CHAPTER 1

Welcome to ASP.NET Web API

“Begin at the beginning,” the King said gravely, “and go on till you come to the end: then stop.”

—Lewis Carroll, Alice in Wonderland

If you have chosen to read this book, which is on ASP.NET Web API security, it is highly likely that you are familiar with ASP.NET Web API. In case you are not or simply would like me to begin at the beginning, this introductory chapter along with the next chapter will help you gain a quick understanding of the basics of ASP.NET Web API and help you appreciate the need for the emphasis on security for ASP.NET Web API applications.

ASP.NET Web API Security: If we break this down, we get multiple terms—Web API, ASP.NET Web API, and Security. We start by understanding what a web API is in general before moving on to a primer on RESTful Web API, followed by a review of how the Microsoft ASP.NET Web API framework can help you build web APIs. We complete the chapter with a primer on security that looks at all aspects of security, above and beyond the login screen accepting a username and password, which for many is synonymous with the word security.

What Is a Web API, Anyway?

It all started with the launch of Sputnik in 1957, by the Union of Soviet Socialist Republics (USSR). The United States, under the leadership of then President Eisenhower, started the Advanced Research Projects Agency (ARPA) to advance the United States in the technology race, in the light of the Sputnik launch. One of the ARPA-funded projects was ARPANET, the world’s first operational packet switching network. ARPANET led to the development of protocols that allowed networks to be joined together into a network of networks that evolved into the ubiquitous Internet of today.

The terms Internet and World Wide Web or simply Web, are generally used interchangeably, but they are separate although related things. The Internet is the infrastructure on which the World Wide Web has been built. The Internet connects islands of smaller and bigger networks into one huge network.

The World Wide Web builds on this network by providing a model to share data or information with the computer users who are all part of the Internet. Servers or web servers serve data in the form of documents or web pages to the clients, called web browsers, which display the documents in a format readable by human beings. Typically, a web page is created in a language called Hyper Text Markup Language (HTML) and is served to a browser by the web server as a result of both parties following a protocol, Hyper Text Transfer Protocol (HTTP). The Web is just one of the ways information can be shared over the Internet. Just like HTTP, there is Simple Mail Transfer Protocol (SMTP) for e-mail, File Transfer Protocol (FTP) for transfer of information in the form of files, and so on.

Initially, web pages were just static pages existing in the file system of some computer with data that hardly changed. As the World Wide Web started to grow and the user base started to expand, there was a need for web pages to be generated on the fly. Web servers started delegating this responsibility to engines such as the Common Gateway Interface (CGI) to generate web pages on the fly. The dynamic web pages and the introduction of the client-side JavaScript scripting language led to a new generation of software applications called web applications. The end user of a web application is a human being with an objective of performing a task.
Because the end user of a web application is a human being, there is a user interface associated with a web application. The browser is what provides this interactive interface for a user. In addition, there is a need for nonhuman entities such as a machine running some software to communicate and exchange data over the World Wide Web. Enter the web service. Although not mandated, a web service uses HTTP to exchange data. Unlike a web application, which is mainly about HTML over HTTP, for a web service it is mainly Extensible Markup Language (XML) over HTTP. A client sends a request in XML, and the server responds with an XML response. This XML can be Plain Old XML (POX), which is typically a nonstandard XML only the client and server will be able to make sense out of, or it can be standard Simple Object Access Protocol (SOAP).

To appreciate the value SOAP brings to the table, let us pretend we got some XML response representing an employer in an organization, as shown in Listing 1-1.

Listing 1-1. Response XML

```xml
<employee>
    <firstname>John</firstname>
    <lastname>Human</lastname>
    <salary>2000</salary>
    <doj>06/01/1998</doj>
    <lastlogin>10/20/2012 09:30:00</lastlogin>
</employee>
```

To do anything useful with this in our application, this XML might need to be loaded into some data structure, say an object as defined by a class in the case of an object-oriented programming (OOP) language. If I’m programming, how will I define the data type of the field to store salary? Will it be an integer or a fractional number? What if my request to get the employee fails because there is no such employee or there is some other problem? How will I know where to look in the XML if the request has failed? SOAP helps us with questions like these by providing a basic messaging framework on which web services can be built. SOAP has Microsoft roots, although it is currently maintained by the World Wide Web Consortium (W3C).

Microsoft technologies such as the ASMX-based web service, which is currently a legacy technology, and its successor Windows Communication Foundation (WCF) all have great affinity toward SOAP. An ASMX-based web service allows the exchange of SOAP messages over HTTP and that’s pretty much it. WCF builds on this and tries to abstract away the infrastructure from the programming. If I have an Employee service that returns the details of an employee, I can host the service to be consumed over HTTP, over Transmission Control Protocol (TCP), through Microsoft Message Queuing (MSMQ), or any combinations thereof. By having the same contract with the client, I can have multiple binding for multiple ways my service can be reached. In both cases, though, the payload will be SOAP, by default. An important aspect of SOAP-based web services is the availability of a Web Service Definition Language (WSDL) file, which allows tooling to be built that helps in consumption of services. For example, Microsoft Visual Studio can generate proxy classes reading WSDL definitions, and the client trying to consume the services (i.e., the programmer writing the client code) can directly work with the generated classes, with the whole existence of the web service hidden from the programmer.

A web API is a service. Technically, there is no difference. What is different is the manner in which a web API is intended to be used. Let’s say I have a web application where a user can post his thoughts in the form of a short message. A user can log in to my application from a browser, add new posts or update the ones she posted in the recent past, or even delete the old ones. In other words, users can perform create, read, update, and delete (CRUD) operations on their posts using my web application. My application became so popular that there are folks who want to integrate this CRUD functionality into their mobile apps so that users can perform CRUD operations from their mobile devices without logging on to my web application while they are away from their normal computers.

I can now create a web service to support the CRUD operations. Technically it is a web service, but it is an application programming interface (API) to interact with my web application, except that it is over the Web. Traditionally, APIs are a bunch of classes with properties and methods that are part of a reusable component to interact with another application. This scenario is exactly that, except that my API is not available in the form of a software component, but over the Web instead. It is a web API!