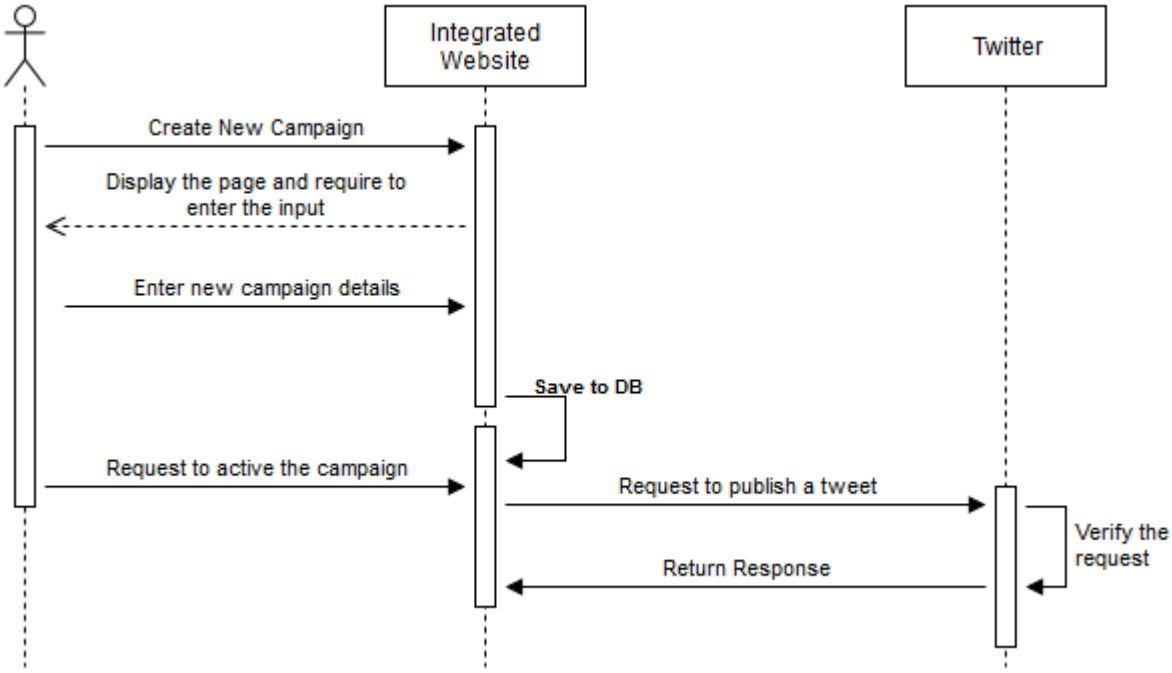


Figure 19: Creating new campaign

Firstly, producer access to creating new product page (at this moment producer must be logged in successfully already). At this page, user is able to create new campaign by entering all the essential information for that product such as product name, product identifier, price, start/ end date and image link. Next, the application will save the information of the new product into the database. There are two options at this time:

* *Automatic publishing*. According to the start date and end date, the post application automatically activates that campaign and immediately posts the new tweet in Twitter for that product even though the account owner is offline.
* *Manual publishing*. In case, producers wish to publish their product immediately before the start date. There is no need to modify the products details. User is able to activate it by a single click.

Obviously, this application communicates with Twitter server to post the new tweet by a valid token in both cases. After activating, "IsActive" property will be changed from "False" to "True". This would help user to manage their product more efficiently.

As can be seen on Figure 19, there is an "Activate Now" button for those inactive products. This is Manual publishment function and the website immediately posts a new tweet on logged-in account as example.

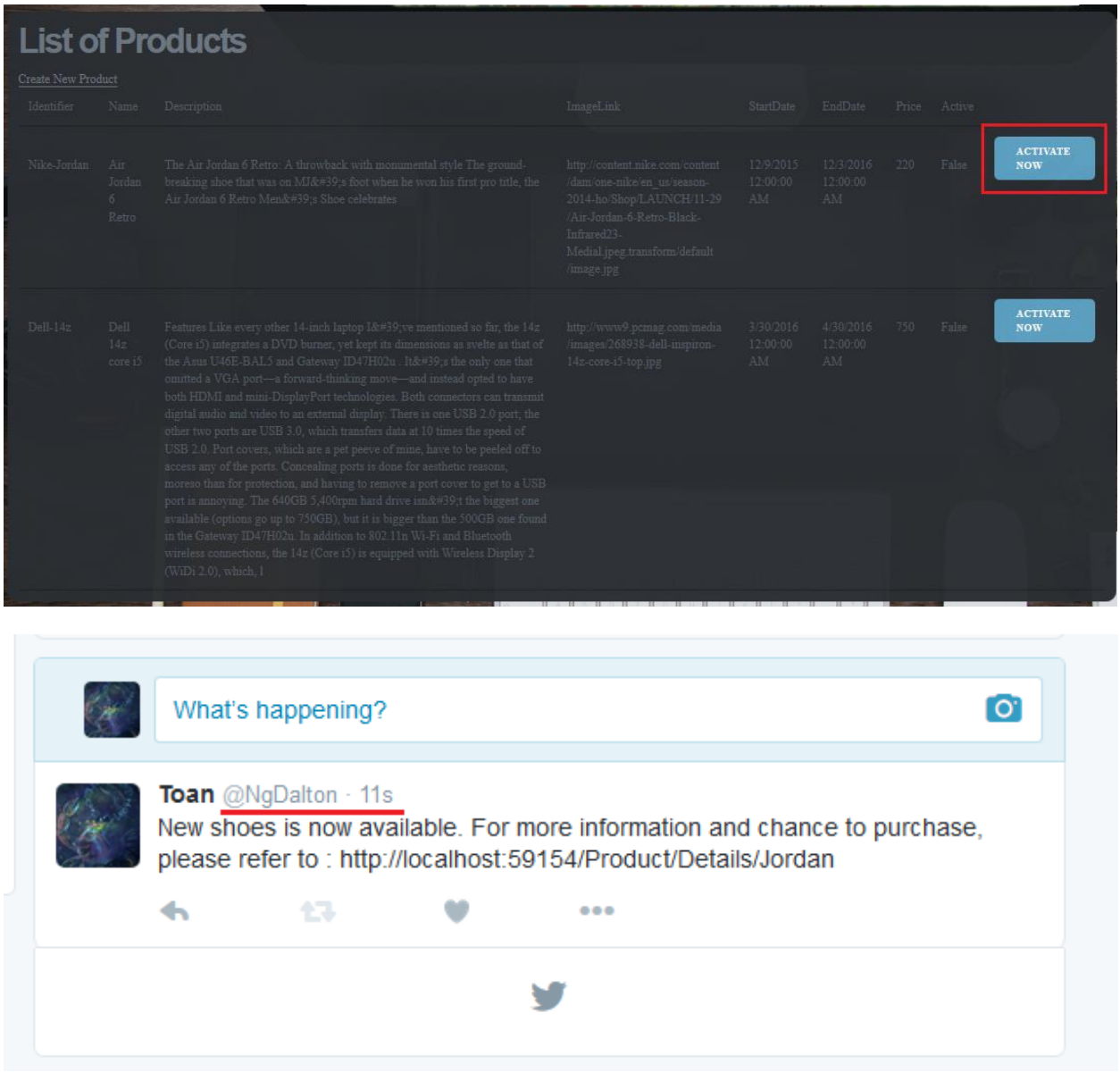


Figure 20: Example of creating product and publishing tweet

Moreover, this application checks the expiration date for each product in order to automatically deactivate the campaign. This function would benefit significantly to the manufacturers, especially big companies. Usually in big companies, there are hundreds of campaigns every month and the managers struggle with reminding themselves when they need to deactivate expired campaigns or to activate new campaigns. With the support from this application, the manager can keep their mind free of this challenge. They only need to configure the start date and end date while creating new product, then, the website will take care of the rest work.

- ***Purchasing products***

In this scenario, Twitter plays a role as the bridge between the customers (the Twitter users) and the Integrated website. After publishing the campaign in Twitter, customers are able to be aware of new product release and able to access the website via the attached link in the tweet. Figure 21 gives more information for this process.

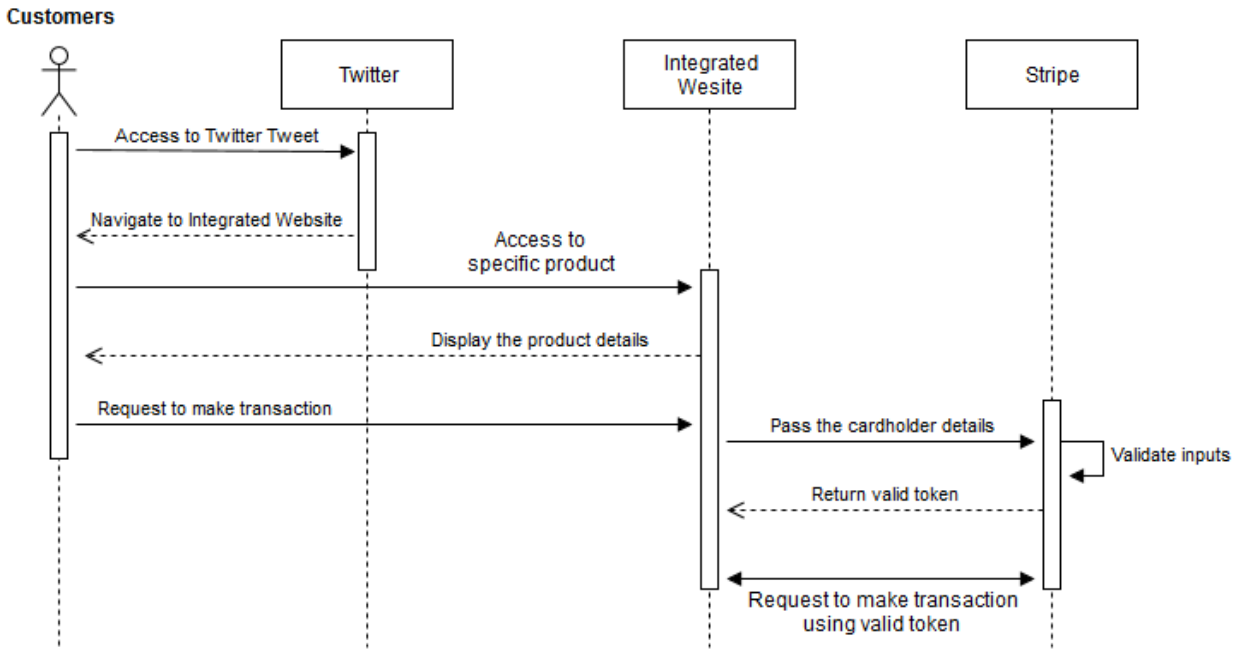
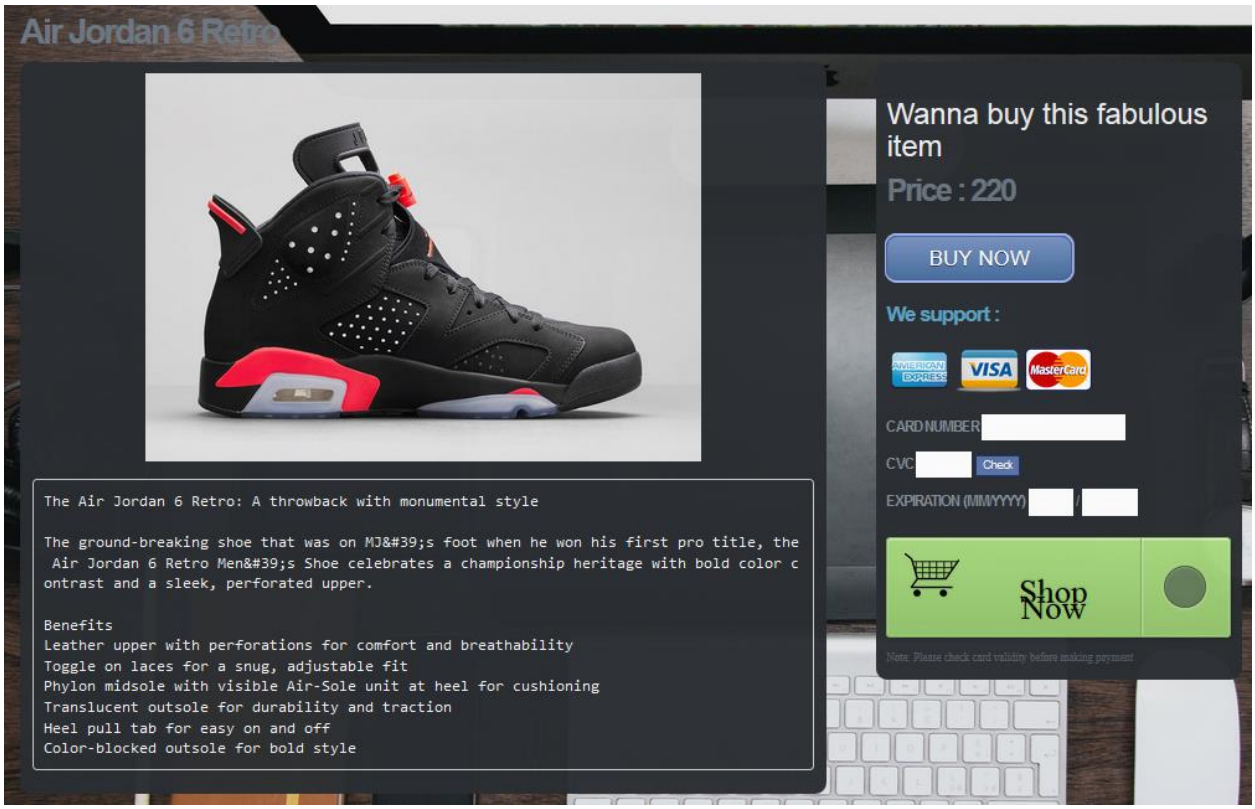


Figure 21: Making transaction flow

After navigating to specific product from Twitter, user may have more information about the product from the Integrated website and purchase it directly from the same page. At this time, purchaser needs to enter the card details such as card number, security number (CVC) and the expiration date. Next, The Integrated Website passes those credentials straight to Stripe server. This would widely reduce the burden for the application since it spends no extra effort to protect and verify that information and it follows the PCI guidelines in section 5.1. At this stage, Stripe server is responsible for validating the inputs. If the inputs are invalid, the Stripe server returns the error message otherwise a valid token is sent back. Finally, the website communicates with the Stripe server by this valid token and this token is single-use token. This would help the website to reduce to its storage since there is no need to save that token value. The example in Figure 22 show intuitively the Details page which was integrated the Stripe services at right banner. At this page, purchaser is able to see more details of the product which were published on Twitter already and purchase the selected product easily and directly without navigating to another page.



Firgure 22: Example of product details page

In the example above, there are three textboxes for cardholder details. These are also required inputs in order to make a transaction via Stripe services. The most outstanding feature of this application is ease transaction. Purchaser only need to enter their card information then click on “Check” button. At this stage, the website passes cardholder credentials to Stripe and waits for the answer (the token). Finally, purchaser must confirm the transaction by clicking on “Shop Now” button. This time, the website once again communicates with Stripe server by a valid single-use token to complete the transaction.

By collaboration, this website exploits the power and speed of third parties services in order to improve the quality and reduce the workload of my own services. The manufacturers now are able to promote their own products on social networks fast and securely. Moreover, they can earn more possibilities to bring their own goods over around the world since the website offers users to make payment in some simple and quick steps.

In the next section, this website is evaluated by comparing to some similar existing applications in the market. The functions, target will be highlighted to emphasize the potential of the application so far.

6.3. Result of the demo

Target customer

Currently, this application does not restrict any kinds of client; however, it is optimized for the small manufacturers and start-up companies. The reasons why they are ideal target for this application are:

- *Limitation in resources*. The small companies do not have powerful resources in both finance and human to boost their business. Because of this, they are forced to reduce their expense as much as possible but still maintain high quality of the products and services. As the result, this application could be right solution for them since this will cut off reasonable amount of cost for marketing and management campaign. Manufacturers could focus on improving their services instead of spending extra effort on another fields.
- *Lacking of skills*. Usually, start-up companies focus on specific field to develop their own business. This is understandable since they could not have possibility to master all fields. For example, a start-up company which provides services for clients to search and locate the popular restaurant around them. That company would rather to research how they can improve their services by collecting more information about the restaurant, locating restaurant more exactly and obtaining as many review of those restaurants as possible. Therefore, this application offers them a chance to get hand-free of marketing and payment process but still maintain the integrity of whole system.
- *Newly forming process*. Small companies usually do not have certain processes since they do not have many products and service to offer. Therefore, they would prefer the pre-made processes to apply immediately. As mentioned in Figure 19, the website will automatically manage all campaigns according to the configurations when creating the new campaign. This would benefit the start-up companies significantly since all steps are automated.
- *Timing*. Whenever companies are ready to publish their products and services to the market, they wish to proceed it promptly. With the live stream functions from Twitter, once the tweet is published, it will be spread over the globe and Twitter's clients, regardless the location, are able to reach those products immediately. Having support from Stripe, the users just have more chance to own the products quickly.

Overall, the application aims to reduce the workload for the manufacturers by providing pre-made processes. This website is expected to take care various aspects of business from marketing to sale. Therefore, small manufacturers are the target clients instead of the big companies which have clear processes and partners to help them develop their own services.

Platform

This is a web application; thus, it would be operated on various browsers and devices from small-screen device such as smart phone and tablet, to wide-screen devices such as desktop and laptop. This is also a potential point for the application since it could run across the operation systems and devices. The only restriction, so far, is the connection. The device is required to have stable internet in order to use application probably. The reason for that requirement is that the website needs to communicate with many another web servers, in this case is Twitter and Stripe. Therefore, this requirement is reasonable.

Functions

Currently, at this initiation stage, the basic functions have been finished; however, there are still more functions which need to be developed. The website is expected to communicate with many different social networks such as Facebook, Google, Youtube etc to promote products and services more efficiently. Different from many another social media marketing applications, instead of focusing on specific process, this website plays a role as ERP (Enterprise resource planning) systems. It means that this website will take care many business activities, including marketing, sales, shipping and payment. Let's take a look on some existing application in market in order to compare and evaluate this website :

Ebay: a shopping website. Nowadays, almost everyone has used Ebay services in order to purchase various products from clothes to electronic facilities at least once time. This would emphasize the importance of the e-commerce. Similar to my website, Ebay is cooperating with a third party to foster better environment for e-payment and it is PayPal. According to EBay Inc. Chief Executive John Donahoe statement, "EBay and PayPal we believe are stronger together than they would be apart. EBay helps PayPal grow faster by creating a flywheel effect because PayPal is the primary and, in some cases, exclusive way to pay on eBay [Wall Street Journal, 2014].", my website is catching this trend to build up the better service by the support from Stripe.

However, certainly my website is not only a shopping website but also a social marketing tools. This is the most feature disparity from another marketing and e-commercial tools.

Sprout Social: this is social media publishing tool. There are two main similar functions with my website [Sproutsocial, 2016]. They are:

- *Automated Schedule*: user are able to specify precise publishing time of their campaigns in advance.
- *Multiple Social Networks*: the application posts messages to many servers such as Facebook, Twitter, Google+, LinkedIn.

Because this application emphasizes social marketing; thus, it offers quite many more functions for team work, managing posts and exploiting social network clients. For example, Sprout Social supports users to detect potential clients and track your post performance. These functions could be developed later on in my website; however, at this stage, my idea is expected to cover as many business activities as possible.

Hootsuite is the leader in the Social Media Marketing Management space. Besides of two functions above (Automated Schedule and Multiple Social Networks), Hootsuite offers user a central account management. It displays all user's Social Media accounts in multiple streams and they are able to reply directly at this page. Moreover, this application could manage your notifications and what kinds of posts you would love to see. For example, the main aim of using this application is to promote your own products and services, not to follow what your friends are sharing at the moment or another promoted posts. Therefore, there is no need to display those kinds of posts at management page [Hootsuite, 2016;Youtube, 2015].

Contrary to my website, this application focus on big organizations, agencies and governments which have the huge amount of social network accounts. This application supports users to manage their accounts more efficiently.
Overall, the most outstanding difference between my application and the others is that my website supports the manufacturers on many business activities instead of only one specific aspect. This could benefit company because:

- *Central data*: even though the application manages marketing and sale, the data for every campaigns are stored in a central database. Therefore, users are able to query data to make report easily. For example, how many products have been purchased since they are published?
- *Completed process*: the application aims to support small company to complete their lacking skills. Having this support, manufacturers now could specialize their services instead of spending unessential extra expense.

7. Conclusion

7.1. Objectives

The objective of this thesis is to elicit the communication between the Integrated Website and another web servers (in this thesis, they are Twitter and Stripe). The message, which is sent back and forth between client side and server side, may contain sensitive information of the owner. Therefore, each of API providers needs to have an algorithm in order to secure user's credentials. In addition, this thesis focuses on two API providers (Twitter and Stripe) from two different fields (Social Networks and E-payment). If Twitter applies 3-Legged OAuth 2.0 and a keyed-hash message authentication code (HMAC) to hide the meaning of the message, then Stripe follows public-key infrastructure framework to protect cardholder details.
Furthermore, a website was introduced as a result of the thesis. The main aims of the application are to:

- Demonstrate the researched theories intuitively. It will help the audience to have clearer vision about what this thesis is researching.
- Exploit the power and the speed of the multiple API providers to support users. In this case, this thesis took Twitter and Stripe as examples. The ideally result is to complete the website by selecting and integrating the best functions from various Web Servers. For instance, automatically posting tweets from Twitter, e-payment from Stripe, analysis from Google Analytics and so on.

In case, the users feel that the website is not flexible enough to satisfy them, a C# library is provided to customers for using on their purpose. This library offers many single actions without any scenarios. This would assist to develop their own application on their wish. However, it is required reasonable knowledge of technical skills to get full benefit from the library.
The next section will discuss the limitations of this thesis and the website as well. According those limitations, potential improvements or developments are proposed in the following section.

7.2. Progress

At the beginning, I was inspired from my internship in Kwork Finland. I was responsible to program various methods, which could make API calls to single server. Finally, I came up with the idea about the Integrated Website. After the quick research, I found that this topic is very interesting, potential to develop and useful for start-up companies. As a result, I spend more effort in order to complete the theoretical part such as determining suitable web servers, analyzing their services and forming my thesis structure. Finally, Twitter and Stripe were selected because of their simple and clear processes.

In order to dig intensively in Twitter's and Stripe's services, all essential materials such as official guideline on website, various relevant discussions on forums for developer and feedbacks from clients of Twitter and Stripe, were processed carefully.

After this step, I could not only fully aware of their workflow, but also detect the pros and cos in their services. At this stage, basically I had two main things, which could be processed concurrently. The first part is to finish my thesis as what I have designed in previous stages. The other part is to develop my own application following the official materials strictly.
The website's functionalities were designed based on the knowledge after analyzing Stripe and Twitter services. I have put myself into client shoes to complete and improve the main workflow of the website and judge the application. Basically, the website needs to satisfy those main requirements:

- The website is able to post the new tweet as defined and allow users to access and buy the product immediately after publishing without signing in.
- Clients could make transactions instantly via Stripe services without any problems in techniques.
- Unauthorized account must not activate the campaigns or access database.

Moreover, Testing was proceeded to secure that those requirements have been carefully implemented. Thus, Testing plays a big role in the project. So far, Data testing, Integration testing and Unit testing were completed in order to maintain the high product quality. Each of testing methods has different meaning for my application:

- *Data testing*: not only double checks the rigor of the database, but also verify the flow of code which connects to database. This kind of testing assists to judge the structure of database. Missing properties in database and bug in coding were revealed after this process.
- *Integration testing*: helps to find problems of environment conflict. For this testing, I have done it on different environments such as various browsers and Twitter accounts. The main point of this testing is to guarantee unauthorized access could not retrieve data from database.
- *Unit testing*: validates every piece of code. After finishing coding, a functionality of a single method was qualified again by this testing. The final destination of Unit testing is to ensure the method has been coded as design.

7.3. Problems of services

Stripe supports their clients to make a payment easily by various kinds of card such as visa card, master card, American express and many more. Because of processing the sensitive information such as card number, card security number, cardholder name and expiration date, Stripe is facing many challenges on how to secure those credentials as highest as possible. The Stripe's solution went to RESTful structure and Public Key Infrastructure (PKI). Stripe encrypts the cardholder details by using a pair of keys (a public and a private key). An encryption result is a valid token. Next, users are able to make a transaction by that valid token instead of passing cardholder details directly. This system seems to operate smoothly; however, Stripe made some small mistakes, which could influence negatively on their services.

The first Stripe's vulnerability is to include the private key in every request. According to the PKI guidelines, the private key should be kept in secret. In contrast with the public key, the private key must be controlled by the key owner only. Nevertheless, Stripe requires clients to attach their transparent private key in the request. This would lead to leaking out private key unexpectedly. Hackers could listen to the request, which was sent from client side. Because the private key has not gone through any encryption process, hackers easily obtain this key by analyzing the request. As the result, hackers are free to make transactions on behalf of the victims. Then, Stripe will struggle with another problem. It is determining hackers. It is hard to trace back to hackers since all evidences are pointed to a stolen account, a victim.

Moreover, including the customer's public key in a Javascript function. As mentioned above, users communicate with Stripe server by a valid token and this token was generated by this Javascript function. Although the public key could be shared widely, Stripe should not reveal the clear value in the code as they are doing at the moment. Nowadays, the vast majority of web browsers allow users to view the code behind. It means that everyone could obtain the customer's public key in a blink of the eyes. This would lead to a bigger risk for Stripe which could ruin the whole system. Let us assume that hackers are able to guess the algorithm to calculate the token. It is extremely difficult but feasible. Firstly, hackers register as new user. Next, they generate their own token by using their private/public keys and some encryption methodologies. Finally, those token are sent to server one by one in order to make transactions until one request went through the authentication process. After obtaining the encryption algorithm, any Stripe's client could be the target of hackers. They attack the victim by determining payment page, viewing the code behind and collecting the victim's public key. Lastly, they can detect victim's private key by applying encryption algorithm or simple listen to request from victims. Having all essentials arguments (public, private keys and valid token), hackers are able to make authorized requests on behalf of victims.

7.4. Solution

So far, the answer for two main problems of Stripe is to encrypt the private key before sending back and forth to server or even remove the private key from the request. According to another server's guideline such as Facebook, Twitter and Google, the server is able to detect the private key by the public key. Because Stripe is the server, which generated those keys. Therefore, Stripe is able to detect the private key based on the relationship between those two keys. As the result, users could replace their private key by the transparent public key, which claims to be distributed commonly.

Furthermore, In my opinion, Stripe should improve the code quality of Javascript function in order to avoid the second problem (leaking out the public key in Javascript file). Even better, Stripe should provide an API call for obtaining token. This is the popular method which is applying by many big API services. In this way, the public keys and cardholder details are encrypted and sent to Stripe. This not only secures the sensitive information but also avoids an unauthorized access to view and retrieve confidential credentials.

7.5. Future Work

At this moment, there are two potential topics which could be developed more in the future. One possibility is to dig more deeply into theory part. The other possibility is to develop the application by extending its functionalities.

Theoretical topic:

This thesis focuses on specific API Providers (Twitter and Stripe); therefore, it is possible to analyze another API services such as Facebook with Graph API, Google with various API and one of them is Adwords API for marketing. Because each of API Provider will apply one different method to secure their services.

Besides that, Stripe services are worth to spend more effort on. According to what was mentioned in Problem section, more solutions should be proposed since this is a potential target of hacker attacks. Any small flaw could be exploited and cause unpredictable problems in the future.

The last but not least, updating the latest security version of Twitter and Stripe is a noticeable consideration at this moment. Because this research narrowed down the scope by selecting specific web servers for analyzing, and the current security of selected API provider could be updated frequently. Security is one of the most important features in software development, as soon as the flaw has been discovered; the new version is released promptly. For example, Adwords API from Google has released 20 versions since 2009 [Google Developer, 2015]. It means that Google continually introduces new version in every three months on average. Therefore, Twitter and Stripe could also update their security systems in order to improve their services. However, those web servers still maintain their current system as long as their clients are satisfied and the providers believe that there is no flaw in their system.

Practical topic:

Currently the website is shaping day by day. There are quite many incomplete functions left such as:

* **Multiple Social Account**. The application needs to integrate more services from different social networks, for instance Facebook, Google, LinkedIn and Instagram etc. This would require more knowledge at their API because each API Provider follows different framework in order to secure their services.
* **A central account**. Currently, the application is using Twitter account instead of its own account. However, in case multiple social accounts, the website should detect users regardless social accounts. The ideal solution is to have a central for multiple accounts. This local account contains all information about various social accounts. Once users log-in by any social accounts, users are detected by this local account. This would lead to change slightly on database structure.
* **Interface improvement**. Because of lacking of fund, the interface was developed by my own. Therefore, it is neither friendly to users nor as fancy as expectations. One possibility is to purchase a professional design; however, it will cost some extra fees.

- **Shipping API**. The website aims to support small companies as much as possible and shipping could be taken into account. One of the most popular shipping API is EasyPost. Obviously, manufacturers needs to pack the product and deliver it to clients by themselves, thus, this API provides various services from easy purchasing shipping types to package tracking in real-time. This could satisfy clients better since they are able to check status of their package and estimate more correctly when their package will arrive.
- **Reporting Function**. This is a basic function which should have in every e-commerce website; however, this function is irrelevant to the topic of this thesis. Therefore, it was postponed to further version. This would allow manufacturers to query how many products have been sold so far, how many clients have accessed the advertisement from Facebook, Twitter, Google and so on.
- **Some small functions**. Feedback from customers is always worth to get as many as possible; so, this function needs to be implemented. Moreover, some useful functions such as email to confirm to clients after purchasing and inform news, smart chat between clients and providers and so on.

The flexibility of the Integrated website is that it could be extended its services by adding more API functions easily. Although the website still needs to be developed more, it is ready to use from now on. The new version could be released and applied as soon as possible. Hopefully, I could have opportunity to complete the website day by day.

8. References

[Allott et al., 2008] N. Allott, D. Rogers and G. Preston, "W3C Security Workshop Position Paper A web based security model fit for purpose", Aivailable from : *http://www.w3.org/2008/security-ws/papers/OMTP_Security_Position_Paper.pdf* [Accessed 12th December 2015].

[CGI Group , 2004] CGI Group , 2004 , Public Key Encryption and Digital Signature: How do they work?

[Cryptome, 2013] Cryptome, Fundamental Security Concepts, available from : *https://cryptome.org/2013/09/infosecurity-cert.pdf* , [Accessed 29th December 2015].

[Envoisolutions, 2007] Envoisolutions , "Scalable, Reliable, and Secure RESTful services".

[Fielding, 2000] R.T. Fielding, "Chapter 5: Representational State Transfer (REST), Architectural Styles and the Design of Network-based Software Architectures.", 2000, Available at : *http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm*

[Google Developer, 2015] Google Developer, Release Notes for Previous Versions, available from : *https://developers.google.com/adwords/api/docs/reference/archived-release-notes* [Accessed 18th December 2015].

[Hall , 2005] Prentice Hall, "Cryptography and Network Security : Principles and Practic, Second Edition", Chapter 14, Available from : *https://msdn.microsoft.com/en-us/library/cc767139.aspx* [Accessed 12th December 2015].

[Hardt, 2012] D. Hardt Ed, ISSN: 2070-1721, October 2012, The OAuth 2.0 Authorization Framework

[Hayat etal., 2005] Z. Hayat, J. Reeve and C. Boutle, "Domain Based Security: Improving Practices", Southampton University, sponsored by BAe Systems, p.1.

[Hevner et al, 2004] A. Hevner, S. March, J. Park, and S. Ram, "Design Science in Information Systems Research", MIS Quarterly, 28(1), 2004, pp. 75-105.

[Hevner and Chatterjee, 2010] A. Hevner and S. Chatterjee, Design Research in Information Systems, Integrated Series in Information Systems 22, DOI 10.1007/978-1-4419-5653-8_2.

[Hootsuite, 2016] Hootsuite, available from : *https://hootsuite.com/* [Accessed 4th January 2016].

[Hughes Systique Coporation, 2006] Hughes Systique Coporation, 2006 , "Securing RESTful Web Services Using Spring and OAuth 2.0", Available at: *www.hsc.com/portals/0/uploads/articles/wp_securing_restful_webservices_oauth26354 06646412464000.pdf* [Accessed 4th January 2016].

[IBM, 2015] IBM, “RESTful Web services: The basics”, 09 February 2015 (First published 06 November 2008) , available at : *https://www.ibm.com/developerworks/library/ws-restful/*

[Krasner and Pope, 1998] Glenn E. Krasner and Stephen T. Pope, “A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System”, p.2, available at : *http://www.create.ucsb.edu/~stp/PostScript/mvc.pdf* [Accessed 4th January 2016].

[Langford, 2003] J. Langford , “Implement Least Privilege at your Enterprise”, SANS Insitute, 5th July 2003.

[Loukas and Oke, 2009] G. Loukas and G. Oke , Protection against Denial of Service Attacks: A Survey, The Computer Journal Vol. 00 No. 0, 2009.

[M. Elkstein, 2008] M. Elkstein, “Learn REST: A Tutorial”, [Online], available at : *http://rest.elkstein.org/2008/02/how-simple-is-rest.html* [Accessed 4th January 2016].

[Microsoft, 2003] Microsoft Technet, “How Certificates Work” [online], Available from : *https://technet.microsoft.com/en-us/library/cc776447%28v=ws.10%29.aspx* [Accessed 12th December 2015].

[Microsoft, 2014] Microsoft, “What Is Windows Communication Foundation”, [Online], available at : *https://msdn.microsoft.com/en-us/library/ms731082%28v=vs.110%29.aspx* [Accessed 4th January 2016].

[Pautasso, 2008] C. Pautasso, REST vs. SOAP: Making the Right Architectural Decision, 7.10.2008.

[PCI, 2006] PCI Security Standards Council, " Maintaining Payment Security", [Online], Available : *https://www.pcisecuritystandards.org/pci_security/maintaining_payment_security* [Accessed 4th January 2016].

[PCI DSS Compliance Requirement 3, 2006] PCI DSS Compliance, “How to Comply to Requirement 3 of PCI”, [Online], available at : *http://pcidsscompliance.net/pci-dss-requirements/how-to-comply-to-requirement-3-of-pci-dss/* [Accessed 4th January 2016].

[PCI DSS Compliance Requirement 4, 2006] PCI DSS Compliance, “How to Comply to Requirement 4 of PCI”, [Online], available at : *http://pcidsscompliance.net/pci-dss-requirements/how-to-comply-to-requirement-4-of-pci-dss/* [Accessed 4th January 2016].

[ProgrammableWeb, 2012] ProgrammableWeb, “39 Statistics APIs: Compete, Clicky and Cownt Us”, Dec. 11 2012, 07:00AM EST, available at : *http://www.programmableweb.com/news/39-statistics-apis-compete-clicky-and-cownt-us/2012/12/11*

[PWC, 2014] PWC and The CERT® Division of the Software Engineerin Institute at Carnegie Mellon University ,”US cybercrime: Rising risks, reduced readiness Key findings from the 2014 US State of Cybercrime Survey” [Online]. Available from :

http://www.pwc.com/us/en/increasing-it-effectiveness/publications/2014-us-state-of-cybercrime.html [Accessed 12th December 2015]

[Richardson, 2010/2011] R.Richardson, CSI Director, "2010 / 2011 CSI Computer Crime and Security Survey" [Online]. [Accessed 12th December 2015]. Available from : *http://itlaw.wikia.com/wiki/CSI_Computer_Crime_and_Security_Survey_2010/2011*

[Simon, 1996] Herbert Alexander Simon, ""The Sciences of the Artificial", MIT Press, ISBN 0-262-69191-4.

[Slideshare, 2013] Slideshare, "Chart of Web API Growth From 2005 Through 2013", available at *http://www.slideshare.net/programmableweb/web-api-growthsince2005*

[Sproutsocial, 2016] Sproutsocial, available from: *http://sproutsocial.com/features/social-media-publishing* [Accessed 4th January 2016].

[Statista, 2015] Statista, "Number of monthly active Twitter users worldwide from 1st quarter 2010 to 3rd quarter 2015 (in millions)", available at *http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users* [Accessed 4th January 2016].

[Stripe, 2015] Stripe, "Stripe Integrating Checkout",available at : *https://stripe.com/docs/tutorials/checkout* [Accessed 4th January 2016].

[Taylor and Francis Group, 2003] Taylor and Francis Group, "Network Perimeter Security", ISBN : 978-0-203-50804-6.

[The Internet Society, 1999] The Internet Society (1999), "HTTP Authentication: Basic and Digest Access Authentication" [Online], available at : *http://www.rfc-base.org/txt/rfc-2617.txt* [Accessed 4th January 2016].

[Twitter Documentation, 2015] Twitter Documentation, "How long does an access token last?" , OAuth FAQ, available at *https://dev.twitter.com/oauth/overview/faq* [Accessed 4th January 2016].

[Twitter Developer, 2015] Twitter Developer, "Error Codes & Responses", [Online], available at : *https://dev.twitter.com/overview/api/response-codes* [Accessed 4th January 2016].

[Wall Street Journal, 2014] EBay CEO Reiterates Company Stronger With PayPal, available from : *http://www.wsj.com/articles/SB10001424052702303630904579419233890681114* [Accessed 16th December 2015].

[Wagner et al., 2007] Raymond Wagner, J. Ryan Stinnett, Marco Duarte, Richard Baraniuk, David B. Johnson and T. S. Eugene Ng Rice, University Technical Report TREE0705, "A Network Application Programming Interface for Data Processing in Sensor Networks".

[Wagnon, 2013] J. Wagnon , June 20 2013, “SSL Profiles Part 8: Client Authentication”, available at *https://devcentral.f5.com/articles/ssl-profiles-part-8-client-authentication* [Accessed 4th January 2016].

[Weber et al, 2012] S. Weber, R. Beck, R.W. Gregory , “Combining Design Science and Design Research Perspectives - Findings of Three Prototyping Projects”, 2012 45th Hawaii International Conference on System Sciences.

[Youtube, 2015] Youtube, Overview of Hootsuite, available from *https://www.youtube.com/watch?v=8tgPuE5Latw*