O'REILLY®

Early Release

RAW & UNEDITED

Security for Web Developers

USING JAVASCRIPT, HTML, AND CSS

John Paul Mueller

I

Developing a Security Plan

1

Defining the Application Environment

Data is the most important resource that any business owns. It's literally possible to replace any part of a business except the data. When the data is modified, corrupted, stolen, or deleted, a business can suffer serious loss. In fact, a business that has enough go wrong with its data can simply cease to exist. The focus of security, therefore, is not hackers, applications, networks, or anything else someone might have told you—it's data. Therefore, this book is about data security, which encompasses a broad range of other topics, but it's important to get right to the point of what you're really looking to protect when you read about these other topics.

Unfortunately, data isn't much use sitting alone in the dark. No matter how fancy your server is, no matter how capable the database that holds the data, the data isn't worth much until you do something with it. The need to manage data brings applications into the picture and the use of applications to manage data is why this introductory chapter talks about the application environment.

However, before you go any further, it's important to decide precisely how applications and data interact because the rest of the chapter isn't very helpful without this inside. An application performs just four operations on data, no matter how incredibly complex the application might become. You can define these operations by the CRUD acronym:

- Create
- Read
- Update
- Delete

The sections that follow discuss data, applications, and CRUD as they relate to the web environment. You discover how security affects all three aspects of web development, keeping in mind that even though data is the focus, the application performs the required CRUD tasks. Keeping your data safe means understanding the application environment and therefore the threats to the data the application manages.

Specifying Web Application Threats

You can find lists of web application threats all over the Internet. Some of the lists are quite complete and don't necessarily have a bias, some address what the author feels are the most important threats, some lists tell you about the most commonly occurring threats, and you can find all sorts of other lists out there. The problem with all these lists is that the author doesn't know your application. A SQL injection attack is only useful if your application uses SQL in some way—perhaps it doesn't.

Obviously, you need to get ideas on what to check from somewhere and these lists do make a good starting place. However, you need to consider the list content in light of your application. In addition, don't rely on just one list—use multiple lists so that you obtain better coverage of the threats that could possibly threaten your application. With this need in mind, here is a list of the most common threats you see with web applications today:

- Buffer Overflow: An attacker manages to send enough data in an input buffer to overflow an application or output buffer. As a result, memory outside the buffer becomes corrupted. Some forms of buffer overflow allow the attacker to perform seemingly impossible tasks because the affected memory contains executable code. The best way to overcome this problem is to perform range and size checks on any data, input or output, that your application handles.
- Code Injection: An entity adds code to the data stream flowing between a server and a client (such as a browser). The target often views the added code as part of the original page, but it could contain anything. Of course, the target may not even see the injected code. It might be lurking in the background ready to cause all sorts of problems for your application. A good way to overcome this attack is to ensure you use encrypted data streams, the HTTPS protocol, and code verification (when possible). Providing a client feedback mechanism is also a good idea.

  > Code injection occurs more often than you might think. In some cases, the code injection isn't even part of an attack, but it might as well be. A recent article (see http://www.infoworld.com/article/2925839/net-neutrality/code-injection-new-low-isps.html) discusses how Internet Service Providers (ISPs) are injecting JavaScript code into the data stream in order to overlay ads on top of a page. In order to determine what sort of ad to provide, the ISP also monitors the traffic.

- Cross-site Scripting (XSS): An attacker injects JavaScript or other executable code into the output stream of your application. The recipient sees your application as the source of the infection, even when it isn't. In most cases, you don't want to allow users to send data directly to each other through your application without strict verification. A moderated format for applications such as blogs is a must to ensure your application doesn't end up serving viruses or worse along with seemingly benign data.

  > Few experts remind you to check your output data. However, you don't actually know that your own application is trustworthy. A hacker could modify it to allow tainted output data. Verification checks should include output data as well as input data.

- File Uploads: Every file upload, even those that might seem otherwise innocuous, is suspect. If possible, disallow file uploads to your server. Of course, it isn't always possible to provide this level of security, so you need to allow just certain types of file and then scan the file for problems. Authenticating the file as much as is possible is always a good idea. For example, some files contain a signature at the beginning that you can use to ensure the file is legitimate. Don't rely on file extension exclusion alone—hackers often make one file look like another type in order to bypass server security.
- Hard Coded Authentication: Developers often place authentication information in application initialization files for testing purposes. It's essential to remove these hard coded authentication entries and rely on a centralized data store for security information instead. Keeping the data store in a secure location, off the server used for web applications, is essential to ensuring that hackers can't simply view the credentials used to access the application in certain ways. If you do need initialization files for the application, make sure these files reside outside the webroot directory to ensure that hackers can't discover them accidentally.
- Hidden or Restricted File/Directory Discovery: When your application allows input of special characters such as the forward slash (/) or backslash (\), it's possible for a hacker to discover hidden or restricted files and directories. These locations can contain all sorts of information that a hacker can find useful in attacking your system. Disallowing use of special characters whenever possible is a great idea. In addition, store critical files outside the webroot directory in locations that the operating system can control directly.
- Missing or Incorrect Authentication: It's important to know whom you're dealing with, especially when working with sensitive data. Many web applications rely on common accounts for some tasks, which means it's impossible to know who has accessed the account. Avoid using guest accounts for any purpose and assign each user a specific account to use.
- Missing or Incorrect Authorization: Even if you know the person you're dealing with, it's important to provide only the level of authorization needed to perform a given task. In addition, the authorization should reflect the user's method of access. A desktop system accessing the application from the local network is likely more secure than a smartphone accessing the application from the local coffee shop. Relying on security promotion to assist in sensitive tasks lets you maintain minimal rights the rest of the time. Anything you can do to reduce what the user is authorized to do helps maintain a secure environment.
- Missing or Incorrect Encryption: Use encryption to transmit data of any sort between two endpoints to help keep hackers from listening in on your communication. It's important to keep track of the latest encryption techniques and rely on the best encryption supported by the user's environment. For example, Triple Data Encryption Standard (3DES) isn't secure any longer, yet some organizations continue to use it. The current Advanced Encryption Standard (AES) remains mostly secure, but you want to use the largest key possible to help make it harder to crack.
- Operating System Command Injection: An attacker modifies an operating system command your application uses to perform specific tasks. Your web-based application probably shouldn't use operating system calls in the first place. However, if you absolutely must make operating system calls, make sure the application runs in a sandbox.

> Some experts will emphasize validating input data for some uses and leave the requirement off for other uses. Always validate any data you receive from anywhere. You have no way of knowing what vehicle a hacker will use to obtain access to your system or cause damage in other ways. Input data is always suspect, even when the data comes from your own server. Being paranoid is a good thing when you're performing security-related tasks.

- Parameter Manipulation: Hackers can experiment with parameters passed as part of the request header or URL. For example, when working with Google, you can change the URL and the results of your search. Make sure you encrypt any parameters you pass between the browser and the server. In addition, use secure web page protocols, such as HTTPS, when passing parameters.
- Remote Code Inclusion: Most web applications today rely on included libraries, frameworks, and APIs. In many cases, the include statement contains a relative path or uses a variable containing a hard coded path to make it easier to change the location of the remote code later. When a hacker is able to gain access to the path information and change it, it's possible to point the remote code inclusion to any code the hacker wants, giving the hacker full access to the application. The best way to avoid this particular problem is to use hard coded full paths whenever possible, even though this action makes it harder to maintain the code.

  > Many experts will recommend that you use vetted libraries and frameworks to perform dangerous tasks. However, these add-ons are simply more code. Hackers find methods for corrupting and circumventing library and framework code on a regular basis. You still have a need to ensure your application and any code it relies upon interacts with outside elements safely, which means performing extensive testing. Using libraries and frameworks does reduce your support costs and ensures that you get timely fixes for bugs, but the bugs still exist and you still need to be on guard. There is no security silver bullet. Chapter 6 contains more information about working with libraries and frameworks.

- Session Hijacking: Every time someone logs into your web server, the server gives that user a unique session. A session hijacker jumps into the session and intercepts data transferred between the user and the server. The three common places to look for information used to hijack a session are: cookies, URL rewriting, and hidden fields. Hackers look for session information in these places. By keeping the session information encrypted, you can reduce the risk of someone intercepting it. For example, make sure you rely on the HTTPS protocol for logins. You also want to avoid doing things like making your session IDs predictable.
- SQL Injection: An attacker modifies a query that your application creates as the result of user or other input. In many cases, the application requests query input data, but it receives SQL elements instead. Other forms of SQL injection attack involve the use of escape or other unexpected characters or character sequences. A good way to avoid SQL injection attacks is to avoid dynamically generated queries.

This may look like a lot of different threats, but if you search long enough online, you could easily triple the size of this list and not even begin to scratch the surface of the

ways in which a hacker can make your life interesting. As this book progresses, you'll encounter a much larger number of threat types and start to discover ways to overcome them. Don't worry, in most cases the fixes end up being common sense and a single fix can resolve more than one problem. For example, look through the list again and you'll find that simply using HTTPS solves a number of these problems.

Considering the Privacy Aspect of Security

When delving into security, an organization tends to focus first on its own data security. After all, if the organization's data becomes lost, corrupted, modified, or otherwise unusable, the organization could go out of business. The next level of scrutiny usually resides with third parties, such as partners. Often, the security of user data comes last and many organizations don't think too much about customer data security at all. The problem is that many users and customers see the safety of their data as paramount. The whole issue of privacy comes down to the protection of user data such that no one misuses or exposes the information without the user's knowledge and consent. In short, when building an application, you must also consider the privacy of user data as a security issue and an important one at that.

A recent article points out that users and customers view the tech industry as poor trustees of their data (http://www.infoworld.com/article/2925292/internet-privacy/feds-vs-silicon-valley-who-do-you-trust-less.html). In fact, the tech industry has actually fallen behind the government—people trust the government to safeguard their information more often. Many tech companies publicly support enhanced security policies for other entities (such as the government) and privately build more ways to thwart any notion of privacy that the user or customer might have. This duality makes the situation even worse than it might otherwise be if the tech industry were open about the encroachment on user and customer data.

In order to create a truly secure application, you must be willing to secure every aspect of it, including user and customer data. This act requires that the application only obtain and manage the data necessary to perform its task and that it discard that data when no longer needed. Trust is something that your application can gain only when it adheres to the same set of rules for working with all data, no matter its source.

Understanding Software Security Assurance (SSA)

The purpose of software is to interact with data. However, software itself is a kind of data. In fact, data comes in many forms that you might not otherwise consider and the effect of data is wider ranging that you might normally think. With the Internet of Things (IoT), it's now possible for data to have both abstract and physical effects in ways that no one could imagine even a few years ago. A hacker gaining access to the right application can do things like damage the electrical grid or poison the water system. On a more personal level, the same hacker could potentially raise the temperature of your home to some terrifying level, turn off all the lights, spy on you through your webcam, or do any of a number of other things. The point of SSA is that software needs some type of regulation to ensure it doesn't cause the loss, inaccuracy, alteration, unavailability, or

misuse of the data and resources that it uses, controls, and protects. This requirement appears as part of SSA. The following sections discuss SSA in more detail.

> SSA isn't an actual standard at this time. It's a concept that many organizations quantify and put into writing based on that organization's needs. The same basic patterns appear in many of these documents and the term SSA refers to the practice of ensuring software remains secure. You can see how SSA affects many organizations, such as Oracle (http://www.oracle.com/us/support/assurance/overview/index.html) and Microsoft (https://msdn.microsoft.com/library/windows/desktop/84aed186-1d75-4366-8e61-8d258746bopq.aspx) by reviewing that organizations SSA documentation online. In fact, many large organizations now have some form of SSA in place.

Considering the OSSAP

One of the main sites you need to know about in order to make SSA a reality in web applications is the Open Web Application Security Project (OWASP) (https://www.owasp.org/index.php/OWASP_Software_Security_Assurance_Process) (see Figure 1-1). The site breaks down the process required to make the OWASP Security Software Assurance Process (OSSAP) part of the Software Development Lifecycle (SDLC). Yes, that's a whole bunch of alphabet soup, but you need to know about this group in order to create a process for your application that matches the work done by other organizations. In addition, the information on this site helps you develop a security process for your application that actually works, is part of the development process, and won't cost you a lot of time in creating your own process.

Figure 1-1. The OWASP site tells you about SSA for web applications.

> Even though OSSAP does provide a great framework for ensuring your application meets SSA requirements, there is no requirement that you interact with this group in any way. The group does license its approach to SSA. However, at this time, the group is just getting underway and you'll find a lot of TBDs on the site will the group plans to fill in as time passes. Of course, you need a plan for today, so OWASP and its OSSAP present a place for you to research solutions for now and possibly get additional help later.

The whole reason to apply SSA to your application as part of the SDLC is to ensure that the software is as reliable and error free as you can make it. When talking with some people, the implication is that SSA will fix every potential security problem that you might encounter, but this simply isn't the case. SSA will improve your software, but you can't find any pieces of software anywhere that are error free. Assuming that you did manage to create a piece of error free software, you still have user, environment, network, and all software of other security issues to consider. Consequently, SSA is simply one piece of a much larger security picture and implementing SSA will only fix so many security issues. The best thing to do is to continue seeing security as an ongoing process.

Defining SSA Requirements

The initial step in implementing SSA as part of your application is to define the SSA requirements. These requirements help you determine the current state of your software, the issues that require resolution, and the severity of those issues. After the issues are defined, you can determine the remediation process and any other requirements needed to ensure that the software remains secure. In fact, you can break SSA down into eight steps:

1. Evaluate the software and develop a plan to remediate it.
2. Define the risks that the security issues represent to the data and categorize these risks to remediate the worst risks first.
3. Perform a complete code review.
4. Implement the required changes.
5. Test the fixes you create and verify that they actually do work on the production system.
6. Define a defense for protecting application access and therefore the data that the application manages.
7. Measure the effectiveness of the changes you have made.
8. Educate management, users, and developers in the proper methods to ensure good application security.

Categorizing Data and Resources

This process involves identifying the various pieces of data that your application touches in some way, including its own code and configuration information. Once you identify every piece of data, you categorize it to identify the level of security required to protect that data. Data can have many levels of categorization and the way in which you categorize the data depends on your organization's needs and the orientation of the data. For example, some data may simply inconvenience the organization, while other data could potentially cause harm to humans. The definition of how data security breaches affects the security environment as a whole is essential.

After the data categorization process is complete, it's possible to begin using the information to perform a variety of tasks. For example, you can consider how to reduce vulnerabilities by:

- Creating coding standards
- Implementing mandatory developer training
- Hiring security leaders within development groups
- Using automated testing procedures that specifically locate security issues

All of these methods point to resources that the organization interacts with and relies upon to ensure the application manages data correctly. Categorizing resources means determining how much emphasis to place on a particular resource. For example, denying developers training will have a bigger impact than denying individual application users training because the developers work with the application as a whole. Of course, training is essential for everyone. In this case, categorizing resources of all sorts helps you

determine where and how to spend money in order to obtain the best Return on Investment (ROI), while still meeting application security goals.

Performing the Required Analysis

As part of SSA, you need to perform an analysis on your application. It's important to know precisely what sorts of weaknesses your code could contain. The operative word here is "could." Until you perform analysis in depth, you have no way of knowing the actual security problems in your code. Web applications are especially adept at hiding issues because, unlike desktop applications, the code can appear in numerous places and scripts tend to hide problems that compiled applications don't have because the code is interpreted at runtime, rather than compile time.

> It's important to understand that security isn't just about the code—it's also about the tools required to create the code and the skill of the developers employing those tools. When an organization chooses the wrong tools for the job, the risk of a security breach becomes much higher because the tools may not create code that performs precisely as expected. Likewise, when developers using the tool don't have the required skills, it's hardly surprising that the software has security holes that a more skilled developer would avoid.
>
> Some experts claim that there are companies that actually allow substandard work. In most cases, the excuse for allowing such work is that the application development process is behind schedule or that the organization lacks required tools or expertise. The fact that an organization may employ software designed to help address security issues (such as a firewall), doesn't alieve the developer of the responsibility to create secure code. Organizations need to maintain coding standards to ensure a good result.

Logic

Interacting with an application and the data it manages is a process. Even though users might perform tasks in a seemingly random fashion, specific tasks follow patterns that occur because the user must follow a procedure in order to obtain a good result. By documenting and understanding these procedures, you can analyze application logic from a practical perspective. Users rely on a particular procedure because of the way in which developers design the application. Changing the design will necessarily change the procedure.

The point of the analysis is to look for security holes in the procedure. For example, the application may allow the user to remain logged in, even if it doesn't detect activity for an extended period. The problem is that the user might not even be present—someone else could access the application using the users credentials and no one would be the wiser because everyone would think that the user is logged in using the same system as always.

However, data holes can take other forms. A part number might consist of various quantifiable elements. In order to obtain a good part number, the application could ask for the elements, rather than the part number as a whole, and build the part number from those elements. The idea is to make the procedure cleaner, clearer, and less error prone so that the database doesn't end up containing a lot of bad information.

Data

It may not seem like you can perform much analysis on data from a security perspective, but there really are a lot of issues to consider. In fact, data analysis is one of the areas where organizations fall down most because the emphasis is on how to manage and use the data, rather than on how to secure the data (it's reasonable to assume you need to address all three issues). When analyzing the data, you must consider these issues:

- Who can access the data
- What format is used to store the data
- When the data is accessible
- Where the data is stored
- Why each data item is made available as part of the application
- How the data is broken into components and the result of combining the data for application use

For example, some applications fail to practice data hiding, which is an essential feature of any good application. Data hiding means giving the user only the amount of information actually needed to perform any given task.

Applications also format some data incorrectly. For example, storing passwords as text will almost certainly cause problems should someone break in. A better route is to store the password hash. The hash isn't at all valuable to someone who has broken in because the application needs the password on which the hash is based.

Making all data accessible all the time is also a bad idea. Sensitive data should only appear on screen when someone is available to monitor its use and react immediately should the user do something unexpected.

Storing sensitive data in the cloud is a particularly bad idea. Yes, using cloud storage makes the data more readily available and faster to access as well, but it also makes the data vulnerable. Store sensitive data on local servers when you have direct access to all the security features used to keep the data safe.

Application developers also have a propensity for making too much information available. You use data hiding to keep manager-specific data hidden from other kinds of users. However, some data has no place in the application at all. If no one actually needs a piece of data to perform a task, then don't add the data to the application.

Many data items today are an aggregation of other data elements. It's possible for a hacker to learn a lot about your organization by detecting the form of aggregation used and taking the data item apart to discover the constituent parts. It's important to consider how the data is put together and to add safeguards that make it harder to discover the source of that data.

Interface

A big problem with software today is the inclusion of gratuitous features. An application is supposed to meet a specific set of goals, perform a specific set of tasks. Invariably, someone gets the idea that the software might be somehow better if it had certain features that have nothing to do with the core goals the software is supposed to meet. The term feature bloat has been around for a long time. You normally see it discussed in a monetary sense—as the source of application speed problems, the elevator of user

training costs, and the wrecker of development schedules. However, application interface issues, those that are often most affected by feature bloat, have a significant impact on security in the form of increased attack surface. Every time you increase the attack surface, you provide more opportunities for a hacker to obtain access to your organization. Getting rid of gratuitous features or moving them to an entirely different application, will reduce the attack surface—making your application a lot more secure. Of course, you'll save money too.

Another potential problem is the hint interface—one that actually gives the security features of the application away by providing a potential hacker with too much information or too many features. Even though the password used to help a user retrieve a lost password is necessary, some implementations actually make it possible for a hacker to retrieve the user's password and become that user. The hacker might even lock the real user out of the account by changing the password (although, this action would be counterproductive because an administrator could restore the user's access quite easily). A better system is to ensure that the user actually made the request before doing anything and then ensuring that the administrator sends the login information in a secure manner.

Constraint

A constraint is simply a method of ensuring that actions meet specific criteria before the action is allowed. For example, disallowing access to data elements unless the user has a right to access them is a kind of constraint. However, constraints have other forms that are more important. The most important constraint is determining how any given user can manage data. Most users only require read access to data, yet applications commonly provide read/write access, which opens a huge security hole.

Data has constraints to consider as well. When working with data, you must define precisely what makes the data unique and ensure the application doesn't break any rules regarding that uniqueness. With this in mind, you generally need to consider these kinds of constraints:

- Ensure the data is the right type
- Define the range of values the data can accept
- Specify the maximum and minimum data lengths
- List any unacceptable data values

Delving into Language-specific Issues

The application environment is defined by the languages use to create the application. Just as every language has functionality that makes it perform certain tasks well; every language also has potential problems that make it a security risk. Even low-level languages, despite their flexibility, have problems induced by complexity. Of course, web-based applications commonly rely on three particular languages: HTML, CSS, and JavaScript. The following sections describe some of the language specific issues related to these particular languages.

Defining the Key HTML Issues

HTML5 has become extremely popular because it supports an incredibly broad range of platforms. The same application can work well on a user's desktop, tablet, and

smartphone without any special coding on the part of the developer. Often, libraries, APIs, and microservices provide content in a form that matches the host system automatically, without any developer intervention. However, the flexibility that HTML5 provides can also be problematic. The following list describes some key security issues you experience when working with HTML5.

- Code Injection: HTML5 provides a large number of ways in which a hacker could inject malicious code, including sources you might not usually consider suspicious, such as a YouTube video or streamed music.
- User Tracking: Because your application uses code from multiple sources in most cases, you might find that a library, API, or microservice actually performs some type of user tracking that a hacker could use to learn more about your organization. Every piece of information you give a hacker makes the process of overcoming your security easier.
- Tainted Inputs: Unless you provide your own input checking, HTML5 lets any input the user wants to provide through. You may only need a numeric value, but the user could provide a script instead. Trying to check inputs thoroughly to ensure you really are getting what you requested is nearly impossible on the client side, so you need to ensure you have robust server-side checking as well.

Defining the Key CSS Issues

Applications rely heavily on CSS3 to create great looking presentations without hard coding the information for every device. Libraries of pre-existing CSS3 code makes it easy to create professional looking applications that a user can change to meet any need. For example, a user may need a different presentation for a particular device or require the presentation use a specific format to meet a special need. The following list describes some key security issues you experience when working with CSS3.

- Overwhelming the Design: A major reason that CSS3 code causes security issues is that the design is overwhelmed. The standards committee originally designed CSS to control the appearance of HTML elements, not to affect the presentation of an entire web page. As a result, the designers never thought to include security for certain issues because CSS wasn't supposed to work in those areas. The problem is that the cascade part of CSS doesn't allow CSS3 to know about anything other than its parent elements. As a result, a hacker can create a presentation that purports to do one thing, when it actually does another. Some libraries, such as jQuery, can actually help you overcome this issue.
- Uploaded CSS: In some cases, an application designer will allow a user to upload a CSS file to achieve a particular application appearance or make it work better with a specific platform. However, the uploaded CSS can also contain code that makes it easier for a hacker to overwhelm any security you have in place or to hide dirty dealings from view. For example, a hacker could include URLs in the CSS that redirect the application to unsecure servers.
- CSS Shaders: A special use of CSS can present some extreme problems by allowing access to the user agent data and cross-domain data. Later chapters in the book will discuss this issue in greater detail, but you can get a quick overview of the topic at http://www.w3.org/Graphics/fx/wiki/CSS_Shaders_Security. The big thing is that sometimes the act of rendering data on screen opens potential security holes you might not have considered initially.

Defining the Key JavaScript Issues

The combination of JavaScript with HTML5 has created the whole web application phenomenon. Without the combination of the two languages, it wouldn't be possible to create applications that run well anywhere on any device. Users couldn't even think about asking for that sort of application in the past because it just wasn't possible to provide it. Today, a user can perform work anywhere using a device that's appropriate for the location. However, JavaScript is a scripted language that can have some serious security holes. The following list describes some key security issues you experience when working with JavaScript.

- Cross-site Scripting (XSS): This issue appears earlier in the chapter because it's incredibly serious. Any time you run JavaScript outside a sandboxed environment, it becomes possible for a hacker to perform all sorts of nasty tricks on your application.
- Cross-site Request Forgery (CSRF): A script can use the user's credentials that are stored in a cookie to gain access to other sites. While on these sites, the hacker can perform all sorts of tasks that the application was never designed to perform. For example, a hacker can perform account tampering, data theft, fraud, any many other illegal activities, all in the user's name.
- Browser and Browser Plug-in Vulnerabilities: Many hackers rely on known browser and browser-plug in vulnerabilities to force an application to perform tasks that it wasn't designed to do. For example, a user's system could suddenly become a zombie transmitting virus code to other systems. The extent of what a hacker can do is limited by the vulnerabilities in question. In general, you want to ensure that you install any updates and that you remain aware of how vulnerabilities can affect your application's operation.

Considering Endpoint Defense Essentials

An endpoint is a destination for network traffic, such as a service or a browser. When packets reach the endpoint, the data they contain is unpacked and provided to the application for further processing. Endpoint security is essential because endpoints represent a major point of entry for networks. Unless the endpoint is secure, the network will receive bad data transmissions. In addition, broken endpoint security can cause harm to other nodes on the network. The following sections discuss three phases of endpoint security: prevention, detection, and remediation.

> It's important not to underestimate the effect of endpoint security on applications and network infrastructure. Some endpoint scenarios become quite complex and their consequences hard to detect or even understand. For example, a recent article discusses a router attack that depends on the attacker directing an unsuspecting user to a special site: http://www.infoworld.com/article/2926221/security/large-scale-attack-hijacks-routers-through-users-browsers.html. The attack focuses on the router that the user depends upon to make Domain Name System (DNS) requests. By obtaining full control over the router, the attacker can redirect the user to locations that the attacker controls.

Preventing Security Breaches

The first step in avoiding a trap is to admit the trap exists in the first place. The problem is that most companies today don't think that they'll experience a data breach—it always happens to the other company—the one with lax security. However, according to the Poneman Institute's *2014 Cost of Cyber Crime* report, the cost of cybercrime was $12.7 million in 2014, which is up from the $6.5 million in 2010 (http://info.hpenterprisesecurity.com/LP_CP_424710_Ponemon_ALL). Obviously, all those break-ins don't just happen at someone else's company—they could easily happen at yours, so it's beneficial to assume that some hacker, somewhere, has targeted your organization. In fact, if you start out with the notion that a hacker will not only break into your organization, but also make off with the goods, you can actually start to prepare for the real world scenario. Any application you build must be robust enough to:

- Withstand common attacks
- Report intrusions when your security fails to work as expected
- Avoid making assumptions about where breaches will occur
- Assume that, even with training, users will make mistakes causing a breach

> Don't assume that security breaches only happen on some platforms. A security breach can happen on any platform that runs anything other than custom software. The less prepared that the developers for a particular platform are, the more devastating the breach becomes. For example, many people would consider Point-of-Sale (POS) terminals safe from attack. However, hackers are currently attacking these devices vigorously in order to obtain credit card information access (see http://www.computerworld.com/article/2925583/security/attackers-use-email-spam-to-infect-pos-terminals.html). The interesting thing about this particular exploit is that it wouldn't work if employees weren't using the POS terminals incorrectly. This is an instance where training and strong policies could help keep the system safe. Of course, the applications should still be robust enough to thwart attacks.

As the book progresses, you find some useful techniques for making a breach less likely. The essentials of preventing a breach, once you admit a breach can (and probably will) occur, are to:

- Create applications that users understand and like to use (see Chapter 2)
- Choose external data sources carefully (see the "Accessing External Data" section of this chapter for details)
- Build applications that provide natural intrusion barriers (see Chapter 4)
- Test the reliability of the code you create, and carefully record both downtime and causes (see Chapter 5)
- Choose libraries, APIs, and microservices with care (see the "Using External Code and Resources" section of this chapter for details)
- Implement a comprehensive testing strategy for all application elements, even those you don't own (see Part III for details)

- Manage your application components to ensure application defenses don't languish after the application is released (see Part IV for details)
- Keep up-to-date on current security threats and strategies for overcoming them (see Chapter 16)
- Train your developers to think about security from beginning to end of every project (see Chapter 17)

Detecting Security Breaches

The last thing that any company wants to happen is to hear about a security breach second or third hand. Reading about your organization's inability to project user data in the trade press is probably the most rotten way to start any day, yet this is how many organizations learn about security breaches. Companies that assume a data breach has already occurred are the least likely to suffer permanent damage from a data breach and most likely to save money in the end. Instead of wasting time and resources fixing a data breach after it has happened, your company can detect the data breach as it occurs and stop it before it becomes a problem. Detection means providing the required code as part of your application and then ensuring these detection methods are designed to work with the current security threats.

Your organization, as a whole, will need a breach response team. However, your development team also needs individuals in the right places to detect security breaches. Most development teams today will need experts in:

- Networking
- Database management
- Application design and development
- Mobile technology
- Cyber forensics
- Compliance

Each application needs such a team and the team should meet regularly to discuss application-specific security requirements and threats. In addition, it's important to go over various threat scenarios and determine what you might do when a breach does occur. By being prepared, you make it more likely that you'll detect the breach early—possibly before someone in management comes thundering into your office asking for an explanation.

Remediating Broken Software

When a security breach does occur, whatever team your organization has in place must be ready to take charge and work through the remediation process. The organization, as a whole, needs to understand that not fixing the security breach and restoring the system as quickly as possible to its pre-breach state could cause the organization to fail. In other words, even if you're a great employee, you may well be looking for a new job.

The person in charge of security may ask the development team to help locate the attacker. Security Information and Event Management (SIEM) software can help review logs that point to the source of the problem. Of course, this assumes your application

actually creates appropriate logs. Part of the remediation process is to build logging and tracking functionality into the application in the first place. Without this information, trying to find the culprit so that your organization can stop the attack is often a lost cause.

Your procedures should include a strategy for checking for updates or patches for each component used by your application. Maintaining good application documentation is a must if you want to achieve this goal. It's too late to create a list of external resources at the time of a breach, you must have the list in hand before the breach occurs. Of course, the development team will need to test any updates that the application requires in order to ensure that the breach won't occur again. Finally, you need to ensure that the data has remained safe throughout the process and perform any data restoration your application requires.

Dealing with Cloud Storage

Cloud storage is a necessary evil in a world where employees demand access to data everywhere using any kind of device that happens to be handy. Users have all sorts of cloud storage solutions available, but one of the most popular now is Dropbox (https://www.dropbox.com/), which had amassed over 300 million users by the end of 2014. Dropbox (and most other cloud storage entities) have a checkered security history. For example, in 2011, Dropbox experienced a bug where anyone could access any account using any password for a period of four hours (see the article at http://www.darkreading.com/vulnerabilities-and-threats/dropbox-files-left-unprotected-open-to-all/d/d-id/1098442). Of course, all these vendors will tell you that your application data is safe now that it has improved security. It isn't a matter of if, but when, a hacker will find a way inside the cloud storage service or the service itself will drop the ball yet again.

> A major problem with most cloud storage is that it's public in nature. For example, Dropbox for Business sounds like a great idea and it does provide additional security features, but the service is still public. A business can't host the service within its own private cloud.
>
> In addition, most cloud services advertise that they encrypt the data on their servers, which is likely true. However, the service provider usually holds the encryption keys under the pretense of having to allow authorities with the proper warrants access to your data. Because you don't hold the keys to your encrypted data, you can't control access to it and the encryption is less useful than you might think.

Security of Web applications is a big deal because most applications tomorrow (if not all of them) will have a web application basis. Users want their applications available everywhere and the browser is just about the only means of providing that sort of functionality on so many platforms in an efficient manner. In short, you have to think about the cloud storage issues from the outset. You have a number of options for dealing with cloud storage as part of your application strategy.

- Block Access: It's actually possible to block all access to cloud storage using a firewall, policy, or application feature. However, the ability to block access everywhere a user might want to access cloud storage is extremely hard and users are quite determined. In addition, blocking access can actually have negative effects on meeting business needs. For example, partners may choose to use cloud storage as a

method for exchanging large files. A blocking strategy also incurs user wrath so that the users don't work with your application or find ways to circumvent the functionality you sought to provide. This is the best option to choose when your organization has to manage large amounts of sensitive data, has legal requirements for protecting data, or simply doesn't need the flexibility of using cloud storage.

- Allow Uncontrolled Access: You could choose to ignore the issues involved in using cloud storage. However, such a policy opens your organization to data loss, data breaches, and all sorts of other problems. Unfortunately, many organizations currently use this approach because controlling user access has become so difficult and the organization lacks the means of using some other approach.
- Relying on Company Mandated Security Locations: If you require users to access cloud storage using a company account, you can at least monitor file usage and have the means to recover data when an employee leaves. However, the basic problems with cloud storage remain. A hacker with the right knowledge could still access the account and grab your data or simply choose to snoop on you in other ways. This option does work well if your organization doesn't manage data with legally required protections and you're willing to exchange some security for convenience.
- Control Access Within the Application: Many cloud services support an Application Programming Interface (API) that allows you to interact with the service in unique ways. Even though this approach is quite time consuming, it does offer the advantage of letting you control where the user stores sensitive data, while still allowing the user the flexibility to use cloud storage for less sensitive data. You should consider this solution when your organization needs to interact with a large number of partners, yet also needs to manage large amounts of sensitive or critical data.
- Rely on a Third Party Solution: You can find third party solutions, such as Accellion (http://www.accellion.com/) that provide cloud storage connectors. The vendor provides a service that acts as an intermediary point between your application and the online data storage. The user is able to interact with data seamlessly, but the service controls access using policies that you set. The problem with this approach is that you now have an additional layer to consider when writing the application. In addition, you must trust the third party providing the connector. This particular solution works well when you need flexibility without the usual development costs and don't want to create your own solution the relies on API access.

Using External Code and Resources

Most organizations today don't have the time or resources needed to build applications completely from scratch. In addition, the costs of maintaining such an application would be enormous. In order to keep costs under control, organizations typically rely on third party code in various forms. The code performs common tasks and developers use it to create applications in a Lego-like manner. However, third party code doesn't come without security challenges. Effectively you're depending on someone else to write application code that not only works well and performs all the tasks you need, but does so securely. The following sections describe some of the issues surrounding the use of external code and resources.

Defining the Use of Libraries

A library is any code that you add into your application. Many people define libraries more broadly, but for this book, the essentials are that libraries contain code and that they become part of your application as you put the application in use. A commonly used library is jQuery (https://jquery.com/). It provides a wealth of functionality for performing common tasks in an application. The interesting thing about jQuery is that you find the terms library and API used interchangeably as shown in Figure 1-2.

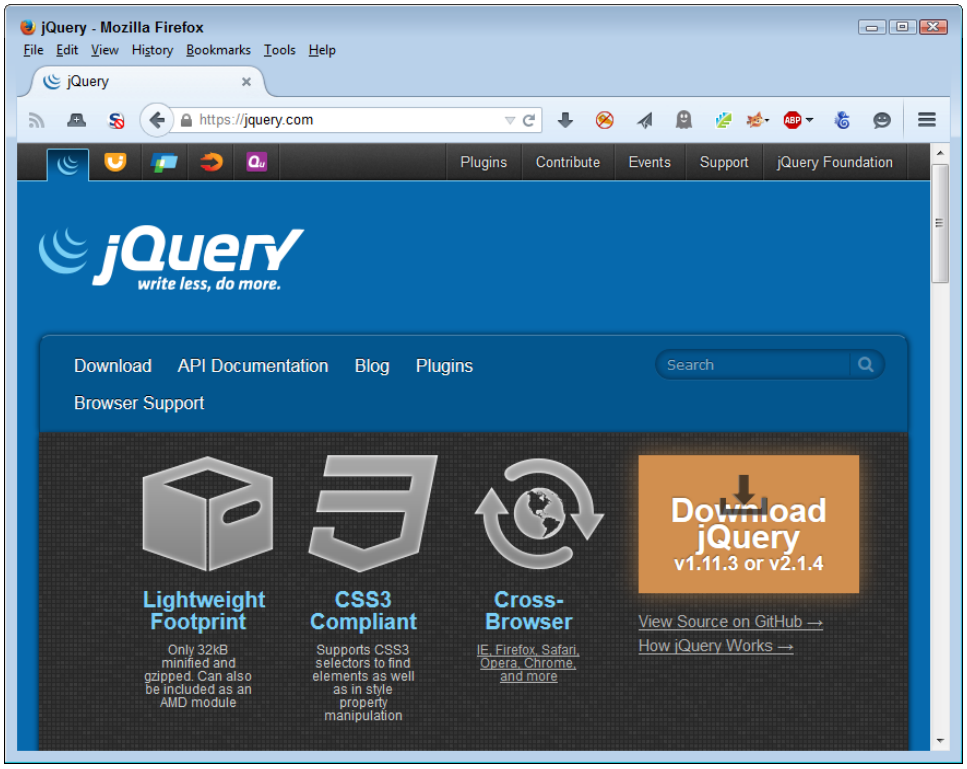


Figure 1-2. Many sites use library and API interchangeably

Looking at the jQuery site also tells you about optimal library configurations. In fact, the way in which jQuery presents itself is a good model for any library that you want to use. The library is fully documented and you can find multiple examples of each library call (to ensure you can find an example that is at least close to what you want to do). More importantly, the examples are live, so you can actually see the code in action using the same browsers that you plan to use for your own application.

> Like any other piece of software, jQuery has its faults too. As the book progresses, you're introduced to other libraries and to more details about each one so that you can start to see how features and security go hand-in-hand. Because jQuery is such as large, complex library it has a lot to offer, but there is also more attack surface for hackers to exploit.

When working with libraries, the main focus of your security efforts is your application because you download the code used for the application from the host server. Library code is executed in-process, so you need to know that you can trust the source from which you get library code. Chapter 6 discusses the intricacies of using libraries as part of an application development strategy.

Defining the Use of APIs

An Application Programming Interface (API) is any code you can call as an out-of-process service. You send a request to the API and the API responds with some resource that you request. The resource is normally data of some type, but APIs perform other sorts of tasks too. The idea is that the code resides on another system and that it doesn't become part of your application. Because APIs work with a request/response setup, they tend to offer broader platform support than libraries, but they also work slower than libraries do because the code isn't local to the system using it.

A good example of APIs is the services that Amazon offers for various developer needs (https://developer.amazon.com/). Figure 1-3 shows just a few of these services. You must sign up for each API you want to use and Amazon provides you with a special key to use in most cases. Because you're interacting with Amazon's servers and not simply downloading code to your own system, the security rules are different when using an API.

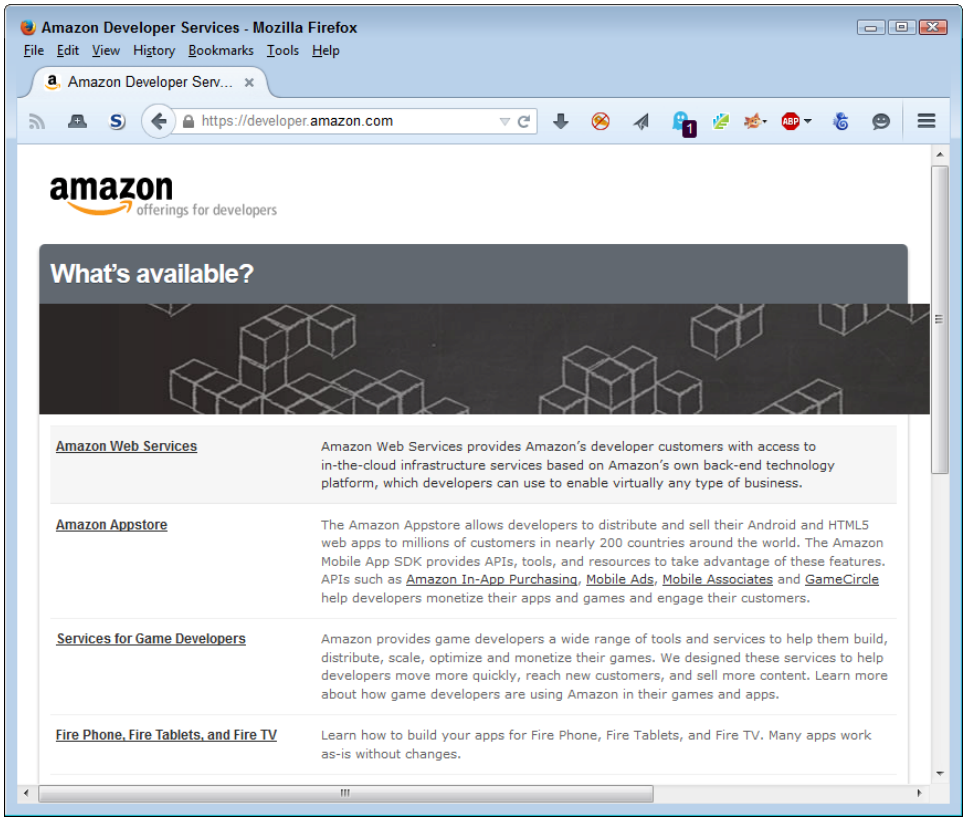


Figure 1-3. The Amazon API is an example of code that executes on a host server.

Each API tends to have a life of its own and relies on different approaches to issues such as managing data. Consequently, you can't make any assumption as to the security of one API when compared to another, even when both APIs come from the same host.

APIs also rely on an exchange of information. Any information exchange requires additional security because part of your data invariably ends up on the host system. You need to know that the host provides the means for properly securing the data you transmit as part of a request. Chapter 7 discusses how to work with APIs safely when using them as part of an application development strategy.

Defining the Use of Microservices

Like an API, microservices execute on a host system. You make a request and the microservice responds with some sort of resource—usually data. However, microservices differ a great deal from APIs. The emphasis is on small with microservices—a typical microservice performs one task well. In addition, microservices tend to focus heavily on platform independence. The idea is to provide a service that can interact with any platform of any size and of any capability. The difference in emphasis between APIs and microservices greatly affects the security landscape. For example, APIs tend to be more security conscious because the host can make more assumptions about the requesting platform and there is more to lose should something go wrong.

Current microservice offerings also tend to be homegrown because the technology is new. Look for some of the current API sites to begin offering microservices once the technology has grown. In the meantime, it pays to review precisely how microservices differ by looking at sites such as Microservice Architecture (http://microservices.io/). The site provides example applications and a discussion of the various patterns in use for online code at the moment as shown in Figure 1-4.

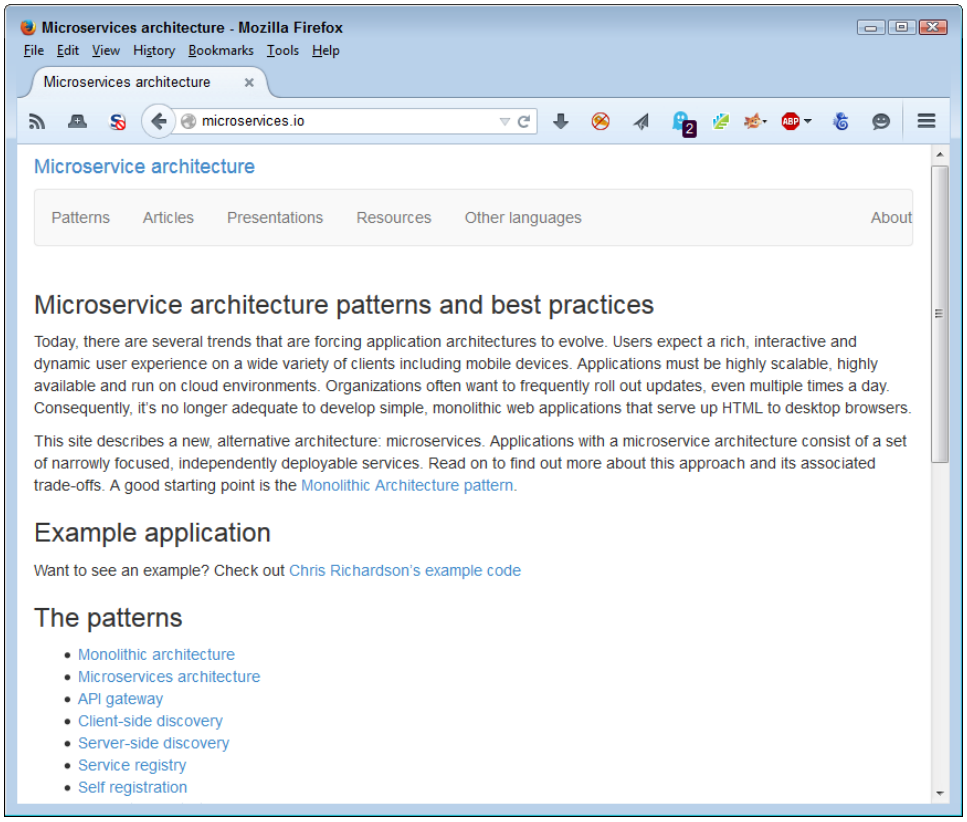


Figure 1-4. Microservices are new technology and it helps to see them in light of other usage patterns.

When working with microservices, you need to ensure that the host is reliable and that the single task the microservice performs is clearly defined. It's also important to consider how the microservice interacts with any data you supply and not to assume that every microservice interacts with the data in the same fashion (even when the microservices exist on the same host). The use of microservices does mean efficiency and the ability to work with a broader range of platforms, but you also need to be aware of the requirement for additional security checks. Chapter 8 discusses how to use microservices safely as part of an application development strategy.

Accessing External Data

External data takes all sorts of forms. Any form of external data is suspect because someone could tamper with it in the mere act of accessing the data. However, it's helpful to categorize data by its visibility when thinking about security requirements.

You can normally consider private data sources relatively secure. You still need to check the data for potential harmful elements (such as scripts encrypted within database fields). However, for the most part, the data source isn't purposely trying to cause problems.

- Data on hosts in your own organization
- Data on partner hosts

- Calculated sources created by applications running on servers
- Imported data from sensors or other sources supported by the organization

Paid data sources are also relatively secure. Anyone who provides access to paid data wants to maintain your relationship and reputation is everything in this area. As with local and private sources, you need to verify the data is free from corruption or potential threats, such as scripts. However, because the data also travels through a public network, you need to check for data manipulation and other potential problems from causes such as man-in-the-middle attacks.

> There are many interesting repositories online that you could find helpful when creating an application. Rather than generate the data yourself or rely on a paid source, you can often find a free source of the data. Sites such as the Registry of Research Data Repositories offer APIs now so that you can more accurately search for just the right data repository. In this case, you can find the API documentation at http://service.re3data.org/api/doc.

Data repositories can be problematic and the more public the repository, the more problematic it becomes. The use to which you put a data repository does make a difference, but ensuring you're actually getting data and not something else in disguise is essential. In many cases, you can download the data and scan it before you use it. For example, the World Health Organization (WHO) site shown in Figure 1-5 provides the means to sift through its data repository to find precisely the data you require and then to download just that dataset, reducing the risk that you'll get someone you really didn't want.

Figure 1-5. Some repositories, such as the WHO database, are downloadable.

There are many kinds of data repositories and many ways to access the data they contain. The technique you use depends on the data repository interface and the requirements of your application. Make sure you read about data repositories in Chapter 3. Chapters 6, 7, and 8 all discuss the use of external data as it applies to libraries, APIs, and microservices. Each environment has different requirements, so it's important to understand precisely how your code affects the access method and therefore the security needed to use the data without harm.

Allowing Access by Others

The vast majority of this chapter discusses protection of your resources or your use of data and resources provided by someone else. Enterprises don't exist in a vacuum. When you create a data source, library, API, microservice, or other resource that someone else can use, the third party often requests access. As soon as you allow access to the resource by this third party, you open your network up to potential security threats that you might not ever imagine. The business partner or other entity is likely quite reliable and doesn't intend to cause problems. However, their virus becomes your virus. Any security threat they face, becomes a security threat you face as well. If you have enough third parties using your resources, the chances are high that at least one of them has a security issue that could end up causing you problems as well.

Of course, before you can do anything, you need to ensure that your outside offering is as good as you think it is. Ensuring the safety of applications you support is essential. As a supplier of resources, you suddenly become a single point of failure for multiple outside entities. Keeping the environment secure means:

- Testing all resources regularly to ensure they remain viable
- Providing useful resource documentation
- Ensuring third party developers abide by the rules
- Performing security testing as needed
- Keeping up with potential threats to your resource
- Updating host resources to avoid known vulnerabilities

Developers who offer their resources for outside use have other issues to consider as well, but these issues are common enough that they're a given for any outside access scenario. In addition, you must expect third parties to test the resources to ensure they act as advertised. For example, when offering a library, API, or microservice, you must expect that third parties will perform input and output testing and not simply take your word for it that the resource will behave as expected.

Once you get past the initial phase of offering a resource for third party use, you must maintain the resource so applications continue to rely upon it. In addition, it's important to assume you face certain threats in making a resource offering. Here are some of the things you must consider:

- Hackers will attempt to use your resource to obtain access to your site

- Developers will misuse the resource and attempt to get it to perform tasks it wasn’t designed to perform
- The resource will become compromised in some manner

2

Embracing User Needs and Expectations

Security won’t work unless you are able to convince the user to embrace it. Any Draconian device developers contrive to enforce security without the user’s blessing will eventually fail because users are adept at finding ways around security. In situations where security truly is complete enough to thwart all but the hardiest user attempts, the user simply refuses to use the application. Long lists of failed applications attest to the fact that you can’t enforce security without some level of user assistance, so it’s best to ensure the user is on board with whatever decisions you make.

Users have two levels of requirements from an application and security must address both of them. A user needs to have the freedom to perform work-required tasks. When an application fails to meet user needs, it simply fails as an application. User expectations are in addition to needs. A user expects that the application will work on their personal device in addition to company-supplied devices. Depending on the application, ensuring the application and its security work on the broadest range of platforms creates good will, which makes it easier to sell security requirements to the user.

This chapter discusses both needs and expectations as they relate to security. For many developers, the goal of coding is to create an application that works well and meets all the requirements. However, the true goal of coding is to create an environment in which the user can interact with data successfully and securely. The data is the focus of the application, but the user is the means of making data-oriented modifications happen.

Developing a User View of the Application

Users and developers are often at loggerheads about security because they view applications in significantly different ways. Developers see carefully crafted code that does all sorts of interesting things; users see a means to an end. In fact, the user may not really see the application at all. All the user is concerned about is getting a report or other product put together by a certain deadline. For users, the best applications are invisible.

When security gets in the way of making the application invisible, the security becomes a problem that the user wants to circumvent. In short, making both the application and its attendant security as close to invisible as possible is always desirable and the better you achieve this goal, the more the user will like the application.

The problem with developers is that they truly don't think like users. The cool factor of abstract technology just has too big of a pull for any developer to resist. A development team should include at least one user representative (one who truly is representative of typical users in your organization). In addition, you should include users as part of the testing process. Security that doesn't work, like any other bug in your application, is easier to fix when you find it early in the development process. When security is cumbersome, burdensome, obvious, or just plain annoying, it's broken, even if it really does protect application data.

> Even though this book doesn't discuss the DevOps development method, you should consider employing it as part of your application design and development strategy. DevOps (a portmanteau of development and operations) emphasizes communication, collaboration, integration, automation, and cooperation between the stakeholders of any application development process. You can find a lot of DevOps resources at http://devops.com/. A number of people have attempted to describe DevOps, but one of the clearer dissertations appears in the article at http://www.jedi.be/blog/2010/02/12/what-is-this-devops-thing-anyway/. The article is a little old, but it still provides a great overview of what DevOps is, what problems it solves, and how you can employ it at your own organization.

Security is actually a problem that an entire organization has to solve. If you, as a developer, are the only one trying to create a solution, then the solution will almost certainly fail. The user view of the application is essential for bringing users on board with the security strategy you define for an application. However, you must also include:

- Management: To ensure organizational goals are met.
- Legal: To ensure data protection meets government requirements.
- Human resources: To ensure you aren't stepping on anyone's toes.
- Support: To ensure that any required training takes place.
- Every other stakeholder involved in defining business policies that control the management of data.

After all, enforcing the data management rules is what security is all about. It's not just about ensuring that a hacker can't somehow find the hidden directory used to store the data. Security is the informed protection of data such that users can make responsible changes, but damaging changes are avoided.

Considering Bring Your Own Device (BYOD) Issues

Users will bring their own devices from home and they'll use them to access your application—get used to it. Theoretically, you could create methods of detecting which

devices are accessing your application, but the fact is that users will find ways around the detection in many cases. Creating applications that work well in a BYOD environment is harder than working with a precise hardware configuration, but you can achieve good results. The main point is to assume users will rely on their own devices and to create application security with this goal in mind. The following sections describe the issues that you face when working with BYOD and provide you with potential solutions to these issues.

> Some organizations have actually embraced BYOD as a means to save money, which means that developers in these organizations have no standardized device to use for testing purposes. The organization simply assumes that the user will have a suitable device to use for both work and pleasure. If your organization is part of this trend, then you not only need to deal with BYOD devices as an alternative to company products, you need to deal with BYOD devices as the only option. It pays to know what sorts of devices your users have so that you have some basis on which to decide the kinds of devices to use for application testing and how to set device criteria for using the application.

Understanding Web-based Application Security

The solution that is most likely to make BYOD possible is to create web-based applications for every need. A user could then rely on a smartphone, tablet, notebook, PC, or any other device that has an appropriate browser to access the application. However, web-based applications are also notoriously difficult to secure. Think about the requirements for providing anything other than password security for all of the devices out there. In fact, the password might not even be a password in the true sense of the word—you might find yourself limited to a Personal Identification Number (PIN). The weakest security link for web-based applications in most cases is the mobile device. In order to make mobile devices more secure, you need to consider performing these steps:

1. Involve all the stakeholders for an application (including users, CIO, CISO, human resources, and other people outside the development team) in the decision making process for application features. You need to make this the first step because these individuals will help you create a strategy that focuses on both user and business needs, yet lets you point out the issues surrounding unmet expectations (rather than needs).
2. Develop a mobile security strategy and put it in writing. The problem with creating agreements during meetings and not formalizing those agreements is that people tend to forget what was agreed upon and it becomes an issue later in the process. Once you do have a formalized strategy, make sure everyone is aware of it and has read it. This is especially important for the developers who are creating the application design.
3. Ensure that management understands the need to fund the security measures. Most companies today suffer from a lack of resources when it comes to security. If a development team lacks resources to create secure applications, then the applications will have openings that hackers will exploit. The finances for supporting the development effort must come before the development process begins.

4. Obtain the correct tools for creating secure applications. Your development team requires the proper tools from the outset or it's not possible to obtain the desired result. In many cases, developers fall short on security goals because they lack the proper tools to implement the security strategy. The tools you commonly require affect these solution areas:
   a) User or system authentication
   b) Data encryption
   c) Mobile device management
   d) Common antivirus protection
   e) Virtual Private Network (VPN) support (when needed)
   f) Data loss prevention
   g) Host intrusion prevention
5. Create a partnership with an organization that has strong security expertise (if necessary). In many cases, your organization will lack development staff with the proper skills. Obtaining those skills from another organization that has already successfully deployed a number of web-based applications will save your organization time and effort.
6. Begin the development effort. Only you have created a robust support system for your application should you start the development effort. When you follow these steps, you create an environment where security is part of the web application from the beginning, rather than being bolted on later.

> A 2014 IDG Research Services report based on surveys of IT and security professionals describes a number of issues surround mobile device usage. The top concern (voiced by 75 percent of the respondents) is data leakage—something the organization tasks developers with preventing through application constraints. Lost or stolen devices comes in at 71 percent, followed by unsecure network access (56 percent), malware (53 percent), and unsecure Wi-Fi (41 percent).

Considering Native App Issues

The kneejerk reaction to the issues surrounding web-based applications is to use native applications instead. After all developers understand the technology well and it's possible to make use of operating system security features to ensure applications protect data as original anticipated. However, the days of the native application are becoming numbered. Supporting native applications is becoming harder as code becomes more complex. In addition, providing access to your application from a plethora of platforms means that you gain these important benefits:

- Improved collaboration among workers
- Enhanced customer service
- Access to corporation information from any location
- Increased productivity

Of course, there are many other benefits to supporting multiple platforms, but this list points out the issue of using native applications. If you really want to use native applications to ensure better security, then you need to create a native application for each platform you want to support, which can become quite an undertaking. For most organizations, it simply isn't worth the time to create the required applications when viewed from the perspective of enhanced security and improved application control.

Using Custom Browsers

In creating your web-based application, you can go the custom browser route. In some cases, that means writing a native application that actually includes browser functionality. The native application would provide additional features because it's possible to secure it better, yet having the web-based application available to smartphones with access to less sensitive features keeps users happy. Some languages, such as C#, provide relatively functional custom browser capability right out of the box. However, it's possible to create a custom browser using just about any application language.

> It's a good idea to discuss the use of smartphone kill switches with your organization as part of the application development strategy. A kill switch makes it possible to turn a stolen smartphone into a useless piece of junk. According to a USA Today article (http://www.usatoday.com/story/tech/2015/02/11/kill-switch-theft-down/23237959/) the use of kill switches has dramatically reduced smartphone theft in several major cities. A recent PC World article arms you with the information needed to help management understand how kill switches work (see http://www.pcworld.com/article/2367480/10-things-to-know-about-the-smartphone-kill-switch.html). In many cases, you must install software to make the kill switch available. Using a kill switch may sound drastic, but it's better than allowing hackers access to sensitive corporate data.

Beside direct security, the custom browser solution also makes indirect security options easier. Even though the control used within an application to create the custom browser likely provides full functionality, the application developer can choose not to implement certain features. For example, you may choose not to allow a user to type URLs or to rely on the history feature to move between pages. The application would still display pages just like any other browser, but the user won't have the ability to control the session in the same way that a standard browser will allow. Having this additional level of control makes it possible to allow access to more sensitive information because the user will be unable to do some of the things that normally result in virus downloads, information disclosure, or contamination of the application environment.

A custom browser environment also affords the developer an opportunity to rely on programming techniques that might not ordinarily work. Developers experience constant problems making third party libraries, frameworks, APIs, and microservices work properly because not every browser provides the requisite support. For example, in order to determine whether a particular browser actually supports the HTML5 features you want to use, you need to check it using a site such as HTML5test (https://html5test.com/) to obtain a listing of potential problem areas like the one shown in Figure 2-1.

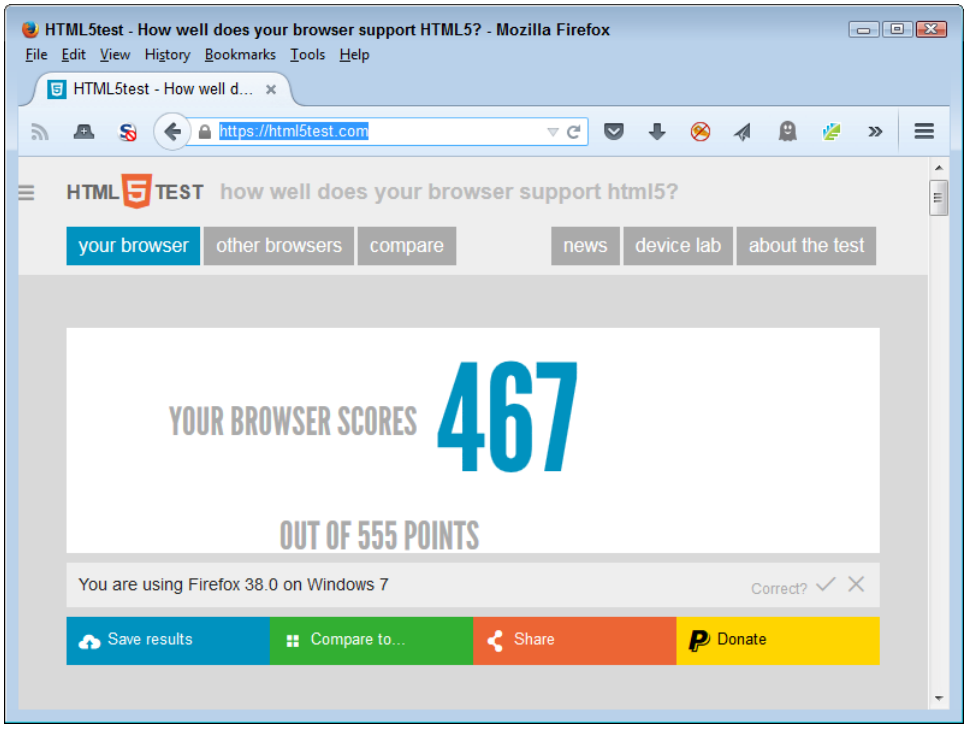


Figure 2-1. Using standard browsers means looking for potential support problems.

The problem with the customer browser solution is that it introduces disparities in user support depending on the device the user chooses to rely upon. When these sorts of disparities exist, the developer normally hears about it. Users will want to access the organization's most sensitive data on the least secure device available in the most public place. Imagine working on patient records in a Starbucks using a smartphone. The data could end up anywhere and a data breach will almost certainly occur. In some situations, the developer simply needs to work with everyone from managers on down to come up with a reasonable list of data handling precautions, which may mean that using the custom browser solution won't be popular, but it will tend to enforce prudent data management policies.

Verifying Code Compatibility Issues

The BYOD phenomena means that users will have all sorts of devices to use. Of course, that's a problem. However, a more significant problem is the fact that users will also have decrepit software on those devices because the older software is simply more comfortable to use. As a result of using this ancient software, your application may appear to have problems, but the issue isn't the application—it's a code compatibility problem caused by the really old software. With this in mind, you need to rely on solutions such as HTML5test (introduced in the previous section) to perform checks of a user's software to ensure it meets the minimum requirements.

Another method around the problem is to discover potential code compatibility issues as you write the application. For example, Figure 2-2 shows the W3Schools.com site that provides HTML 5 documentation (http://www.w3schools.com/tags/). At the bottom of

the page, you see a listing of the browsers that support a particular feature and the version required to support it. By tracking this information as you write your application, you can potentially reduce the code compatibility issues. At the very least, you can tell users which version of a piece of software they must have in order to work with your application when using their own device.

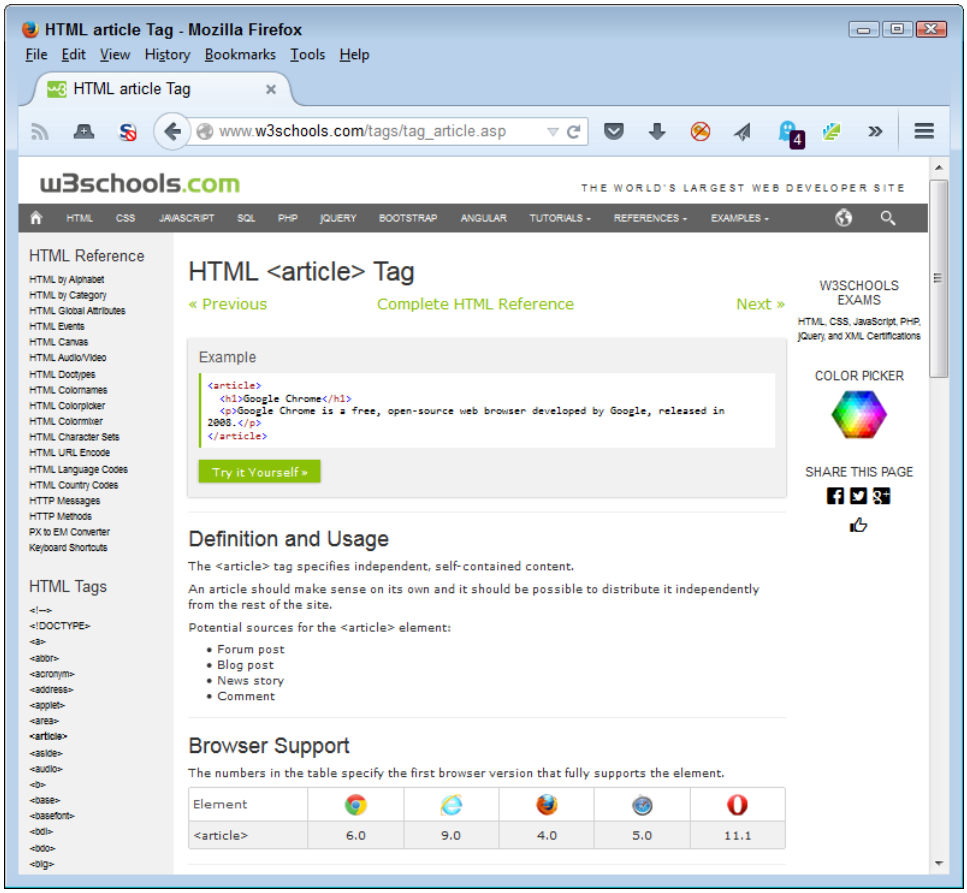


Figure 2-2. Verify that the code feature you plan to use actually works with user software.

It's also important to note that some sites also tell you about compatibility issues in a succinct manner. The W3Schools.com site also provides this feature. Notice that the list of HTML tags shown in Figure 2-3 tell you which tags HTML5 supports, and which it doesn't. Having this information in hand can save considerable time during the coding process because you don't waste time trying to figure out why a particular feature doesn't work as it should on a user system.

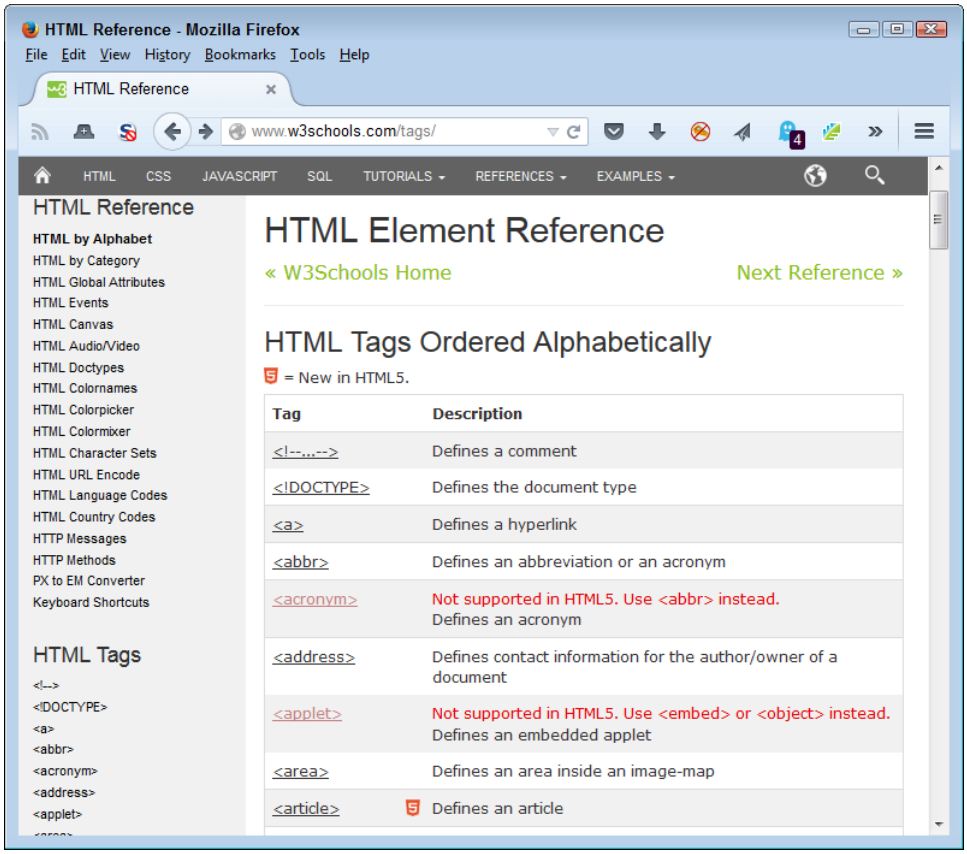


Figure 2-3. Finding documentation tells you about version issues in a succinct way is important.

A serious problem that most developers will experience with code compatibility is making HTML5 render properly with older browsers in the first place. Sites such as http://www.w3schools.com/html/html5_browsers.asp provide some answers you can use. For example, it shows how to use `html5shiv` to make Internet Explorer support HTML5 elements. The cdnjs site (https://cdnjs.com/) contains a large assortment of these helpful JavaScript add-ins. You can find them at https://cdnjs.com/libraries. Figure 2-4 shows just a small listing of the available libraries. Unfortunately, you need to find examples for all of these libraries because the site doesn't provide much in the way of helpful information. The majority of the library documentation appears on GitHub. For example, you can find the `html5shiv.js` documentation at https://github.com/afarkas/html5shiv.

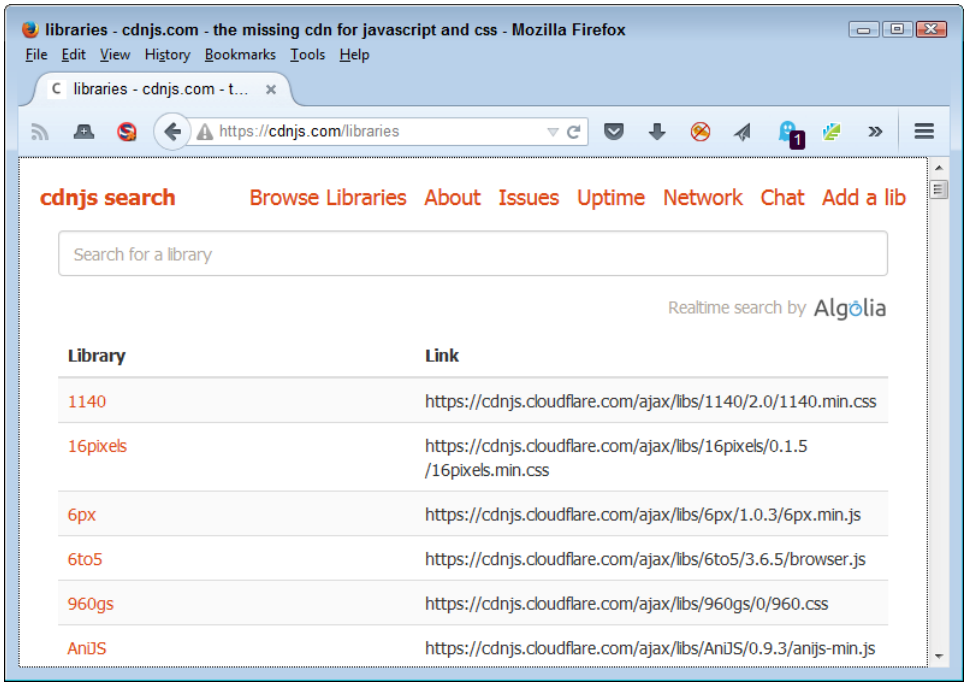


Figure 2-4. The cdnjs site provides you with a large number of helpful libraries.

You see the abbreviation CDN used all over the place online. A Content Delivery Network is a series of services that provides web content of various sorts. The main purpose of a CDN is to provide high availability and high-speed delivery. It also provides region appropriate content when needed. Consequently, cdnjs is simply a CDN specifically designed to provide access to JavaScript code and make it available to a large number of developers, similar to the way that the Google CDN (https://developers.google.com/speed/libraries/) performs the task.

Handling Nearly Continuous Device Updates

Your application will need to be flexible enough to handle all sorts of odd scenarios. One of the more common scenarios today is dealing with nearly continuous device updates. In this case, a user is perfectly happy using your application on a smartphone one day, but can't get anything to work the next. Common practice is for support to blame the user, but in many situations the user isn't at fault. At the bottom of the whole problem is that updates often take place without user permission or knowledge. Unfortunately, an update can introduce these sorts of issues:

- Code compatibility
- Security holes
- Lost settings
- Unbootable device

- Data damage

One of the ways around this issue is to make turning automatic updates off as part of the security policy for an application. Making the updates manually after you have tested the update as part of a rollout process is the best way to ensure that your application will continue to run. Unfortunately, this solution won't work. Users won't bring in their devices for testing unless something has already gone wrong. Even if users were willing to part with their devices, you likely won't have the resources needed to perform the required testing.

An alternative solution is to design applications such that they automatically check for updates during startup. If the version number of a product that the user relies on to work with the application changes, the application can send this information to an administrator as part of a potential solution reminder.

Creating flexible designs is also part of the methodology for handling constant updates. Even though a fancy programming trick to aid in keeping data secure looks like a good idea, it probably isn't. Keep to best practices development strategies, use standard libraries when possible, and keep security at a reasonable level to help ensure the application continues to work after the mystery update occurs on the user's device. Otherwise, you may find yourself spending considerable time trying to fix the security issues that's preventing your application from working during those precious weekend hours.

Devising Password Alternatives

Passwords seem like the straightforward way to identify users. You can change them with relative ease, make them complex enough to reduce the potential for someone else guessing them, and they're completely portable. However, users see passwords as difficult to use, even harder to remember, and as a painful reminder about the security you have put in place. (Support personnel who have to continually reset forgotten passwords would tend to agree with the users in this particular situation.) Passwords make security obvious and bothersome. Users would rather not have to enter a password to access or to make specific application features available. Unfortunately, you still need some method of determining the user's identity. The following sections present you with some mature ideas (those you can implement today) that you might find helpful in searching for your perfect solution.

A Word About Near Field Communication (NFC)

One of the more interesting developments in technology has been the use of NFC for all sorts of needs. NFC is an extension of the Radio Frequency Identifier (RFID) technology used mainly in products. You find the passive tags in just about everything you buy. The RFID tag doubles as a security device, but its main purpose is to identify the product. Radio waves from a scanner power the tag and the tag returns the information it contains. NFC is a high frequency subset of RFID that provides some special qualities you may need useful when creating security solutions for your application.

- The same device can act as a sender and tag.
- Device exchange information securely.
- You can reprogram a device as needed.
- The device contains both intelligence and local memory, making it more flexible than RFID.

NFC offers the potential to eliminate passwords. A person could have a chip embedded in a credit card sized ID badge. Tapping the badge on an NFC-enabled device would provide the required login. There are many NFC-enabled devices today, including PCs, tablets, and smartphones, so this solution could work everywhere.

The technology is also immature, so it's important to weigh your options carefully. Fortunately, the World Wide Web Consortium (W3C) is already working on standards for web development. You can see the working draft at http://www.w3.org/TR/nfc/. A library, API, or microservice that conforms to this standard can provide the resources needed to create robust applications that meet both organization and user needs, while reducing support costs and make the application significantly easier for the user to work with. Best of all, using an NFC solution lets you create a truly secure application.

Working with Passphrases

Passwords are hard to remember. Creating a password such as `!jS2Zd5L8` makes it really hard for hackers to guess the password and does improve the chances that your application and its data will remain safe. However, the password is also impossible to anyone to memorize, so users often write it down somewhere. In fact, the user might just keep a file right on the hard drive to make access easy. Of course, hackers know this and look for such files. In short, complex passwords are a great idea from a security perspective, but they show terrible planning from the user's perspective. It's possible, however, to get nearly the same level of security using a passphrase that's actually easy to remember.

A passphrase is a semi-human readable phrase of some sort. You use letters, numbers, and special symbols in combination to make the password complex. For example, you could create a passphrase such as `I luv fl0w3rs!` as a passphrase. The passphrase contains upper and lower case letters, numbers, and special symbols. It's resistant to dictionary and other common brute force attacks. It's probably longer than must users will ever allow for a password. Yet, it's still easy to remember. A user won't have to write the passphrase down.

It's important to choose phrases that the user will remember, but that aren't associated with the application, the user's personal life, or the user's work environment. There isn't a good reason to let the hacker have advantages when it comes to guessing the password. So, a passphrase such as `I Work in R00m 23a.` isn't a particularly good passphrase—it would be too easy to guess.

Simply telling the user to rely on passphrases isn't going to work. In fact, users will still use passwords such as `secret` and `master` because they've used them for so long. Users tend to reject anything that makes them work even a little harder. As a consequence, you must

still enforce complexity rules as part of your application code so that the user is forced to rely on something better than the usual suspects when it comes to passwords. If a user has to come up with a truly complex password, then the idea of using a passphrase instead becomes a lot more appealing.

Using Biometric Solutions

Biometric solutions rely on some unique characteristic of you as a person to generate a password. The idea behind a biometrics is that a biometric password is unique to a particular person, isn't easy to steal, the user can't lose it, and the password is complex enough that hackers can't guess it (at least not easily). The three most common biometric solutions in use today are:

- Fingerprint:
- Iris:
- Voice print:

All three of these solutions do have limitations and hackers have ways to overcome them, so you might choose another biometric or combine a biometric with some other form of authentication. Vendors have a number of biometric alternatives in progress. The following list describes some of these alternatives:

- Heartbeat: One of the more interesting biometric alternatives is to combine a heartrate monitor with analysis algorithms. In fact, the solution already exists in the form of the Nymi wristband (https://www.nymi.com/). This solution relies on NFC to transmit the user's password to any NFC-enabled device. The same wristband could log onto a computer, enable an application feature, open a house door, or start a car.
- Full Facial Recognition: In the movie Minority Report, cameras scan people's faces and present them with ads as they walk down the street. The interesting thing is that the technology already exists in the form of Facebook's Deepface (https://research.facebook.com/publications/480567225376225/deepface-closing-the-gap-to-human-level-performance-in-face-verification/). Simply by looking at your computer (webcam attached), you could log into the system and have all the required application features in place. In fact, Facebook recently claimed it could perform the scan from either the front or the side (http://money.cnn.com/2014/04/04/technology/innovation/facebook-facial-recognition/), which makes this solution relatively flexible when compared to other biometrics.

  Interestingly enough, all those selfies people are taking will make it quite easy for law enforcement and others to build a facial recognition database that will make it impossible for anyone to hide. Just think about the effect of every store camera being able to spot people based solely on their facial characteristics.

- Ear Shape: You hold your smartphone up to your ear, just like you would when making a call. However, instead of hearing someone talk to you, you get logged into an application. The solution already exists in the form of the Ergo Lock Screen App (http://www.descartesbiometrics.com/ergo-app/).

- Typing Technique: Every person has a different way to type. The speed at which you type, the time you hold down the keys, even the pauses between letters, all identify you as a typist. By typing a specific phrase and monitoring how you type it, an application could create a two-factor authentication that's indistinguishable from just typing the password. However, now a hacker stealing the password still wouldn't be able to use it. One company who has already implemented such a solution is Corsera in the form of Signature Track (see http://blog.coursera.org/post/40080531667/signaturetrack for details).

The promise of biometrics doesn't quite match the reality of biometrics. You already know from the descriptions that hackers have come up with methods for overcoming biometric passwords. If a user's password is compromised, you simply give the user a new password. However, if a user's fingerprint is compromised, it really isn't feasible to cut their finger off and sew a new finger on.

Why Use Two-factor Authentication?

A problem with most single-factor authentication solutions is that they have a weakness that someone can exploit with relative ease. For example, a hacker could steal either a password or a passphrase using a social engineering attack or even brute force. By using two-factor authentication, it's possible to reduce the risk of someone overcoming the authentication process. Of course, it could be argued that three-factor or four-factor authentication would be even better, but there is a point at which no one could ever get onto their own account.

There are many types of two-factor authentication. For example, you could supply a user with both a password and a token. You could also combine passwords with biometric solutions. Many banks currently use two-factor authentication and you can optionally use it on sites such as Google, Facebook and Twitter.

The problem with two-factor authentication is the same problem that occurs with single factor authentication—users typically don't like authentication at all. The feeling is that it should be possible to use the application without doing anything extra at all. Of course, authentication is important and you should use two-factor authentication for critical or sensitive data. However, it's important to consider the user's view of which authentication choices will work best. Creating a flexible solution is essential if you want this security solution to succeed.

Relying on Key Cards

Most people see key cards as an older type technology that still sees wide usage. For example, go to a motel and it's likely the person behind the desk will give you a key card with a specially encoded magnetic strip. The key card replaces the key that you might have received in the past. Organizations also use key cards to meet various needs, including controlling access to areas such as parking lots. A combination of a key card and a PIN often provides access to sensitive areas. Suffice it to say that you have probably used more than one key card at some point in your life.

Key card technology is constantly improving. Some modern key cards don't look like much of a card at all—they appear as part of a fob or other device that a user can put on a keychain or wear around the neck.

> By relying on RFID or NFC technology, it isn't even necessary for the user to slide the device, simply waving it in front of the lock works. The idea behind a key card hasn't changed though. You have a physical device that contains security information that the user relies upon for access to resources instead of using a password.

PCs can also come with key card technology. You can use the key card to control access to the PC as a whole, or to control access to a specific application. When used for a specific application, the developer needs to provide code to read the card, determine its validity, and authenticate its user.

The main advantage of using a key card is that it can provide a complex password and potentially other details to help identify the user. The password can be as complex as is needed to thwart hackers. Depending on the technology used, you may even be able to change the key card information with relative ease, so a password change need not be a costly thing to do. Key cards also tend to be large enough that users won't routinely lose them (although you can count on users misplacing at least some of the key cards). Because the technology has been around so long, key cards can also be relatively low cost when compared to other security solutions. Users also tend to like key cards (except when lost) because they're fast and easy to use.

The main disadvantage of key card technology is that users do misplace them or leave them at home. A lost key card could provide a hacker with the sort of access needed to do real harm to your organization. Even if the user simply forgets the key card at home, providing a temporary key card is an additional support cost. Of course, there is also the matter of actually getting the temporary key card back when the user no longer needs it.

> An interesting place to look at various key card technologies is Secura Key (http://www.securakey.com/). This site shows you many of the key card options you have available. You can even find keyboards, such as the IOGEAR model at http://www.iogear.com/product/GKBSR201/ that provide the required key card reader as part of the keyboard. The point is to reduce the user interaction required to use your application in order to reduce potential user error and data leakage.

Relying on USB Keys

A USB key is essential a flash drive that contains one or more passwords or tokens. You plug the USB key into your computer, start it up, and the computer uses the data on the USB key to log you into the system. The same key could contain passwords used to access applications. The application would need to be aware that the password is on the key, but the technique makes it possible to log into the application without actually providing a password or any other information. Google is currently working on a USB key setup (see http://www.technologyreview.com/view/510106/googles-alternative-to-the-password/) and you can be sure other vendors will follow. USB keys have some significant advantages over key cards:

- It's possible to change the passwords without obtaining a new USB key.
- The USB key can hold multiple passwords.
- The overall cost of USB keys is lower than using key cards.
- It's possible to upgrade the sorts of credentials that a USB key provides.

Of course, the USB key shares many of the same failings as the key card as well. For example, whether a user loses a key card of a USB key, the result is the same—the user can no longer log into the application and someone else could. In fact, given that USB keys are small than key cards, it's far more likely that a user will lose the USB key and compromise more than one password as a result.

Implementing a Token Strategy

You generally implement a token strategy using smartphones. A special site sends the user an SMS message, image, or sound that acts as the means to log into a computer. For example, Illiri (http://www.illiri.com/) sends a sound to the users smartphone that the user simply plays to the computer in order to log in. Likewise, Clef (https://getclef.com/) performs the same task using an image. In both cases, you can choose to send a different token every time the user logs into the system, which means that even if a hacker steals the token, it's essentially useless.

In most cases, an application uses a token as a second authentication method in a two-factor authentication system. The first authentication method is still a key card, biometric source, password, or passphrase. However, it's theoretically possible to use a token as a primary authentication method should you wish to do so. As with any other method of authentication, this one comes with a number of issues you need to solve:

- The user would need to have a smartphone with them at all times.
- Losing a smartphone could potentially compromise the token system as a means for logging into an application.
- This setup only works to log into devices other than the smartphone and many users today really do want to use their smartphone in place of a computing device whenever possible.
- The computer used to log the user in would need to have the requisite device for accepting the token.

Focusing On User Expectations

Up until this point of the chapter, you have focused on user needs. No user wants authentication, but every user needs authentication. Expectations fall into the nice to have category when it comes to development, but implementing them can create good will, which makes the user a little more open to actually using all those need to have items you added to the application.

Of course, some users have unrealistic expectations, such as an application that makes them look especially appealing to someone else or does all their work for them while they play solitaire and watch movies all day. Unfortunately, even the best developer in the world could never address expectations of that sort. The following sections describe some reasonable user expectations.

Making the Application Easy to Use

The chapter has made a point of emphasizing the need for ease-of-use. Users don't want to know the details about the security solution you implement. In fact, most users really don't want to know anything about the application other than it can provide the data

needed to accomplish a task that matters to the user. In the long run, anything you can do to make security invisible to the end user increases the chances that the user will actually participate in the secure management of data and that the strategy will succeed in keeping data safe.

The counterpoint to the need for ease-of-use is creating a security scenario that works as advertised. Just one data breach can ruin a company's reputation and cost your organization buckets of money. According to a recent ComputerWorld article, a data breach now costs $154.00 on average per record (see the article details at http://www.computerworld.com/article/2926775/security0/data-breach-costs-now-average-154-per-record.html). The cost for each record lost will continue to increase, so the need for solutions that are safe continues to increase. The user expectation is ease-of-use, but the application reality is the need to keep data secure no matter what it takes to do so.

Making the Application Fast

Many developers don't understand the user's need for speed at all costs. Of course, part of the problem is that the user's attention span has decreased over the years. According to an article on The Guardian (http://www.theguardian.com/media-network/media-network-blog/2012/mar/19/attention-span-internet-consumer), you have between one and five seconds to grab the user's attention. Users want instant gratification and quick fixes. Deep thinking and in depth analysis are no longer part of the user's repertoire. Unfortunately, the more security you add to an application, typically, the slower it becomes. If you were to ask most users how much time they want to spend on security, the answer would be zero. User's truly don't care about security—they want data fast.

The user expectation then is that there won't be any sort of wait for the data needed to perform a task. In addition, the data needs to be accurate and in a form the user needs at the outset. Anything less tends to breed frustration. The reality is that security will slow the application, but that you, as a developer, will need to concentrate on keeping delays to a minimum.

In addition, a secure environment means that not all data will be available all the time. You really can't permit a user to look at patient records while sitting at the local Starbucks sipping a latte. It's an unreasonable expectation on the part of the user to have this level of access (not that it will keep users from asking for it). So, a fast application is one that authenticates the user quickly and then presents the user with legal choices for data access. Hiding illegal options will often keep the user from looking for them in the first place, but you also need to provide help that specifies why certain data manipulation options are missing.

Creating a Reliable Environment

Above any other consideration, you must make the application reliable. The application can't behave in an unpredictable manner because the user will become frustrated and not use it. Even if the application is the tiniest bit slow, a user will complain less about speed than an application that truly is fast, but performs tasks incorrectly. It's also essential that the security and the data manipulation features work flawlessly. Chapter 5 discusses the various techniques you use to create a reliable application. Part III of this book is all about various kinds of testing. You need to take testing seriously to create a reliable application that generates few user complaints.

The user expectation in this case is that the application performs flawlessly in every situation. Again, it’s a matter of making the application transparent so that all the user has to focus on is getting a task completed. In the real world, you can come close to flawless with proper coding techniques, testing techniques, threat analysis, and constant checks on issues such as device updates. So, in this one particular case, a user expectation comes quite close to being something you can provide in reality.

Keeping Security in Perspective

If you take nothing else away from this chapter, it’s the fact that security is a balancing act. Unless you keep security in perspective, your application may not protect data enough or encumber the user to the breaking point by implementing Draconian features that don’t really serve a useful purpose. Users are quick to spot functionality that they not only dislike, but apparently doesn’t serve a useful purpose.

Even though it might appear that the user expectation is that applications don’t need security, most users realize that security does have a place, especially if the user wants to keep the organization healthy. It’s not really a matter of selling a user on the need for security, it’s more a matter of selling the user on a specific level of security. When you keep security in perspective, it’s a lot easier to close the deal and have the user on your side.

3

Getting Third Party Assistance

Reinventing the wheel is always a bad idea. It's entirely possible that someone else has already created a security solution that meets your needs well enough that you can use it instead of creating your own security solution. The third party solution needs to appear as part of your security plan so that others know that you have a solution in mind. In addition, by having the third party solution in your plan, you can include all sorts of information for it and invite discussion about the solution. This chapter discusses how you can add various kinds of third party solutions to your security plan.

Of course, before you can do anything, you must discover the third party solution you want to use. Fortunately, there are techniques you can use to reduce the time required to perform the process and then research the individual solutions to ensure they truly do meet your needs. Once you do discover the solutions you want to use, it's important to consider the pros and cons of each solution type. This chapter groups solutions into those that exist in the cloud and those that you add into your own application, such as a library or API. The following sections help you discover that third party solutions really can meet your needs and reduce the time and effort to create a working solution.

> Even though this chapter does discuss specific examples, the information provided applies to a category as a whole. For example, even though Capterra appears as a potential review site for products, other such sites exist, and you need to choose the site that best matches your philosophy of application design and development. The specific example simply helps illustrate the principle at hand.

Discovering Third Party Security Solutions

Finding a third party security solution can be difficult. Enter most search terms you can think of into a search engine and you get a list of individual security solutions back, many of which may not even relate to your particular problem. Search engines provide you with a haystack, not a needle. Unfortunately, they also remove the magnet so that finding

the needle is impossible. You really do need a better way to find the security solution of your dreams.

There is help in the form of review sites such as Capterra (http://www.capterra.com/network-security-software/), shown in Figure 3-1, provide help in the form of filtering that helps you make sense of the various vendors who are vying for your attention. Each of the entries has a short review that you can click to find out additional details. In many cases, the review also includes media that could provide a demo of the product or other useful information. The fact that the reviews are in one place, filtered by the criteria that you specify, and formatted in essentially the same way makes the search process easier.

Figure 3-1. Search sites such as Capterra make looking for third party resources easier.

Relying on magazine reviews can also be helpful, depending on the magazine you choose. For example, SC Magazine (http://www.scmagazine.com/) regularly reviews products that could be helpful to the security professional. In fact, you can get the reviews delivered to your inbox (see http://www.scmagazine.com/events/section/109/). The magazine even sponsors competitions of a sort so that you can determine the most viable solutions available (see http://www.scmagazine.com/2014-sc-awards-us-finalists/section/3694/).

> Review sites of any sort are going to contain bias. Even if the site reviews products fairly, using criteria that you can examine and processes that you can verify, a review is still a matter of someone's

> opinion and you need to consider whether the person performing the review is informed enough to provide you with the information you need. Over time, you develop a sense that particular sites match your opinion of what really is a good product.

In some cases, you can find organizations that provide snapshots of their members that tend to provide you with consistent information. For example, the Cloud Security Alliance (CSA) (https://cloudsecurityalliance.org/membership/solution-providers/) falls into this category. By checking out these sites, you can get a quick overview of the players involved in specific security solutions. Of course, the problem is that the organizations provide the write-ups you read, so the information is biased. You can be sure that the vendors supporting the organization have left out any potentially negative aspects of their various products. You need to read these sorts of information sources carefully and then research the potential problems yourself.

> The good part about organizational sites is that you often find other helpful sorts of information because it's in the site's best interest to keep you involved. Some organizational sites also tell you about upcoming events where you can get additional information about potential solutions, meet with the vendors in person, and discover new techniques that you might not have known about before.

Some online magazine sites feature articles that can provide significant helpful information. For example, the article at http://www.technologyreview.com/news/424298/improving-the-security-of-cloud-computing/ discusses techniques you can use to keep your data safe. The two solutions provided as part of the article point to issues you might not have considered. In the first case, you read about how computer scientist at the University of California, San Diego and MIT demonstrated a need for each organization to have its own virtual server (as a result, Amazon changed the way it does things). In the second case, a new technology breaks your data into 16 pieces. You can then recover the data by using any ten remaining pieces should some sort of failure occur. It's important to remember that articles normally have a goal, so you need to keep the goal in mind as you decide on the viability of the products featured in the article for meeting your organization's needs.

Considering Cloud Security Solutions

Organizations store a lot of data in the cloud today to keep costs under control and make the data easily accessible from any location the company may require. Cloud computing, which includes cloud data storage, is there to stay because it simply makes sense to use it instead of relying on custom solutions.

The problem with cloud computing is that it opens your organization to all sorts of problems. For example, you have no way of knowing that the hosting company will keep your data safe. There are way too many stories in the media about organizations that have had data hacked with devastating consequences. In fact, you can easily find stories about how easy it is to hack cloud-based storage, such as the one at http://www.techtimes.com/articles/14800/20140903/cloud-hacking-isnt-hard-think.htm.

The following sections provide some ideas on how you can use third party solutions to secure your online data. The sections consider three scenarios: data repositories, file

sharing, and cloud storage. You may have to use combinations of solutions to create a complete package for your organization. However, these solutions provide you with a good start and will ultimately save time and effort on your part. Best of all, because someone else maintains the solutions, you save money in the long run as well because you aren't constantly fixing a custom solution.

> A good rule of thumb to remember is that the best way to keep a secret is not to tell anyone. If you have storage requirements for data that simply can't see the light of day, storing it online is probably the worst possible solution. No matter how well you protect your data, if someone is certain they want to obtain access to it, they will. It's always easier to hack security than to build it. Consequently, if you have data you must legally keep secure under every potential scenario, then using local company storage that you control is the best idea. The article at http://computer.howstuffworks.com/cloud-computing/files-safe-in-the-cloud.htm tells you just how many different ways hackers can use to access your data and you can be certain that the hackers will come up with more.

Understanding Data Repositories

A data repository can be many things. Just try to find a consistent definition for one online and what you'll end up with is a new meaning for every site you try. The fact of the matter is that term data repository means something slightly different to everyone that uses the term. However, most people will generally agree that a data repository is a centralized data storage location for information that an organization is maintaining as part of an organization's knowledge base. Using data mining techniques, the organization can probe this knowledge base and actually create new knowledge from it. Of course, there are many other implications of data repositories, but the bottom line is that you're talking about a lot of data in most cases, some of it maintained, some of it stored for historical reasons. Keeping data of this sort safe is a big job.

Data repositories abound. You can find a host of open data repositories on sites such as Open Access Directory (OAD) (http://oad.simmons.edu/oadwiki/Data_repositories), as shown in Figure 3-2. You might actually use some of these repositories in your application. So, security isn't simply a matter of keeping your private repository safe, but also ensuring that any public repository you use is also safe. After all, a hacker doesn't really care too much about the means used to access your application and ultimately your organization—all that matters is that the access happens.

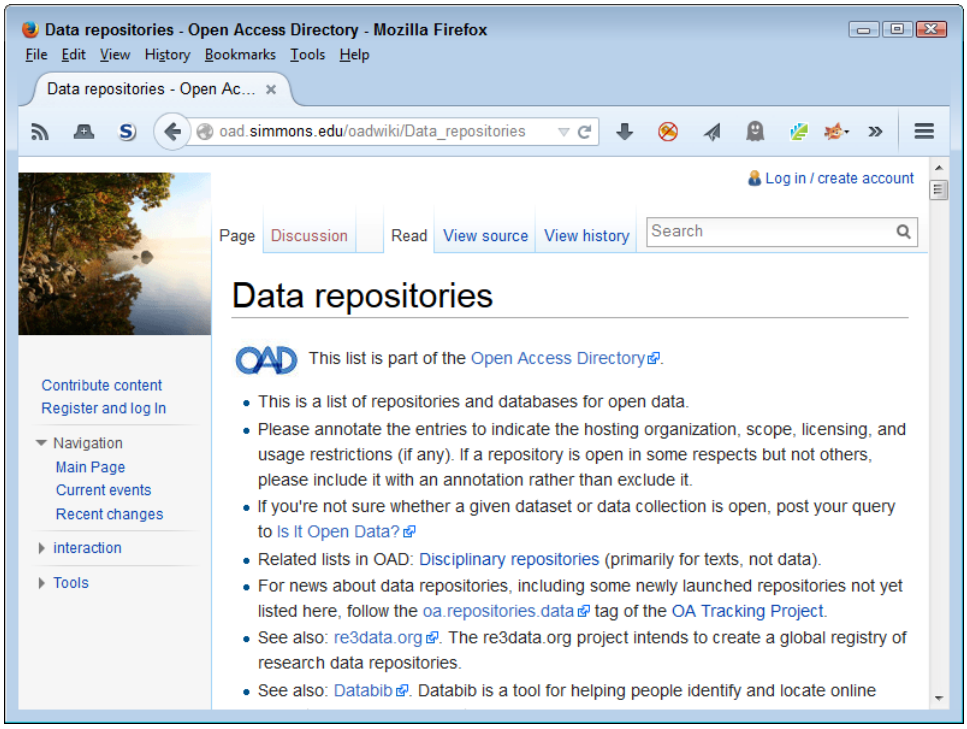


Figure 3-2. Open access repositories provide essential information for some organizations.

Few data repositories contain data from just one project or affect just one project. In fact, there is little point to creating such as repository. Most data repositories contain data from a huge number of projects. For example, sourceforge.net contains 90,000+ projects that include every sort of programming project you can think of, including kernel mode software. Traditional approaches to keeping these data repositories secure, such as relying on the administrator to patch the server and manage access to it, don't work in this case because it's too easy for maintenance items to fall between the cracks.

A new type of protocol called Secure Untrusted Data Repository (SUNDR) from Secure Computer Systems Group (https://www.yumpu.com/en/document/view/34269846/secure-untrusted-data-repository-sundr-usenix) attacks the problem from the client perspective. A client can actually detect modifications to the file. Even if the server is untrusted or compromised in some way, the client can still detect the potential breach in security. The way in which this system work depends on logs maintained on block server and a consistency server using methods that would be hard (but never impossible) to break. You can see a slideshow presentation of the technology at http://slideplayer.com/slide/3389212/.

It's important to remember that these new technologies will work hand-in-hand with existing technologies, such as the security provided by a database manager. They also don't dismiss your responsibility as a developer for including security as part of your solution. However, what technologies such as SUNDR do provide is another level of defense—this one at the server level and not dependent on the server or the administrator to maintain.

Dealing with File Sharing Issues

File sharing used to mean creating a cumbersome configuration that users hated to use, when it worked at all. In order to create a file sharing scenario, the organization needed to set up a special file server and create a Virtual Private Network (VPN) to access it. For a developer, it meant having to write tons of code to work around the problems associated with accessing the data in such an environment, along with an equally large amount of error trapping code to reduce user frustration when the setup failed to work as planned. Moving data from private networks to cloud solutions maintained by third parties who have the deep pockets required to fund such a solution seems like an obvious solution. An organization can save 65 percent or more by moving data to a cloud provider instead of hosting the data on a private server. Developers gain well-considered and documented APIs to make accessing the data easier. Users gain because there is a lot less frustration in using a publicly accessible file sharing solution and such solutions usually work with every device the user owns.

> It's essential to realize that any publicly available file sharing service is going to create potential security breaches. No matter how high the host builds the walls, a hacker will come along and dig under them to access your data. Using any sort of file sharing service incurs risks beyond what you might expect when using a VPN. Even though many organizations use file sharing services successfully, keep in mind that a VPN is generally more secure and you may have to choose a VPN solution for some storage needs, even though the costs are significantly higher.

When most people think about file sharing in the cloud today, they think about products such as Dropbox (https://www.dropbox.com/business). It's true that Dropbox does have an Application Programming Interface (API) (https://www.dropbox.com/developers) that developers can use to create interfaces for their applications. The API provides full functionality and you can use it to meet a variety of needs as shown in Figure 3-3. Of course, Dropbox has taken headlines for security issues as of late (see http://www.zdnet.com/article/dropbox-drops-the-security-notification-ball-again/ and http://www.newsweek.com/wary-privacy-issues-ditch-dropbox-and-avoid-google-says-edward-snowden-276956 as examples) and others are just as ready to tell you about solutions for fixing this problems (see http://www.pcworld.com/article/2918524/how-to-make-dropbox-more-secure-without-spending-a-cent.html).

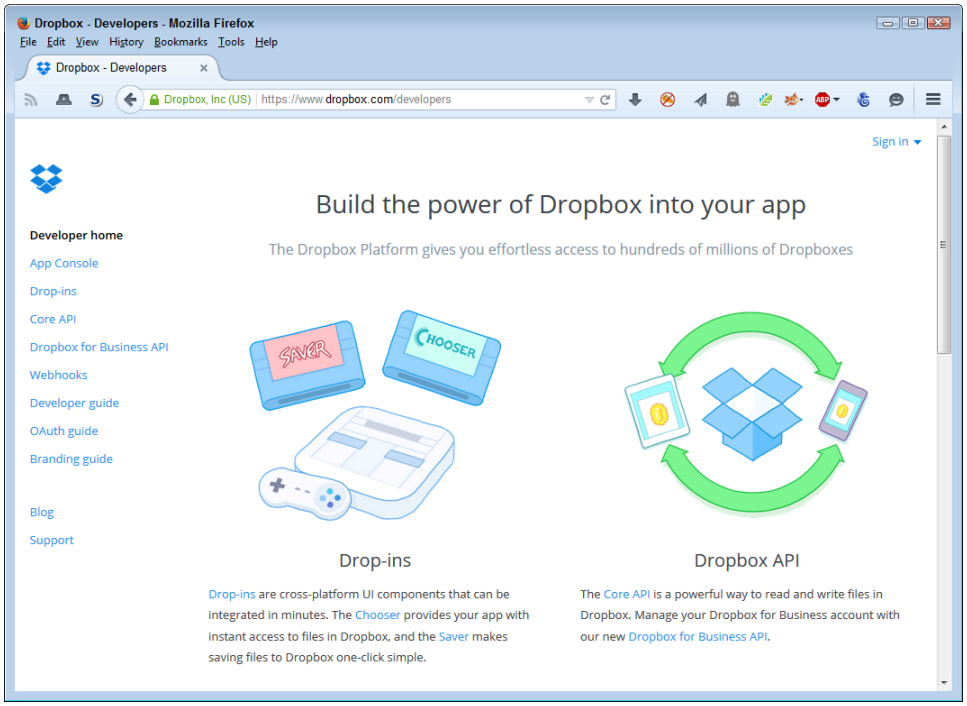


Figure 3-3. The Dropbox API makes adding online file sharing easy to add to your application.

You actually have a number of solutions you can try when creating a file sharing solution for a Small-to-Medium-sized Business (SMB). They all come with security issues, so you need to research each one to determine which security issues you'd prefer to deal with. Some, like Dropbox, attract large headlines, but they also have good viable solutions for their security issues. Here is a list of the most common file sharing solutions (besides Dropbox) in use today:

- Acronis Business (http://www.acronis.com/en-us/business/overview/)
- Box (https://www.box.com/)
- Carbonite Business (http://www.carbonite.com/online-backup/business/how-it-works)
- Citrix ShareFile (http://www.sharefile.com/)
- CrashPlan Pro (http://www.code42.com/business/)
- Engnyte (https://www.egnyte.com/)
- Google Drive (http://www.google.com/drive/start/index.html)
- Hightail (https://www.hightail.com/) (Formerly YouSendIt)
- Microsoft OneDrive (https://onedrive.live.com/about/en-us/)
- MozyPro (http://mozy.com/product/mozy/business)
- SpiderOak (https://spideroak.com/)
- SugarSync for Business (https://www.sugarsync.com/business/)
- SyncPlicity (https://www.syncplicity.com/)

Choosing a third party file sharing solution can be difficult. Your organization will have specific goals when choosing a file sharing solution, such as the cost/benefits ratio. However, as a developer, you can other concerns. Here are the sorts of things that you should consider when asked about a file sharing service:

- Well-documented API: In order to write custom code against the file sharing solution and make some types of access invisible, you need to have a well-documented API.
- Security: A cloud-based file sharing solution requires a number of unique security features to make it successful.
    - Cloud Tiering: The most important of these features is cloud tiering, where you can define how the file is stored. Some files appear only on the file sharing service, some have a local copy in addition to the cloud copy, and some appear only on local drives, despite being available through the file sharing service.
    - File Typing: Some file types may have special storage requirements. These files could contain sensitive data and you may need to store them locally or encrypt them even if they should appear in a different manner based on the file's tier.
    - Access Frequency: Your application may only access files at a specific frequency. When something accesses the file at a different frequency, it could indicate a security breach. The file sharing service should alert you to this potential issue.
- Global Namespace: In some cases, a file sharing service will provide access in a manner that creates data silos, where accounting has its data completely separate from human resources. Data silos do serve a purpose, but sometimes you need data stored in a manner that anyone can access the data for which they have rights from any location using any device. To obtain this sort of access, you need a global namespace.
- Redundancy: In order to keep writing code and not have to service user needs, you require a file sharing solution that has great redundancy. Otherwise, you can't ensure the data a user needs will be available at any given time. However, as part of data redundancy, you need to ensure the data is stored securely at each location using individual virtual servers (several types of shared file attack depend on the attacker gaining access to the target's server—something that becomes harder when the file sharing service relies on virtual servers attached to individual clients).

Considering Cloud Storage

The term, cloud storage, encompasses a number of needs. Two of these needs have special mentions in the chapter: data repositories and file sharing services. However, your organization may have other cloud storage needs. It's usually easier if you can build an application to use a single host, but it may not always be possible. When choosing a host, you must also consider these additional cloud storage requirements:

- Archiving: Data archiving is different from other sorts of cloud storage in that you need to ensure the data is safe, rather than necessarily accessible. If a major event occurs, you want to ensure that your data remains safe. Organizations used to rely on tapes and other media stored in an offsite location to ensure data security, but today cloud storage addresses this need. Any application you build may have to archive data and ensure that the archival occurs in a secure manner.

- Settings Storage: Your users will want to use all sorts of devices to access all sorts of data in all sorts of ways. Users have no clue that applications require settings to work well and that storing settings locally makes it impossible to create flexible applications that work everywhere. Unfortunately, settings stored in the cloud also make it easy for hackers to guess application features and potentially hack into your organization, so you need to ensure the settings storage is encrypted and is hosted in such as manner as to make any breaches obvious.
- Media Storage: Users require access to all sorts of media today, including video, audio, photographs, presentations, and so on. Your application may need to manage the media as part of its purpose, which means relying on features such as tiering to ensure the application and cloud storage handles the data correctly. Your user should only have access to required media and not all the media the organization has to offer.

There are other types of cloud storage that aren't discussed in this book. For example, most developers don't have much of an interest in e-mail unless they're writing an e-mail application. The point is that storage encompasses a broad range of types and requirements that you need to consider as part of your security solution.

Choosing Between Product Types

As you go through this book, you discover how to use various product types to make your coding efforts significantly easier. It's important to understand that you can categorize coding products in three ways: libraries, APIs, and microservices. Each coding resource type has a specific purpose and helps you meet specific needs. It's possible to mix and match product types in a single application, but you need to ensure that the products are used in such a manner that they don't affect security in a negative way. The following sections discuss the different product types and helps you understand how you can use them as part of an application strategy. More importantly, you get a glimpse of some of the security issues for each of the product types.

##Using Other People's Code##

Any time you use other people's code, you take a chance that some hacker is going to discover a way to overcome the security measures offered by that code. The benefits of doing so are huge. By creating a single security breach, the hacker can potentially gain access to the applications written by everyone who uses the library, API, or microservice to which the hacker has gained access. The same hack works everywhere, which means that the hacker can select which sites to invade and what data to acquire. Some hacks are so successful that a hacker gains complete control over a data source and does things like hold the data hostage unless the data owner pays up.

The benefit to using other people's code is that you can create an application in a fraction of the time it would normally take using fewer people. The costs of support and maintenance are also significantly lower. When someone discovers a problem in the library, API, or microservice, the owner of that code performs the fix, not you. Without performing any work at all, you automatically gain all the benefits of the fix that the third party provides.

> The third party also provides enhancements that make the code faster, more efficient, or more secure. When new technologies appear on the scene, the third party provides the updates needed to make your application work in new ways. If users start requiring access a new device type, the third party will create the code needed to make that device function properly. In short, there are many good reasons to use other people's code, but you have to be wary when doing so.

Working with Libraries

Libraries are code that exists in separate files, but you load into your application. The library becomes part of your application. In most cases, you can download libraries to a local server, but in other cases you can't. Downloading a library to a local server has the benefit of not allowing others to see how you're using the library to create an application. This practice can also increase application speed by making the loading process faster and easier. In addition, downloading the library means that the library code remains stable, which makes it possible to create more reliable applications.

From a security perspective, it might seem as if downloading a library to a local drive is the best bet. You gain all the benefits of using someone else's code and don't have to expose your application to potential hacks based on seeing how your use the library. The downside, from a security perspective, is that using a local copy also means you don't get automatic updates. This means that your application could contain bugs that the library developer has already fixed and actually make it easier for a hacker to gain entry to your system.

When working with libraries, you need to consider the originator's reputation, as well as they manner in which you use the library in your application. Because a library integrates directly into your application, you need to consider what type of access the library gains to your application internals. It's important to code defensively when using libraries because you don't want to tell others too much about how your application works. Some examples are libraries are:

- D3.js (http://d3js.org/)
- Google Web Toolkit (GWT) (http://www.gwtproject.org/)
- jQuery (http://jquery.com/)
- jQuery UI (http://jqueryui.com/)
- jQuery Mobile (http://jquerymobile.com/)
- MooTools (http://mootools.net/)
- PDF.js (http://mozilla.github.io/pdf.js/)
- QUnit (http://qunitjs.com/)
- SWFObject (https://code.google.com/p/swfobject/)
- YUI Library (http://yuilibrary.com/)

Accessing APIs

APIs are code that you access from your application by making calls to a central location. The APIs exist as a separate entity and you call on that entity from your application. The

point is that APIs are completely separate from the other code of your application, so you need to create a reference to it and then make requests to the API. An API is normally attached to some type of service, such a data storage. You use APIs to create a client/server kind of connection, with all the security issues that such a connection implies.

A potential problem with APIs is that they can be slow. Your application becomes less efficient and experiences delays. This is an important consideration because users aren't known for their patience and could end up doing something that will inadvertently break your application while waiting. Broken applications always present security issues that hackers are only too happy to exploit.

It's also possible for hackers to use man-in-the-middle attacks to access your data when using an API. A man-in-the-middle attack is especially hard to detect because the calls apparently work and you don't see any difference in the data when you retrieve it later. In the meantime, the hacker uses the data collected to do things like retrieve customer credit card numbers or obtain usable information about your organization. When working with an API, you have to pay close attention to issues such as data encryption to ensure your data remains safe. Some examples of APIs are:

- AccuWeather (www.programmableweb.com/api/accuweather)
- Amazon (https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html)
- Box (https://developers.box.com/)
- Facebook (https://developers.facebook.com/)
- Flickr (https://www.flickr.com/services/developer/)
- Google (https://developers.google.com/products/)
- Pinterest (http://www.programmableweb.com/api/pinterest)
- Salesforce (http://www.salesforce.com/us/developer/docs/api/index.htm)
- Twitter (https://dev.twitter.com/overview/documentation)
- WordPress (http://codex.wordpress.org/WordPress_APIs)
- YouTube (https://developers.google.com/youtube/)

> One of the differences between libraries and APIs is that most libraries offer free use, while many APIs cost money to use. In most cases, you can gain access to an API at a level sufficient for testing purposes, but to gain full API functionality you must obtain a key, which means paying for the level of access you require.
>
> In addition, many API hosts require that you test your application in sandbox mode and obtain a certification for the resulting debugged application before the host allows the application to use the actual API. These requirements also add to the cost of using APIs, which can make them less popular than using libraries.

Considering Microservices

Microservices are a new technology that actually builds on existing technologies. A microservice is a sort of mix of web service, API, and library, but in a small package. In fact, microservices have the following list of features that you need to consider:

- Relies on a small, single purpose, service-based application to create a fully functional application (each single purpose application is a microservice)
- Uses the most appropriate programming language for the task
- Accesses application data using the most efficient data management technique for the particular microservice
- Develops lightweight communication between each microservice
- Depend on protocols such as REST to communicate, so that the pipe is dumb, but the microservice is smart
- Employs decentralized application management by monitoring each microservice separately
- Selects each microservice as needed to build any number of full-fledged applications (desktop, mobile browsers, native mobile apps, and even APIs)

Given the way microservices work, you need to consider some of the same issues that you do for both libraries and APIs. For example, a microservice could appear as part of your application code, so you need to consider just how the microservice interacts with the application. In addition, just like an API, you send and receive data when working with a microservice, so it's possible for man-in-the-middle attacks to cause problems.

From a security perspective, microservices tend to implement security differently than either libraries or APIs. Instead of offering a single solution for everyone and everything that uses the microservices on a site as a whole, each microservice offers individualized security based on that microservices' needs. As a result, microservices tend to provide better security, but the security is also uneven and harder to work with. Here are some examples of microservices (each of these sites provides access to a number of microservices—you'd choose which microservices you want to use in a particular application):

- Akana (http://www.akana.com/solutions/microservices)
- Archivematica (https://ww.archivematica.org/en/)
- Gilliam (http://gilliam.github.io/)
- LSQ.io (https://angel.co/lsq-io)
- Seneca (http://senecajs.org/)

> Because microservices are so new, you'll find a lot of discussion on just what constitutes a microservice. For some authorities, the number of Lines of Code (LOC) matters. A service that uses between 200 and 500 LOC is in the microservice range, but a service above that range isn't. (Fewer than 200 LOC apparently doesn't provide a sufficient level of functionality.) Consequently, a microservice such as Cloner (https://www.npmjs.com/package/app-cloner-heroku) does meet the requirements (305 LOC), but a microservice such as Deploy Hooks

(https://devcenter.heroku.com/articles/deploy-hooks) doesn’t (1,240 LOC).

II

Applying Successful Coding Practices

In this part of the book, you begin looking at the techniques for coding secure applications. Each of the chapters that follow discusses a particular part of a typical web-based application. Chapter 4 begins with the user interface, which is potentially the most important part because security begins and ends with user cooperation. If the interface is flawed, the user simply won't cooperate (at least, not nearly as well) and your security solution will contain flaws. Chapter 5 goes hand-in-hand with Chapter 4. An application that is unreliable is frustrating for a user to interact with and tends to create still more security issues both directly and indirectly.

Modern applications don't exist in a vacuum. If a developer were to build every application from scratch, everyone would suffer. The use of libraries (Chapter 6), Application Programming Interfaces (APIs) (Chapter 7), and microservices (Chapter 8) makes the process of building an application much faster and easier. In addition, because the code provided by these third party sources receives so much scrutiny, it tends to be safer than the standalone code you could build yourself and it usually receives updates faster than your organization could provide them for custom code.

4

Developing Successful Interfaces

Applications rely on interfaces to interact with the user. When the interface is flawed, the user's opinion of the application diminishes and user satisfaction suffers. Every security solution you can conceive of depends on the good will of the user to make it a reality. Yes, you can attempt Draconian measures to force user participation, but often these measures become one of the user constantly finding ways to overcome security, rather than work with it. In most cases, you receive the best results when the user is on your side, which means making an effort to create an interface the user can appreciate (in that it makes the application so easy to use that the application almost disappears from view). Of course, you do need to ensure the application enforces policies. Therefore, this chapter looks at both the carrot (user cooperating) and the stick (enforced policies) of application security from the interface perspective.

It's important to make the carrot part of the equation obvious. A user sees various kinds of eye candy, helpful tips, and a clear interface as signs that the developer really cares about the application and how the user views it. Making the user experience pleasant is essential.

The stick part of the equation is usually hidden and subtle. For example, instead of asking the user to type the name of a state, the application prevents unauthorized entries by creating a state list and having the user select from the list. Even though the user's choices are constrained, what the user sees is ease-of-use.

Some interface decisions are mistakes before you even code them. For example, some validation techniques tell the user that the input is unacceptable without telling the user why the input won't work. More importantly, when the feedback lacks useful tips and examples, the user becomes frustrated. This chapter also discusses techniques you can use to refine the interface. It isn't a matter of some techniques working, while others don't—it's more a matter of some techniques creating less user angst.

It's important to understand that few people would hand code a website any longer. Fewer still would code a website without using libraries, APIs, frameworks, microservices, and any of a number of other third party offerings. However, all of these third party sources rely on the

technologies described in this chapter. All that you're getting from a third party source is prepackaged code that you could have created yourself. Viewing the code in the way shown in this chapter helps you understand the underlying techniques used by third party sources so that you can better tell whether these sources are secure. With this in mind, the code in this chapter is designed to help you understand security principles, rather than something you'd actually employ on your website.

Assessing the User Interface

Most organizations need to spend time assessing the user interface of their applications because these interfaces create clear opportunities for hackers to gain access to your network. The "Specifying Web Application Threats" section of Chapter 1 describes a number of common exploits that hackers use. A surprising number of those exploits rely on some user interface element to help things along. The tighter you can make your interface without inhibiting user interaction, the better. The following sections describe some of the techniques you can use to assess the user interface.

The best way to work with the examples described in this chapter is to use the downloadable source, rather than type it in by hand. Using the downloadable source reduces potential errors. You can find the examples for this chapter in the \S4WD\Chapter04 folder of the downloadable source.

Creating a Clear Interface

An essential part of creating workable interfaces today is making them clear. The complex interfaces of old create security problems because the user isn't sure what to do or possibly has too much to do. The libraries available today make it possible to create clear interfaces that focus on one issue at a time in many situations. For example, the tabbed interface shown in Figure 4-1 provides an example of a clear interface that only does one thing at a time as part of the `Tabs.html` file.

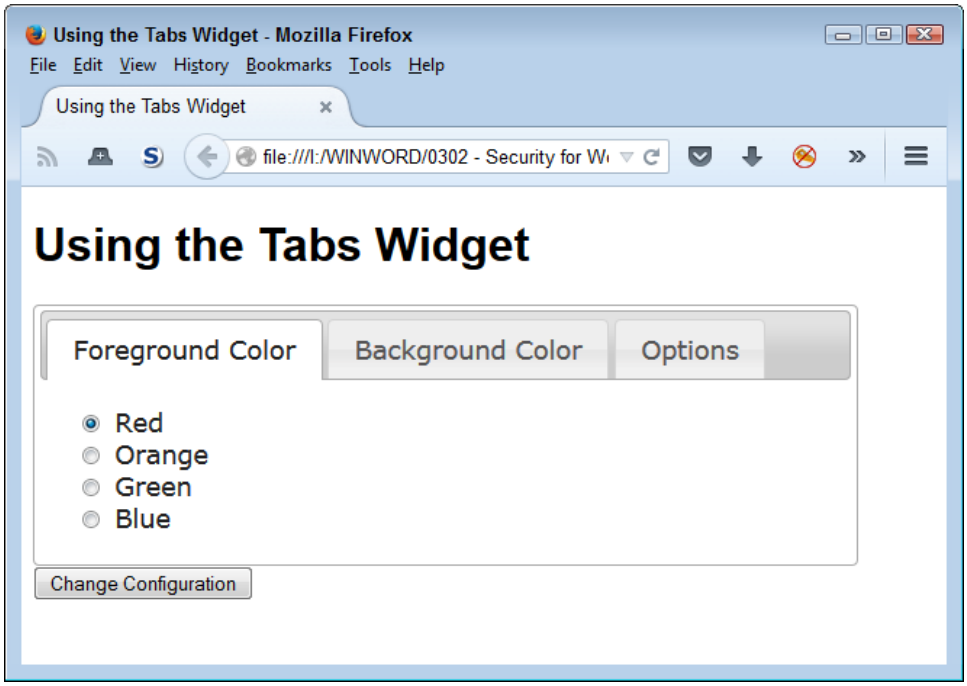


Figure 4-1. A clear interface focuses on one issue in a manner the user can understand.

This interface is quite clear. It asks a simple question, provides a limited number of responses, and literally makes it impossible for the user to enter bad data. The user selects a color, clicks the button, and moves on to the next tab. A number of web applications now make use of this sort of interface. You often see it used for signup sheets. The example relies on the jQuery UI library (http://jqueryui.com/) to perform its task. Here's the source code for this example:

```
<!DOCTYPE html>

<html>
<head>
   <script
      src="http://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="http://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <link
      rel="stylesheet"
      href="http://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
   <title>Using the Tabs Widget</title>
   <style>
      #Configuration
      {
         width: 90%;
         text-align: center;
      }
      #Configuration div
      {
```

```
          text-align: left;
      }
   </style>
   <script language="JavaScript">
      $(function()
        {
            $("#Configuration").tabs();
        });
   </script>
</head>

<body>
   <h1>Using the Tabs Widget</h1>
   <form id="ConfigForm" method="get" action="Tabs.html">
      <div id="Configuration">
         <ul>
            <li><a href="#Tab1">Foreground Color</a></li>
            <li><a href="#Tab2">Background Color</a></li>
            <li><a href="#Tab3">Options</a></li>
         </ul>
         <div id="Tab1">
            <input id="FGRed"
                   type="radio"
                   name="Foreground"
                   value="Red"
                   checked="checked" />
            <label for="FGRed">Red</label><br />
            <input id="FGOrange"
                   type="radio"
                   name="Foreground"
                   value="Orange" />
            <label for="FGOrange">Orange</label><br />
            <input id="FGGreen"
                   type="radio"
                   name="Foreground"
                   value="Green" />
            <label for="FGGreen">Green</label><br />
            <input id="FGBlue"
                   type="radio"
                   name="Foreground"
                   value="Blue" />
            <label for="FGBlue">Blue</label>
         </div>
         <div id="Tab2">
            <input id="BGRed"
                   type="radio"
                   name="Background"
                   value="Red"
                   checked="checked" />
            <label for="BGRed">Red</label><br />
            <input id="BGOrange"
                   type="radio"
                   name="Background"
                   value="Orange" />
            <label for="BGOrange">Orange</label><br />
            <input id="BGGreen"
                   type="radio"
```

```
                name="Background"
                value="Green" />
            <label for="BGGreen">Green</label><br />
            <input id="BGBlue"
                type="radio"
                name="Background"
                value="Blue" />
            <label for="BGBlue">Blue</label>
         </div>
         <div id="Tab3">
            <input id="Sounds"
                type="checkbox"
                name="Sounds"
                value="SpecialSounds" />
            <label for="Sounds">Use Special Sounds</label><br />
            <input id="Effects"
                type="checkbox"
                name="Effects"
                value="SpecialEffects" />
            <label for="Effects">Use Special Effects</label>
         </div>
      </div>
      <input id="ChangeConfig"
             type="submit"
             value="Change Configuration" />
   </form>
</body>
</html>
```

In order to create this interface, you import the jQuery (https://jquery.com/) and jQuery UI libraries, and the associated jQuery UI stylesheet. The tab information appears in a `<div>` with an `id` of `Configuration`. The magic used to create the interface is the result of making the `$("#Configuration").tabs()` call.

Making Interfaces Flexible

Reducing the user's ability to enter invalid data, making the interface clear and concise, and keeping things simple all restrict the user in various ways. They're forms of constraints that help you keep data safe while making the user's task of interacting with the application simpler. However, interfaces also require flexibility. The user will use the application on a number of devices, not all of which will accept the initial organization you provide. The act of letting the user organize the screen as needed provides the user with a sense of power without leaving any security holes open. Whenever possible, an interface should let the user:

- Drag: Moving items around on screen can provide the user with a better view of the interface and potentially reduce errant inputs. As a minimum, reorganizing the interface lets a user interact with the application in a manner that best suits the user.
- Resize: There are many reasons the user may need to resize an interface element. Perhaps the text is too small. Combining the resize action with the ability provided by most browsers of making the text smaller or larger as needed helps the user see the information better.
- Select: Creating positive feedback for selections is essential. You should assume nothing about the user's ability to see. The selection should provide multiple means

of feedback, such as using a thicker boarder, presenting the selection in a whiter (brighter) color, relying on changing textual attributes, and providing feedback in the form of color. This is one time where having someone on staff check the effects of selections using accessibility aids such as screen readers is important.

- Sort: The user should see the data in an order that makes sense to the user—not necessarily to you. With this in mind, provide multiple ways to sort data whenever possible so that the user can perform required analysis and make selections with greater ease.

The jQuery and jQuery UI libraries provide the means to perform all the essential interface flexibility tasks and a host of other tasks (such as allowing drag and drop data entry). Figure 4-2 shows an example of using these libraries to move items around on the screen.

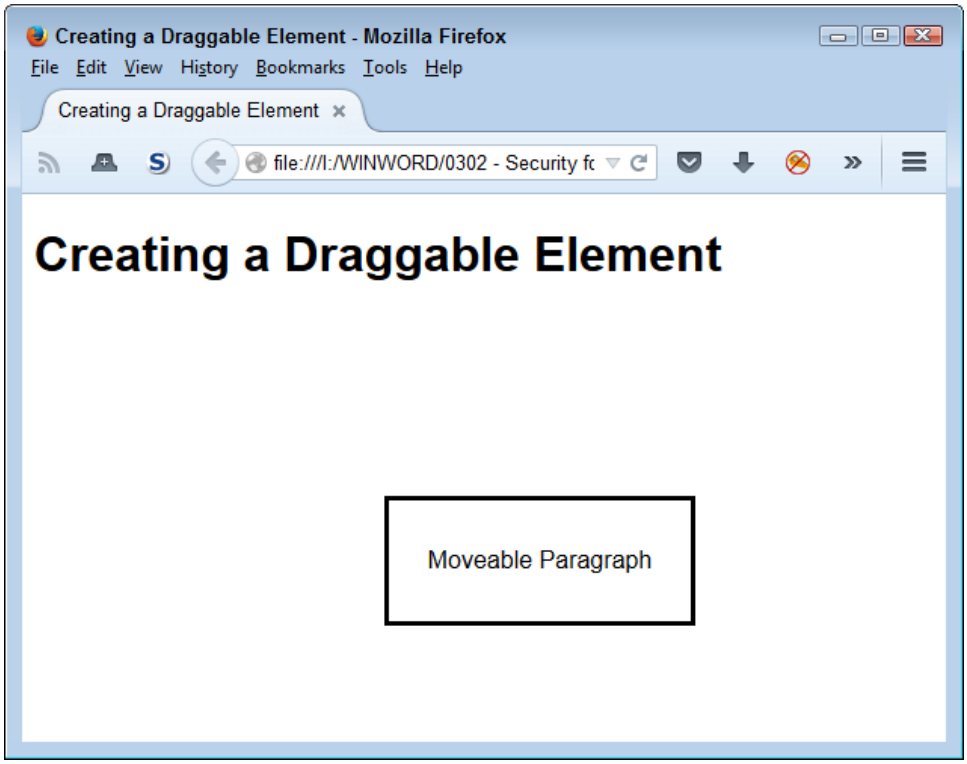


Figure 4-2. Make it possible to move items around.

A library can help you create an appropriate environment for user data entry. However, it can't remove the need to validate every piece of data you receive. Hackers like it when developers make assumptions about inputs based on the interface. Always assume that a hacker will find a way around the interface aids you provide to honest users and validate every piece of data no matter what source it comes from.

Creating an interface with moveable elements is relatively easy. The following code shows you how (you can also see this code in the `DragContent.html` file):

```
<!DOCTYPE html>
```

```
<html>
<head>
   <script
      src="http://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="http://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <link
      rel="stylesheet"
      href="http://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
   <style>
      #MoveMe
      {
         border: solid;
         width: 200px;
         height: 5em;
         text-align: center;
         line-height: 5em;
      }
   </style>
   <script language="JavaScript">
      $(function()
        {
            $("#MoveMe").draggable();
        });
   </script>
   <title>Creating a Draggable Element</title>
</head>

<body>
   <h1>Creating a Draggable Element</h1>
   <p id="MoveMe">
      Moveable Paragraph
   </p>
</body>
</html>
```

In this case, the `MoveMe` paragraph (`<p>` element) is the target. Notice that making the paragraph moveable doesn't affect the heading (`<h1>` element). All you need to do is call `$("#MoveMe").draggable()` when the form loads in order to make the paragraph element moveable.

Providing User Helps

Creating an environment in which even the least able user is able to function well is hard for most developers because most developers can't even imagine not being able to relate well to a computer. Many of the people a developer supports with an application are quite smart—they simply aren't smart about computers or the way in which applications work, so it's important to give them a hand with various types of helps.

The one help item you should always provide is explanatory text. You add explanatory text as part of the HTML tag associated with an element. Here are the attributes most commonly used for explanatory text.

- `title`: The `title` attribute makes it possible to add explanatory information to just about any textual or control element on the page. When a user hovers the mouse over the top of an element that has the `title` attribute defined, the browser displays the additional information in a text balloon. It's important not to make the text too long or to use anything but text within the `title` attribute because screen readers rely on these attributes to describe an element to someone with special visual needs.
- `alt`: Use the `alt` attribute to provide a short description of an image element. When the browser can't display the image, it displays the description instead. In addition, when someone with special needs can't see the picture, the accessibility aid they use will typically use the `alt` attribute text to introduce the image. The point is that the `alt` attribute provides a description of the image and also acts as a substitute for it as needed.
- `longdesc`: Many browsers ignore the `longdesc` attribute. This attribute provides a much longer description of an image than an `alt` attribute does. In most cases, you limit the description to about a paragraph of text. The point is to provide enough information so that someone with special needs can create a reasonable mental version of the image—at least the salient points of the image (rather than background).

User help information can appear in many other forms. A kind of help that most sites miss is to provide a sample of what you want as input for textboxes and other typed data as part of the input. You often see textboxes that contain a description of the desired content, rather than a sample of the desired content. (When the user types something in the textbox, the sample text disappears.) For example, instead of providing Last Name as the text within the textbox, provide an actual name, such as Doe (as in, John Doe or Jane Smith or something along those lines).

Any input field should also provide a help link. When the user clicks the link, the application displays a help page with additional information. The point is to make it as easy as possible for the user to provide correct information. Any mistakes you can prevent at the outset will reduce user frustration, make data entry more accurate, and most importantly, avoid the mistakes that often lead to security issues.

Defining the Accessibility Issues

Accessibility can be a tricky issue to deal with, especially when working with modern tools. However, it's an important issue to consider because according to Web Accessibility In Mind (WebAIM), about 20 percent of the people visiting your site have a special need of some sort: visual, hearing, motor skills, or cognitive. As the world's population continues to age, the percentage will grow larger. In fact, you have a high likelihood of eventually needing some sort of assistance for a special need.

You may wonder what accessibility has to do with security. When people don't understand an interface, they make mistakes. However, training usually helps with this problem because you can instruct people on the proper use of the software (and sometimes they actually listen). When people can't understand an interface because of a special need, they still make mistakes, but no amount of training will ever solve the problem. Even if the user is positively motivated to use the software correctly, unmet accessibility needs will prevent them from doing so. Mistakes often translate into security holes of the worst sort. The unexpected input—the one you never imaged anyone would provide, often creates the biggest problems. An application that lacks accessibility

features is more likely to create an environment where security busting mistakes occur on a regular basis.

Fortunately, many vendors design libraries, APIs, frameworks, and microservices with some level of accessibility in mind. They avoid uses of tables as a formatting aid, for example, which causes problems with the screen readers used by those who have visual needs. However, none of these tools is perfect and some of them are less perfect than others are. Consequently, it's important to test your site to determine just what level of accessibility it provides. Most of these testing tools also provide tips on how to fix the accessibility issues found on your site. Table 4-1 provides a listing of testing tools and a description of how you can use them.

Table 4-1. Accessibility Testing Sites

Site	URL	Description
Lynx	http://www.delorie.com/web/lynxview.html	This is a text-only browser that comes in versions for Linux, OS X, and Windows. It helps you check your site by displaying it in text only. This check helps you see your site as a screen reader will see it—making it possible to detect and correct problems with greater ease. Learn more about this product at http://lynx.browser.org/.
NIST Webmetrics Tool Suite	http://zing.ncsl.nist.gov/webmet/	A group of tools from the National Institute of Standards and Technology (NIST) helps you test the usability and accessibility of a site. For example, Web Static Analyzer Tool (WebSAT) ensures your page meets specific usability goals. The Web Variable Instrumenter Program (WebVIP) helps track user interaction so you know how well users are finding site features. There are more tools on this site and NIST updates them regularly.
Opera	http://www.opera.com/	Like Lynx, this browser enables you to see your site as a screen reader will see it. However, unlike Lynx, this product also helps you turn certain features on and off as needed for comparison. For example, you can toggle images off so you can see how <ALT> tags will look (see the instructions at http://help.opera.com/Windows/12.10/en/images.html). Opera is available for a number of platforms including Windows, OS X, and Linux.
O'Reilly XML.com	http://www.xml.com/pub/a/tools/ruwf/check.html	This site provides an XML syntax checker that also validates XHTML. You can provide the XML input directly or supply an URL containing the XML. The test run by this tool will check that the XML is well formed—it doesn't verify that the XML actually means anything. In most cases, you'll want to use the W3C HTML Validation Service for a final check of your Web page. However, this Web site does perform fast checks

Site	URL	Description
		of intermediate and test XML.
W3C HTML Validation Service	http://validator.w3.org/	This site checks the HTML on your Web page for conformance to World Wide Web Consortium (W3C) recommendations and standards. An error on this site means that the coding for your page is incorrect, even if most browsers will read it, so this tester goes beyond usability and accessibility requirements. Don't get the idea that passing the test on this site automatically makes your site accessible. Passing on this site means that your code is correct. However, making sure you code is correct is a good first step to ensuring you can add accessibility features.
Web Design Group HTML Validator	http://www.htmlhelp.com/tools/validator/	This site checks the HTML on your page or, as an alternative, on your computer. It also provides an option to validate a single page or the entire site. You may also find that this site is a little less picky than the W3C HTML Validation Service about which pages it will check, seems to output about the same information, but may not provide complete validation of your site.
WebAIM	http://webaim.org/	You can find all sorts of interesting tools for verifying that your site is accessible on this site. For example, you can use the WAVE Web Accessibility Evaluation Tool (http://wave.webaim.org/) to determine whether your site has any major accessibility problems. A serious problem for many sites is the use of color. The Color Contrast Checker (http://webaim.org/resources/contrastchecker/) helps you verify that people with color deficiencies can actually see your site.

Testing tools are helpful in verifying that your development team has done a great job in creating an application that everyone can use. However, the actual development effort will often require a checklist of tasks to perform and issues to check. The Section 508 checklist at http://webaim.org/standards/508/checklist helps you create such a list for your application development efforts. Interestingly enough, as hard as this checklist might be to read, the government documentation at http://www.section508.gov/ is a lot harder. The point of these requirements is to ensure that anyone, no matter what special needs they might have, can actually use your application as you intend. Without these features in place, you may find that up to 20 percent of your users have problems that they really shouldn't be having.

> If you have worked through accessibility issues in applications long enough, you know about the Bobby Approved icon that used to appear on some sites. Unfortunately, Bobby isn't available any longer—at least, not as a free service. You can read the history of Bobby at

http://www.bobby-approved.com/. The important thing to note is that accessibility is such an important issue and Bobby did such a terrific job at helping people add it to their sites that IBM eventually bought the product. Bobby is still around—it simply isn't free.

Providing Controlled Choices

Security is about exercising control. The application must maintain control over the manner in which it interacts with data. Managing data, without damaging it or causing a data leak, is the essential basis for creating a secure interface. One of the best ways to exercise control is to use techniques to ensure the user choices are limited to only the choices you expect. The use of specific entry controls limit user choices, but also make the user more efficient because the user has to think less about the available options. Here are the common specific entry control choices:

- Radio Buttons: Lets the user choose one choice out of a number of choices. Selecting a new choice always deselects the previous choice. The most common error developers make is not setting one of the choices as a default. When the data is sent to the server after the user fails to select an option, the server encounters a blank data entry and could potentially experience problems.
- Checkboxes: Lets the user choose anywhere between none and all of the available options by checking the desired options. The most common error developers make is allowing conflicting choices to occur. When working with checkboxes, each of the options is mutually exclusive.
- List Boxes: Depending on the configuration, a list box can act like a radio button or checkbox entry. An advantage of list boxes is that you can set them to hide the choices when the user hasn't selected the list box control. In addition to the issues that developers often experience with radio buttons and checkboxes, list boxes can suffer from issues in populating the list of choices. The user must have at least one choice available for the list box to act as a control. In accessible data sources can prove fatal for this control.
- Menus: Depending on the configuration, menus can allow complex combinations of radio button and checkbox behavior. Of course, menus are also used to make client-side application choices. Many developers fail with menus by making them too complicated. A user won't search long for a particular option, so making the menu too complex leads to errors when the user becomes frustrated.

In addition to standard input types, you can also use special HTML5 functionality in creating a controlled choice environment. However, in this case, the alternative is to provide the user with a text input box, which means that these options aren't as secure as you might think they are. Here are the HTML5-specific controls:

- color: Acts as a button with most supporting browsers. Clicking the button displays the platform's color picker. When the user chooses a new color, the color appears on the button so the user can see the current choice. Newer versions of Firefox, Chrome, and Opera all support this control.
- date: Displays a date picker that the user can use to select a specific date. The `min` and `max` attributes let you set the range of allowable dates. Newer versions of Chrome, Safari, and Opera all support this control.

- datetime: This input type isn't currently supported by any browser, so you shouldn't use it even though the specifications tell you that it's available.
- datetime-local: Provides a combination of the date and time controls so that a user can choose both a date and a time (with the restrictions of those controls in place). When specifying the `min` and `max` attributes, supply both date and time values as input. Newer versions of Chrome, Safari, and Opera all support this control.
- email: Displays a standard text input control. However, the control automatically validates the input to ensure the user supplies an e-mail address. Newer versions of Internet Explorer, Firefox, Chrome, and Opera all support this control.
- month: Displays a date picker that the user can use to select a specific month and year. The `min` and `max` attributes let you set the range of allowable dates (even though you're working with months, you must enter dates for these attributes). Newer versions of Chrome, Safari, and Opera all support this control.
- number: Permits a user to type any numeric value, but not an alphabetic or special symbol value. Setting the `min` and `max` attribute values controls the range of input. Newer versions of Internet Explorer, Firefox, Chrome, Safari, and Opera all support this control.
- range: Appears as a slider with most browsers. Setting the `min` and `max` attribute values controls the range of input. Use the `value` attribute to control the initial range setting. The `step` attribute controls the amount the slider moves for each change in position so that a user can only select specific values. Newer versions of Internet Explorer, Firefox, Chrome, Safari, and Opera all support this control.
- search: There isn't much point in using this HTML5 control because it behaves like a standard `<input>` tag. Newer versions of Chrome and Safari support this control.
- tel: Displays a standard text input control. However, the control automatically validates the input to ensure the user supplies a properly formatted telephone number. Newer versions of Safari support this control.
- time: Displays a time picker that the user can use to select a specific time (but not a time zone). The `min` and `max` attributes let you set the range of allowable times. Newer versions of Chrome, Safari, and Opera all support this control.
- url: Displays a standard text input control. However, the control automatically validates the input to ensure the user supplies a properly formatted URL. Newer versions of Internet Explorer, Firefox, Chrome, and Opera all support this control.

  The `url` input control is the only control that currently provides special support on smartphones (not found on other computer types). On some smartphones, the browser adds a special .com key when it senses the `url` input control.

- week: Displays a date picker that the user can use to select a specific week and year. The `min` and `max` attributes let you set the range of allowable dates (even though you're working with months, you must enter dates for these attributes). Newer versions of Chrome, Safari, and Opera all support this control.

To use this controls, you use the `<input>` tag. For example, to create an number input type you might use (see the `InputTag.html` file for details):

```
<input id="NumberInput" type="number" min=1 max=5 value=1 />
```

In this case, the user sees a text input control, but this one includes an up/down arrow control as shown in Figure 4-3. In addition, the `value` attribute provides a default value. The user can easily change the value by typing a new value or simply using the up/down arrow control.

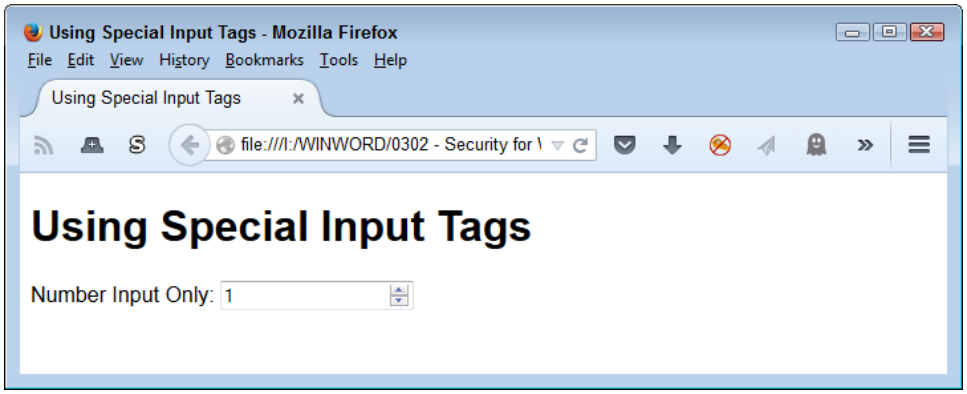


Figure 4-3. A number input features an up/down arrow control.

The `number` input doesn't prevent the user from typing an unwanted value. However, it does perform automatic validation. Depending on the browser, an incorrect entry receives some form of highlight. For example, in the case of Firefox, the border becomes heavier and the box appears in red as shown in Figure 4-4. The point is that HTML5 additions reduce the work you need to do for honest users, but it's possible for a determined hacker to overcome them.

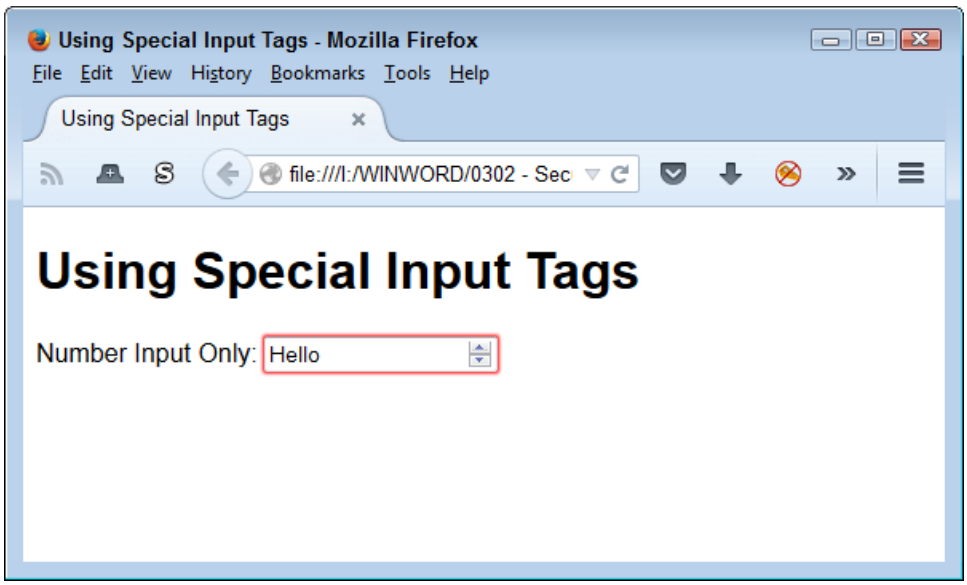


Figure 4-4. HTML5 controls provide automatic validation.

Some HTML5 controls are safer to use than others are. For example, the `range` input type normally appears as a slider, so the user doesn't actually enter a value. You should never depend on any of the controls providing absolute safety, however.

Choosing a User Interface Solution Level

When working with web-based applications, you have a choice of which level to use for implementing the user interface. In fact, most web-based applications rely on multiple levels of interface controls. Some part of the page appears using HTML tags, another part is a manipulation of the interface using Cascading Style Sheets (CSS), and a third part relies on JavaScript. It's important to remember that you can use JavaScript both at the client and at the server, so really, there are four levels of solutions that you can exercise as described in the following sections.

Implementing Standard HTML Controls

The "Providing Controlled Choices" section of this chapter tells you about the ways to control user choices. Many of those options appear as HTML controls. The advantage of HTML controls is that they work automatically for the most part. A browser does have to support the level of HTML that the control requires, but that's the only requirement.

Unlike most other control solutions, the browser doesn't have to support scripting in many cases to obtain an effect. Of course, if you want to perform detailed work, then you need some level of scripting support. As an example, if you want to use special controls to ensure a user fills out a form properly, you can actually perform that task and submit the form to the server for processing without using scripts. On the other hand, if you want to perform a client-side task, such as client-side validation, then you'll need scripting support in most cases (some HTML5 controls do provide rudimentary validation support).

There are a few disadvantages to using HTML controls. The most obvious is that HTML controls are rudimentary. If you want to add pizzazz to your site, then HTML controls will probably disappoint you. In addition, you can't go beyond the bare bones basics. It's impossible to create a good tabbed interface (like the one shown in Figure 4-1) using just HTML controls.

Working with CSS Controls

The essential purpose of CSS is to format page content in such a manner that it doesn't rely on a specific browser, device, or platform. In addition, the formatting doesn't affect accessibility needs because the user can normally replace the fancy CSS formatting with simpler formatting that works better with accessibility devices. That said, it's possible to use CSS for all sorts of other tasks through clever programming techniques. One of those tasks is creating controls of various sorts using a combination of HTML, CSS, and JavaScript, with an emphasis on CSS. In other words, you create the actual control using CSS, rather than relying on either HTML or JavaScript to perform the task.

It helps to have tools available when using CSS to create controls. One place to look for the require tools is Dynamic Drive (http://www.dynamicdrive.com/). When you visit the site, you see it provides access to all sorts of tools, including the focus of this particular example, Button Maker (http://tools.dynamicdrive.com/button/). Figure 4-5 shows how Button Maker appears when you first access it. You use the settings on this page to generate the CSS needed to create a micro-button. However, you could just as easily create other sorts of controls using other tools.

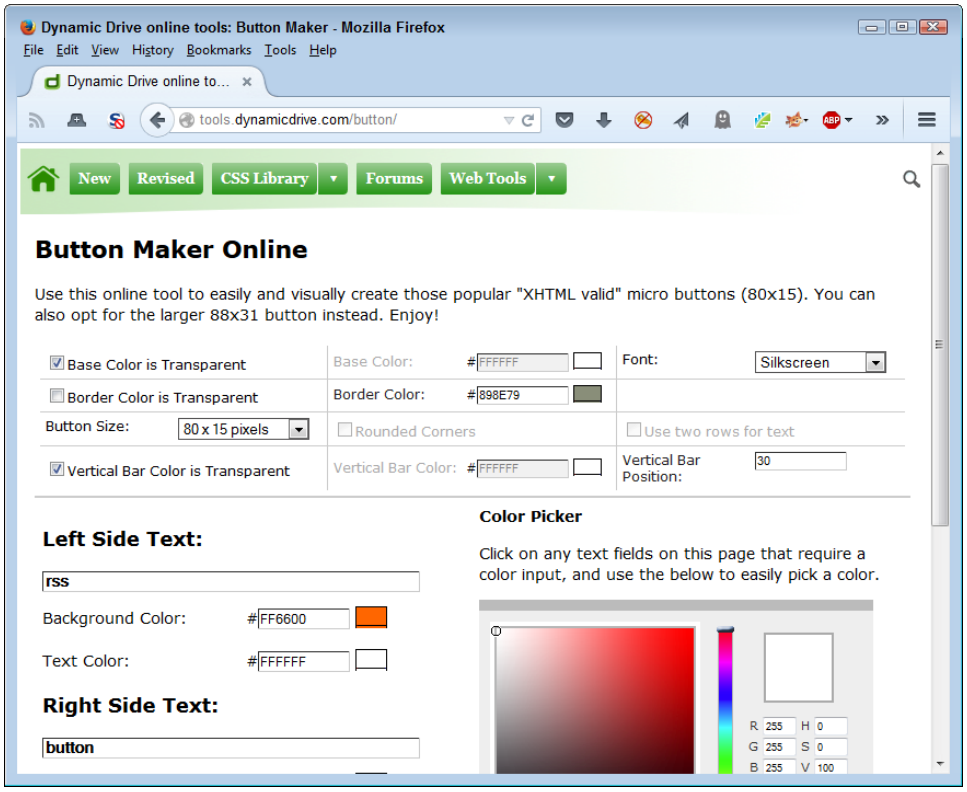


Figure 4-5. The Button Maker tool helps you create micro-buttons.

The output of the process, for this example, is a bit of CSS and a Graphic Interchange Format (GIF) file named `MyButton.gif`. The example code (found in `TestButton.html`) is relatively simple as shown here.

```
<!DOCTYPE html>

<html>
<head>
    <title>Testing a Micro Button</title>
    <style type="text/css">
        #MicroTest
        {
            border: none;
            background-color: transparent;
        }
    </style>
</head>

<body>
    <h1>Testing a Micro Button</h1>
    <button id="MicroTest"
            onclick="alert('Clicked!')">
        <img src="MyButton.GIF">
    </button>
</body>
</html>
```

Notice how the code uses HTML and JavaScript for support. The button itself is a GIF file. The CSS code performs the required formatting tasks. Of course, formatting would be more complex in a real application. The result of this experiment is a button like the one shown in Figure 4-6.

Figure 4-6. The micro-button is fully functional and nice to look at.

The main advantage of creating CSS controls is that everything is generated locally, so they tend to look nice, but the application also performs well. In addition, CSS lets you create applications with pizzazz, so they're a lot more fun to use. The use of CSS can also provide visual hints to users, which means that you can use them to reduce mistakes—enhancing the security of your application.

CSS also presents some limitations when it comes to security. In order to obtain any level of validation, you must either employ HTML5 or JavaScript techniques. CSS doesn't offer any sort of validation potential because it was never designed to perform this sort of support. Remember that the designers of CSS meant it to provide basic page formatting and nothing more. Only through clever programming do you get some of the interesting effects that CSS can provide.

Creating Controls Using JavaScript

When it comes to controls, JavaScript provides the options that most developers use to create complex applications. Yes, the developer also relies on HTML5 for placement support and CSS3 for formatting support, but the essential tasks of defining the interface rests with JavaScript. You have access to two levels of JavaScript support when creating an interface and may implement both levels in a single application. The following sections describe both client-side and server-side controls.

Relying on Client Controls

The chapter contains a number of examples of using JavaScript in various ways to create client-side control setups. Normally, you won't create the controls by hand, but will instead use a library setup such as jQuery and jQuery UI. When working with a client-side setup, the application loads the libraries from a local source or from a remote server,

but once the libraries are loaded, everything happens locally. Later chapters in the book discuss the consequences of using in-process library support (see Chapters 6, 7, and 8 especially). The main benefit of client-side processing is that you gain a speed advantage and the application is more reliable in many cases because you have constant access to everything required to make the application run.

JavaScript offers the best opportunities for client-side control validation. You can configure the validation to work precisely as you want it to. The validation can also provide more functionality than other sorts of validation do. However, because a hacker can see your code, it's possible that the hacker will come up with a strategy to thwart whatever you have in place.

A significant security issue with using client-side controls is that you are incorporating someone else's code into your application. This means that you have a trust relationship between your application and code that you may never actually see. Yes, you can look at the content of libraries such as jQuery an jQuery UI, but few developers have the time to do so and certainly not the patience to review essentially undocumented code. Client-side controls have potential risks associated with them that you need to consider as part of your security strategy for the application.

Relying on Server Controls

Server side controls often start with a PHP or other script on the server. In many cases, you can call the script directly, pass it the data it needs, and view just the script output in a browser. The processing all takes place on the server. Because the script is out-of-process, it's easier to keep your application separated from the third party code, making your application more secure. However, you have reliability and speed issues to content with when using server-side controls.

The client-side portion of this arrangement usually relies on JavaScript. In fact, you may embed the output from the server-side control in a page and update this content as needed using something like Asynchronous JavaScript and XML (AJAX). When using this setup, it's possible to speed the application by sending only the data to the server and only updating the portion of the page that the data affects.

Validating the Input

Of all the things you can do to ensure your application remains secure is to assume that all the input you receive is bad. When you assume that every input you receive contains a virus or some exploit that will cause damage to your system, you begin to view the input in a new way. Yes, it reeks of paranoia, but this viewpoint is essential to keeping your data secure. Validating data in a manner that reflects the corrupted input viewpoint is essential in today's application environment. Even with this viewpoint, you may find that you aren't paranoid enough and need to hire someone to exercise even more paranoia on your behalf. The following sections give you some ideas on how you can product your application and its associated data by validating absolutely ever input in every way that you can think of.

Allowing Specific Input Only

The problem you read about most often when it comes to data validation is that the application didn't control input well enough. When you think about it, that's the

underlying problem of many of the existing exploits. You should never allow a hacker to pass a script or other unwanted content through your application to the server. Of course, no one sets out to allow such content, but the content manages to get on the server anyway. The following list provides you with some ideas on the sorts of checks you should perform on any data passed from the application to the server (and then checked a second time on the server).

- Type: Always verify that the data type is correct. Strings are the hardest data type to check because they can contain anything by definition. Use stricter data types whenever possible. For example, when you need numeric input, use a number data type, rather than a string, to pass it.
- Range: Validate the range of any data type that supports it. Numbers are an obvious data type in this regard. However, you can check dates and times for range as well.
- Regular Expressions: Determine that the form of the data is correct. In some cases, it's possible to check strings using regular expressions that match the input string pattern to an anticipated pattern. For example, when you want a telephone number as input, make sure you actually get a telephone number.
- Special Characters: Users don't typically need to send any special characters to your application. A hacker employs special characters in a number of ways—none of them particularly helpful to your application. So, weeding out content that includes special characters is a must.

The best possible input from the user is the kind of input that you expect at the outset. For example, if you want the user to choose between red, yellow, and green, then the only acceptable answers are red, yellow, and green. Any other choice is invalid and you should reject it. The more specific you can make input, the easier it is to keep your application and its associated data secure.

Looking for Sneaky Inputs

Hackers constantly look for ways of providing unexpected input data. The point of doing so is to cause the application to crash in specific ways or to force the application to react in an unexpected manner. For example, sneaky input is the root cause of SQL injection attacks. The hacker makes use of inputs on a form to create a situation where the application ends up behaving in an unexpected manner. The result of such an attack can range from corrupted data to executing scripts that shouldn't even appear as part of the data.

> It may actually seem like a good idea at the time, but you should never allow the user to supply data used to create dynamic scripts. In fact, it's a good idea not to use dynamic scripts at all unless you can be sure of the source of the data used to create the dynamic script.

Requesting New Input

One of the favorite exploits that hackers employ is looking for situations where an application performs strict checks on input data during the first pass, but then relaxes those checks during subsequent passes. The situation is quite easy to create because of the way that many languages deal with loops. Only a careful developer will avoid the situation during second and subsequent input retries.

However, the issue takes on new meaning when you begin to question user motivations during data entry. For example, you need to consider just why a user would need ten retries to get their name right on a form. You can probably assume that the user does know their name—that it's not something they've suddenly forgotten. A constant need to resupply obvious data to the application may point to something other than a forgetful user. You need to question whether the form is clear enough. When you're sure the form is clear and that you have the proper validation in place, you need to start looking at other sources for the issue, such as a hacker trying various methods to break your application. In short, most applications should allow a certain number of retries and then shut down pending an administrator's attention.

> Never assume that users are fully trained. Some users require a lot of training time and never really get the complete idea of precisely what it is that they're supposed to do. Unless you want to spend your days looking for hackers lurking in places where hackers never go, you need to consider user training as a primary security requirement. A large number of form retries may point to a need for more training time, rather than a potential hacker issue.

Using Both Client-side and Server-side Validation

Application speed is a primary concern when creating a web-based application. A user's focus can wander in as little as one to five seconds—not a long time to perform much in the way of validation. However, it's essential to perform validation and obtain the correct kind of input from the user. With this in mind, using client-side validation seems like the path to travel. A JavaScript routine can quickly check for errant input and tell the user about it before the data makes the round trip to the server. The fact is that you really do want client-side validation.

However, client-side validation causes its own set of problems. The data must leave the host system at some point and travel to the server. Between the time the browser sends the data and the server receives that data, a hacker has all sorts of opportunities to intercept the data and change it. In order to ensure that the data really is of the right type and content, you need to perform a second validation check at the server. If no one has tampered with the data, the second validation will go quite quickly and the user will have a response within a reasonable timeframe.

Some people would consider the use of both client-side and server-side validation checks overkill. However, you really do need both. In fact, sometimes you get both whether you want them or not. For example, when using certain HTML5 controls, you get automatic validation. It's true that the validation isn't always the most precise or complete, but it does provide a certain level of safety for the user. Always remember that client-side validation is for the user's protection and you use it to keep the user happy. Anything you can do to reduce user frustration will also reduce application support costs and security concerns.

Server-side validation checks need to provide complete coverage of every input and output. It isn't enough to check data when it comes from the client and then assume that no one will tamper with the server. Hackers are motivated to attack the server because the server makes it possible to affect a lot of people with a single intrusion. Consequently, you do need to provide validation checks for every input to every routine on the server to ensure that the application and its data remain secure.

The issue you fight most often with validation checks is application speed. It's not just users that can experience problems when an application runs slow—server loading increases as well. Therefore, you must maintain a balance with validation checks. Security is often a matter of defining how much risk you're willing to accept in the pursuit of specific goals. It's important that management understand that you have to maintain a balance as part of the application design and development process.

Some developers also falsely assume that throwing additional hardware at a problem will solve it. Adding hardware reduces the reliability of the system as a while. A reduction in reliability is also a security risk. If a hacker has five servers to attack instead of just one, the probability of finding a server that lacks the proper patches or is loaded just enough to crash becomes greater. Increasing the amount of hardware used to serve an application is part of the solution, but you must balance it with increases in application speed without leaving major security holes in place.

Expecting the Unexpected

Users are amazing at times. They seem to find the one hole you didn't think to patch. A boring day might turn into a session of seeing just what it takes to break an application. In fact, the user might not even set out to break the application, perhaps it's all a matter of mindlessly doing things while talking on the phone. The point is that users will find problems with your application that you never expected to exist. These problems can lead to all sorts of issues in protecting the application and its associate data. Sometimes an issue can crash the application or perhaps even the server on which it's running. You may marvel at what users come up with, but in the real world, bored users spell unexpected events.

Hackers will also perform the unexpected with your application. However, in this case, the unexpected is more thoughtful and methodical. A hacker will continue probing your defenses looking for a way in until another target comes into view, the hacker gets bored, or the hacker finds the way in. In most cases, the hacker will find a way in when truly determined to do so. It's never a matter of if the hacker will gain access to your application, but the timing of when the hacker will succeed in the effort. The use of security measures slows hackers down and with proper monitoring, you can discover the hacker lurking about your system before it's possible to do any real damage.

Many of the unexpected things that users and hackers do are associated with inputs. For example, you might expect a string as input but receive a script instead. Users can click unexpected key combinations that result in a corrupted string that causes the application or server to crash. Sometimes it's simply a matter of assuming the user will do one thing, when quite another seems more appropriate. For example, a user might respond with a string when a number is required (or vice versa). Hackers, of course, are looking for all sorts of nefarious ways of providing input you didn't expect and will purposely provide the input you didn't expect.

5

Building Reliable Code

You might wonder why this book contains a chapter about building reliable code when the topic is security. An interrelation exists between application speed, reliability, and security. Each of these elements has a role to play in turning a good application into a great application. You can't emphasize one over the other without diminishing your application in some way. Of course, companies do make the conscious decision to emphasize one element over another, but usually to gain specific goals. For example, you may have a legal requirement to protect application at all costs, in which case you probably need to sacrifice some speed and reliability to achieve the goal. However, for most applications, the balance between the three elements is critical. Until you understand the interactions between speed, reliability, and security, it's nearly impossible to achieve an application with the maximum possible security in place and it is impossible to create an application that performs well in an environment where all three elements are prized.

This chapter views reliability as it relates to security. In other words, it examines how you balance reliability in order to achieve specific security goals in your organization. It also considers reliability as a statistical science; although the chapter won't bore you with the specifics of reliability calculations. The point is to understand the concept of calculated risk with regard to the state of security in an application. Anyone who tells you that an application doesn't create risks with regard to both data and organizational integrity doesn't truly understand either reliability or security.

As part of dealing with reliability issues, the chapter explores reliability from several perspectives. For example, in order to create a reliable application, you must define team protocols that ensure everyone understands the required concepts. Reliability experts also learn from each iteration of the calculations they make—likewise, you must incorporate a feedback loop to make changes in the way in which your organization views reliability in its unique environment. There are also the issues of using packaged solutions. You won't build many applications from scratch, so it's important to know the effect that using a packaged solution will have on your application.

> The main thought to take away from this chapter is that the most reliable application in the world is one in which the application

performs tasks without any sort of potential for disruption. This would exclude the use of any external processing, input, or resource because all three of these items have failure points. Such an application would have no users and would need to run on hardware with incredibly high reliability. However, even with all these factors in place, it's still impossible to create an application that is 100 percent reliable. Every application has the potential to fail, making it less than reliable.

Lack of Reliability Kills Businesses

The biggest headlines in the trade press are often about security failures. Hackers getting into supposedly secure health records or obtaining access to the social security numbers of thousands of credit card users tend to make for big news. It seems that you see reliability reported far less often. However, a lack of reliability can cause terrifying results—even business failures.

Consider the case of the Knight Capital Group (http://dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/). A glitch in the software it used to interact with the New York Stock Exchange caused it to buy nearly $7 billion dollars of stock it didn't want. Quickly reselling the stock caused the company a net $440 million dollar loss. However, the losses soon became greater. As people lost confidence in the Knight Capital Group, it began bleeding even more red ink. Eventually, another company, Getco, acquired the Knight Capital Group for pennies on the dollar (http://www.reuters.com/article/2012/12/19/us-knightcapital-getco-idUSBRE8BI0OF20121219)—all because of a software glitch.

Differentiating Reliability and Security

Some people equate reliability with security. However, reliability and security are two completely different measures of application performance. Yes, the two measures do interact, but in ways that many people really don't understand well. A reliable application isn't necessarily secure and vice versa. The following sections examine the issue of the interaction between reliability and security in greater detail.

The best way to work with the example described in this chapter is to use the downloadable source, rather than type it in by hand. Using the downloadable source reduces potential errors. You can find the source code examples for this chapter in the \S4WD\Chapter05 folder of the downloadable source.

Defining the Roles of Reliability and Security

Reliability is a measure of how often an application breaks. It's a statistical measure. You use reliability to answer the question of likely an application is to break given a certain set of circumstances. Reliability also tells you when the application environment is changing. For example, when you add staff and the load on the application increases, reliability may decrease unless you design the application to scale well. Even when the software is perfect, however, the hardware reaches a breaking point and reliability will still decrease. Therefore, reliability is a whole system measure. The hardware, user,

platform, operating environment, management techniques, and myriad other things all affect reliability and change what you see as application faults or failures.

Security is a measure of how much effort it takes to break an application. Unlike reliability, there is no method available to measure security statistically. What you have instead is a potential for damage that can only be quantified by the determination of the hacker who is attempting to cause the damage. If hackers are sincerely determined to break into your application can cause damage, they will almost certainly find the means to do so. When dealing with security, you must also consider the effects of monitoring and the ability of a team to react quickly to breaches.

In both cases, an application breaks when the stress applied to the application becomes greater than the application's ability to resist. A broken application fails in its primary responsibility to manage data correctly. Just how this failure occurs depends on the application and the manner in which it's broken. Reliability faults tend to cause data damage—security faults, on the other hand, tend to cause data breaches or result in compromised system integrity.

It's important to demonstrate the difference between simply being secure and also being reliable. The `RangeCheck1.html` example below shows code that is secure from the client side (you would also check the data on the server side).

```
<!DOCTYPE html>

<html>
<head>
    <title>Performing a Range Check</title>
    <script language="javascript">
        function testValue()
        {
            value = document.getElementById("Data").value;
            if (value == "")
            {
                alert("Please type a number!");
                return;
            }
            if ((value < 0) || (value > 5))
            {
                alert("Value must be between 0 and 5!");
            }
            else
            {
                alert("Value = " + value);
            }
        }
    </script>
</head>

<body>
    <h1>Performing a Range Check</h1>
    <input id="Data" type="number" value="0" min=0 max=5 /><br />
    <button id="Test" onclick="testValue()">
        Test
    </button>
</body>
</html>
```

In this case, a casual user who uses the up and down arrows on the `<input>` tag will never provide data outside the range (as shown in Figure 5-1). However, the JavaScript code found in `testValue()` also ensures that the data will never appear outside the range even when typed. Using a `number` input type means that if someone types a value such as Hello, what `testValue()` actually receives is an empty value. It's possible to test for this condition as well and provide the user with an appropriate response.

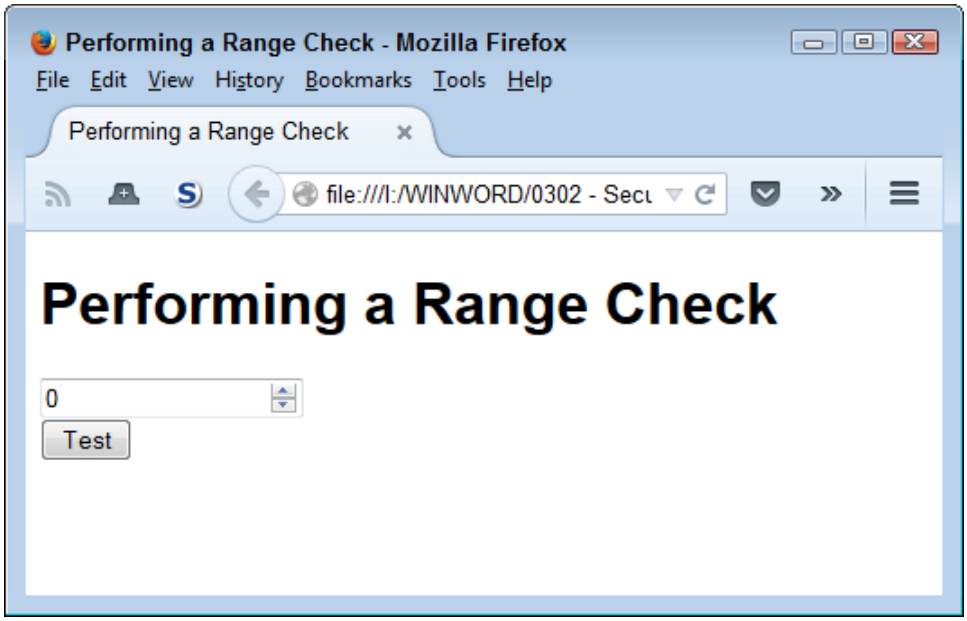


Figure 5-1. Range checks are made easier using the appropriate controls.

The problem with this code is that it's secure, but it's not reliable. If conditions change, then the code will no longer function as it should. The `RangeCheck2.html` example below makes the code more reliable by tying the range check to the `min` and `max` attributes of the `<input>` tag.

```
<!DOCTYPE html>

<html>
<head>
    <title>Performing a Range Check</title>
    <script language="javascript">
        function testValue()
        {
            inputObj = document.getElementById("Data");
            value = inputObj.value;
            if (value == "")
            {
                alert("Please type a number!");
                return;
            }
            if ((value < inputObj.getAttribute("min")) ||
                (value > inputObj.getAttribute("max")))
            {
```

```
                alert("Value must be between 0 and 5!");
            }
            else
            {
                alert("Value = " + value);
            }
        }
    </script>
</head>

<body>
    <h1>Performing a Range Check</h1>
    <input id="Data" type="number" value="0" min=0 max=5 /><br />
    <button id="Test" onclick="testValue()">
        Test
    </button>
</body>
</html>
```

The basic checks work as before. However, if someone chooses to change the `min` and `max` values accepted by the `<input>` tag, the code automatically responds by changing the conditions of the check. The failure points in this example are fewer.

However, to obtain code that is both secure and reliable, you must play a price in speed. Notice the number of additional lines of code in the second example and the increased number of function calls. You won't likely notice a difference in the speed of this particular example, but when you start adding these sorts of checks to an entire application, you can see some serious speed degradation. The code is more reliable and secure, but the user may not be happy with the result.

Avoiding Security Holes in Reliable Code

Just as secure software isn't automatically reliable, reliable software isn't automatically secure. In fact, you may find that the more reliable the software, the greater the risk that it truly isn't secure. The problem is one of the divergent goals of reliability and security. A reliable application is always accessible, usable, and predictable, which causes more than a few security issues. Here are cases in which reliability has divergent goals from security:

- A complex password keeps software secure, but it means that the software may not be available when the user forgets the password, making the software unreliable.
- Checks that ensure no one has tampered with application data can cause the software to become unavailable when the checks find tampering.
- False positives to security checks make the application's behavior unpredictable.
- Managing security settings increases application interface complexity and make it less usable.
- Denial of access to required resources due to security restrictions (such as, those created by a policy) makes the application less accessible and also makes it less predictable.

> The results of security holes in reliable code really can be terrifying. In a recent InfoWorld listing of ten extreme hacks (see

> http://www.infoworld.com/article/2933868/hacking/10-extreme-hacks-to-be-truly-paranoid-about.html), medical devices came in at number two. These devices are tested for five to ten years to ensure they continue working no matter what else might happen. However, no one patches the software during the testing period (and patching would entail additional testing). In addition, medical devices must prove easy to use, so anything that even resembles comprehensive security is left out of the development process. As a result, it's quite easy to kill someone by hacking their medical device. Of all the examples of reliable software with serious security issues, medical devices are at the top of the heap. They also have the honor of being the software with the most devastating results when hacked.

In fact, there are many situations where security and reliability butt heads. You must choose some sort of balance between the two in order to ensure that application data remains reasonably safe and the application still runs reliably.

Using the examples in the previous section as a starting point, it's possible to see how a range check would interfere with the user's ability to enter values outside the predicted range. Of course, the range check makes the application more secure. However, it's important to consider what happens when the person configuring the application's range check performs the task incorrectly and now the user is unable to enter a perfectly valid value. The application is still secure, but it becomes unreliable.

In some situations, a designer may view the potential ramifications of such a limitation as unwanted and make the application more reliable by excluding the range check. After all, if the purpose of the software is to prevent a nuclear reactor from going critical, yet the software prevents the entry of a value that will keep the reactor from going critical, then the security of the software is no longer important because no one will be around to debate the issue.

A middle ground fix for such a situation does exist, but it increases the complexity of the software and therefor affects reliability even more. In addition, because the fix requires added coding, application speed is also affected. However, by including the various security checks during normal operation and allowing an override by a manager or administrator to run the checks off during an emergency, the user can enter the correct value for saving the reactor, even though the software wouldn't normally allow it.

The point is that you can usually find a workaround for the security-only or reliability-only conundrum. It's usually a bad idea to focus on one or the other because the hackers (or the users) will make you pay at some point.

Focusing On Application Functionality

Making an application both secure and reliable fills a developer with joy, but the user won't care. User's always focus their attention on getting a task that the user cares about accomplished. The user's task might involve creating a report. (In reality, the report might not be the focus—the focus might involve getting money from investors—the report simply helps the user accomplish that goal.) If hand typing the report is easier and more transparent than using your application, the user will hand type the report. User's don't care what tool they use to accomplish a task, which is why you see users trying to create complex output using a smartphone. As a developer, you can secretly revel in the amazing strategies contained within your code, but the user won't care about it. The point

is that a user won't come to you and say that the application is unreliable. The user's input will always involve the user's task—whatever that task might be. It's up to you to determine that the issue involved in accomplishing the user's task successfully is that the application you created isn't reliable in some important way.

The balance between reliability and security becomes more pronounced when you focus on application functionality. It isn't simply a matter getting the task done, but getting the task done in the way that the user originally envisioned. In today's world, this means doing things like:

- Counting keystrokes—fewer is better
- Allowing the application to run on any platform
- Ensuring the application is always available
- Reducing the number of non-task-related steps to zero
- Making answers to questions obvious or avoiding the questions completely

Developing Team Protocols

Any effort made toward creating a reliable application has to consider the entire team. A development team needs to design and build the application with reliability in mind from the beginning. The team also needs to keep the matter of balance in mind during this process. It doesn't matter if an application is both reliable and secure if no one uses it because it runs slowly.

Team protocols can take in all sorts of issues. For example, in 1999 the Mars Climate Orbiter burned up on entry into the Martian atmosphere because one group working on the software used metric units and another group used English (imperial) units (http://www.wired.com/2010/11/1110mars-climate-observer-report/). The problem was a lack of communication—part of the protocol that you need to create for successful application development.

In order to create appropriate protocols for your organization, you need to break the tasks up in several ways. The team that creates an application must consider the issues of reliability from three separate levels:

- Accidental Design or Implementation Errors: When most people think about reliability problems, they think about glitches that cause the application to work in a manner other than the way in which the development team originally designed it to function. The application fails to perform tasks correctly. However, these errors could also be of the sort that opens the application to access by hackers or simply doesn't provide the required flexibility. Development teams overcome this problem through the use of developer training, use of secure development practices, and the employment of tools designed to location reliability issues of this sort.
- Changing Technology: An application becomes obsolete the day you finish working on it. In fact, sometimes the application is obsolete before you complete it. Technology changes act against software to make it unreliable. There are two levels of change you must consider:
    - Future Proofing: In order to create an environment in which an application can maintain its technical edge, you must future proof it. The best way to accomplish this goal is to create the application as components that interact, but

are also separate entities, to make it possible to upgrade one without necessarily upgrading the entire system. This is the reason that microservices have become so popular, but you can practice module coding strategies using monolithic designs as well.

    - Hacker Improvements: Hackers do innovate and become better at their jobs. A security or reliability problem that wasn't an issue when you started a project may become quite problematic before you complete the application. You may not even know the issue exists until a hacker points it out. The best way to handle this problem is to ensure you keep your tools and techniques updated to counter hacker improvements.

- Malicious Intent: There is a good chance that someone on your development team isn't happy or has possibly taken a job with your organization with the goal of finding ways to exploit software glitches. This team member may even introduce the glitches or backdoors with the notion of exploiting them after leaving the organization. Of course, you don't want to create a big brother atmosphere because doing so stifles innovation and tends to create more problems than it fixes, but you also need to ensure any management staff actually does manage the development team.

Communication between members of the team is essential, but ensuring that the communication isn't misunderstood is even more important. Assumptions create all sorts of problems and humans are especially good at filling in information gaps with assumptions. Of course, the assumptions of one team member may not be held by another member of the same team. Developing the sort of communication that teams require includes these best practices (you can find additional best practices, tools, and resources on the Cyber Security and Information Systems Information Analysis Center (CSIAC) site at https://sw.csiac.org/databases/url/key/2):

- Reliability and Security Training: Team members can't communicate unless they speak the same language and understand reliability concerns at the same level. The only way to accomplish this goal is to ensure team members receive proper training. When creating a training program, ensure that the training is consistent, whether you use external or in-house trainers.
- Reliability Requirements: Creating an application without first defining what you want is a little like building a house without a blueprint. Just as you wouldn't even start digging the basement for a house without a blueprint in hand, you can't start any sort of coding effort without first establishing the roles that reliability and security will play in the overall operation of the application. Make sure any definition you create includes specific metrics and goals for each development phase. The definition should also outline the use of both security and reliability reviews, code audits, and testing.
- Reliable Design: Just as you begin any project by identifying the security threats an application will face, you must also identify the reliability issues and specify methods for overcoming them. The design process must include specifics on how to deal with reliability issues. Although many organizations are aware of the need for security experts to help solve security issues, few are aware that a similar capability exists with reliability experts such as Reliability Consulting Services (http://www.reliasoft.com/consulting/).

- Reliable Coding: The coding process must keep both security and reliability in mind. It doesn't matter how much preparation you do unless you put what you've learned and designed into practice.
- Secure Source Code Handling: In order to handle threats such as malicious intent, your organization must practice secure source code handling. This means that only people with the proper training and credentials see the source code. In addition, it also means that you perform both design and code reviews to ensure the application remains faithful to the original design goals.
- Reliability Testing: It's essential to test the code to ensure it actually meets reliability goals that are part of the application requirements and design. Reliability testing can include all sorts of issues, such as how the application responds when a load is applied or when it loses access to a needed resource. The task is to test the failure points of the application and verify that the application handles each of them successfully.
- Reliability and Security Documentation: It's important to document the requirements, design, coding techniques, testing techniques, and other processes you have put into place to ensure the application is both reliable and secure. The documentation should express the balance issues that you found and explain how you handled them as part of the application requirements and design.
- Reliability and Security Readiness: Just before application release, the development team needs to ensure no new threats have appeared on the scene that the application must address to work successfully. Reliability and security both deal with risk and this phase determines the risk posed by new threats. It may work just as well to handle low risk threats as part of an update, rather than hold the application release.
- Reliability and Security Response: After application release, the development team needs to response to any new reliability and security threats in a timely manner. It's important to understand that sources outside the development team may report these issues and expect that the development team will provide a quick response.
- Integrity Checking: Ensuring the application continues to work as it should means securing the code using some type of signing technique (to ensure no one modifies it). In addition, the development team should continue testing the code against new threat and verify that the application manages data in a secure way. The idea is to keep looking for potential problems, even if you're certain that none exist. Hackers are hoping that your team will lack the diligence to detect new threats until it's too late.
- Security and Reliability Research: It's important to task individuals with the requirement to find new threats as they appear and to come up with methods for handling them. Testing the application is fine, but knowing how to test it against the latest threats is the only way to ensure your testing is actually doing what it should.

Creating a Lessons Learned Feedback Loop

Reliability is based on statistical analysis of events over time. The more time that elapses and the more events recorded, the more accurate the prediction. The average time between failure events is the Mean Time Between Failures (MTBF). Most software texts don't seem to pursue the topic from this perspective, but software, like anything else, has failure patters and it's possible to analyze those patterns to create a picture of when you

can expect the software to fail. Of course, like any statistic, it's not possible to pin down precise moments—only the general course of activity for a given application in a specific environment.

> Some people view MTBF as an incorrect measure of software reliability because they feel it literally indicates the next time that a software bug, environmental issue, or other factor will cause the software to fail. The important thing to remember is that MTBF is based on a specific environment. Because software rarely operates in precisely the same environment from organization to organization, trying to create an MTBF for an application that works in any organization won't work. An MTBF for an application for a specific organization does work because the environment for that organization is unlikely to change significantly. When it does, the MTBF value is no longer valid.

The higher the MTBF of an application, the less time spent supporting it and the lower the maintenance costs. In addition, a high MTBF also signals a situation where security breaches due to application failures are less likely. Application failures don't occur just because of bugs in the software—they also occur due to usage failures, environmental issues (such as a lack of memory), unreproducible causes (such as cosmic rays causing a spike in power), and other sources. Because it's not possible to manage all of these failure sources, software can never be 100 percent reliable. At some point, even the best software will experience a failure.

> Cosmic rays really do affect computers. In fact, the higher the altitude of the computer's storage, the greater the effect experienced. The size of the transistors in the chips also affects the incidence of soft errors. You can discover more about this interesting effect at http://www.nature.com/news/1998/980730/full/news980730-7.html, http://www.ncbi.nlm.nih.gov/pubmed/17820742, and http://www.newscientist.com/blog/technology/2008/03/do-we-need-cosmic-ray-alerts-for.html. The information is interesting and it may finally explain a few of those unreproducible errors you've seen in the past.

A feedback loop as to the cause of failures can help you increase MTBF within a given organization. By examining the causes of failure, you can create a plan to improve MTBF for the lowest possible cost. Here are some ideas to consider as part of analyzing the failure sources in software:

- Quality: As the quality of the software improves, the MTBF becomes higher. It's possible to improve the quality of software by finding and removing bugs, improving the user interface to make it less likely that a user will make a mistake, and adding checks to ensure needed resources are available before using them.
- Failure Points: Reducing the number of failure points within an application improves MTBF. The best way to reduce failure points is to make the application less complex by streamlining routines and making them more efficient. However, you can also do things such as increase redundancy when possible. For example, having two sources for the same data makes it less likely that the loss of a single source will cause an application failure.

- Training: Better training reduces operational errors and improves MTBF. There is a point of diminishing returns for training, but most users today don't receive nearly enough training on the software that they're expected to use to perform useful work. In addition, training support personnel to handle errors more accurately and developers to spot the true sources of failures will help improve the feedback process.

Creating complete lists of failures, along with failure causes, is the best way to begin understanding the dynamics of your application. As the knowledge base for an application grows, it's possible to see predictive patterns and use those patterns as a means of determining where to spend time and resources making corrections. Of course, it's not possible to fix some failure sources. Yes, you could possibly shield all your hardware to get rid of those pesky cosmic rays, but the chances of any organization expending the money is incredibly small (and the returns are likely smaller still).

Considering Issues of Packaged Solutions

As mentioned in earlier chapters, most applications today rely on packaged solutions to perform common tasks. Trying to create an application completely from scratch would be too time intensive and financially prohibitive. There really isn't a good reason to reinvent the wheel. However, using these packaged solutions will affect your application's reliability. You're relying on code written by someone else to make your application work, so that code is also part of the reliability calculation. The following sections discuss some issues you need to consider when working with packaged solutions.

Dealing with External Libraries

External libraries create a number of interesting reliability problems. The most important issue is that the application will generally download a copy of the library each time it begins to run. If you keep the application running, this process doesn't happen often, but most web-based applications run on the client system, which means that the client will need to download the library every time it starts the application. Speed becomes a problem because of the library download. However, the application might not even start if something prevents the client download from succeeding. For example, consider the use of jQuery UI to create an accordion effect like the one shown in Figure 5-2.

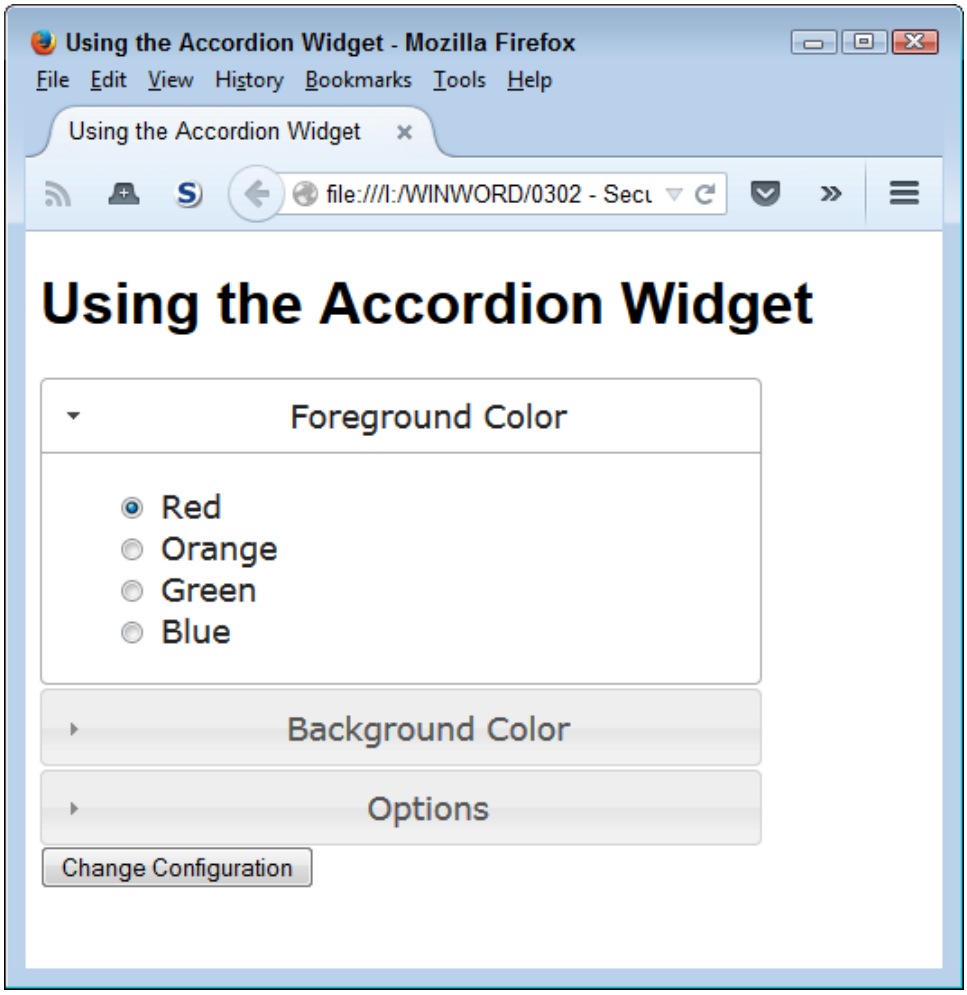


Figure 5-2. Even a simple jQuery UI example requires downloaded code.

This example won't even start should the library files it depends on become inaccessible for some reason. The following code (found in the `Accordian.html` file) appears in the header to create the required connectivity.

```
<head>
   <script
      src="http://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="http://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <link
      rel="stylesheet"
      href="http://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
```

If any of the three files shown in this example are missing, the application will fail to run. This example also shows two different approaches to selecting the libraries. Notice that the jQuery library relies on the latest version, which means that you automatically get

bug fixes and other updates. However, the jQuery UI library (and its associated CSS file) both rely on a specific version of the library. You won't get updates in this case, but may get bug fixes that could still cause your application to crash if you've already included workaround code for the problem.

One of the best ways to improve the reliability of external library use is to download the library to local drives and use that copy, rather than the copy on the vendor site. This strategy ensures that you have the same version of the library at all times and that there is less of a chance that the library won't be available for use. Of course, you'll need to supply local hard drive space to store the library, but the increased use of local resources is a small price to pay for the stability you gain.

As with everything else, there is a price for using a localized copy of the library. The most important of these issues is the lack of upgrades. Most vendors provide upgrades relatively often that include bug fixes, improvements in reliability, and speed enhancements. Of course, the library will usually contain new features as well. Balancing the new features are deprecated features that you may need to make your application run. In order to achieve the reliability gains, you give up some potential security improvements and other updates you may really want to use in your application.

A middle ground approach is to download the library locally, keep track of improvements in updates, and time your application updates to coincide with needed security and feature updates in the majority of the libraries you use. This would mean performing updates on your schedule instead of the vendor's schedule. Even though this alternative might seem like the perfect solution, it isn't. Hackers often rely on zero-day exploits to do the maximum harm to the greatest number of applications. Because your application won't automatically receive the required updates, you still face the possibility of a devastating zero-day attack.

Dealing with External APIs

External Application Programming Interfaces (APIs) provide the means to access data and other resources using an external source. An API is usually a bundle of classes that you instantiate as objects and use for making calls. The code doesn't execute on the client system. Rather, it executes as an Out-Of-Process (OOP) Remote Procedure Call (RPC) on the server. A client/server request/response cycle takes place with the client making requests of the server. Creating a link to the API is much like creating a link to a library, except that you must normally provide a key of some sort to obtain access as shown here for the `GoogleAPI.html` file (see Chapter 7 for details on this example).

```
<head>
   <script type="text/javascript"
      src="https://maps.googleapis.com/maps/api/js?key=Your Key Here&sensor=false">
```

The query requires that the client build a request and send it to the server. When working with the Google API, you must provide items such as the longitude and latitude of interface, along with the preferred map type. The API actually writes the data directly to the application location provided as part of the request as shown here.

```
// This function actually displays the map on
// screen.
function GetMap()
{
   // Create a list of arguments to send to Google.
   var MapOptions =
```

```
    {
       center: new google.maps.LatLng(
                  Latitude.spinner("value"),
                  Longitude.spinner("value")),
       zoom: 8,
       mapTypeId: google.maps.MapTypeId.ROADMAP
    }

    // Provide the location to place the map and the
    // map options to Google.
    var map = new google.maps.Map(
       document.getElementById("MapCanvas"),
       MapOptions);
};
```

The result of the call is a map displayed on the client canvas. Figure 5-3 shows a typical example of the output from this application.

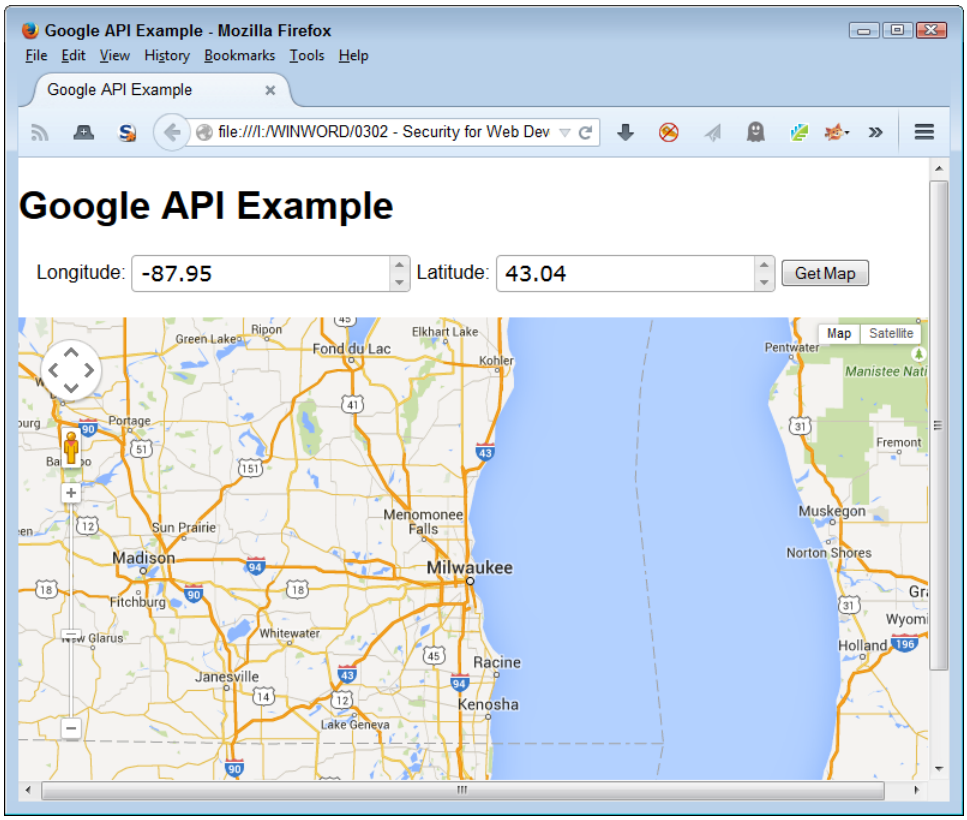


Figure 5-3. The Google API draws directly to the client canvas.

Using external APIs makes it possible to use someone else's code with fewer security concerns and potentially fewer speed issues (depending on whether it's faster to request the data or download the code). However, from a reliability perspective, external APIs present all sorts of risk. Any interruption of the connection means that the application stops working. Of course, it's relatively easy for a developer to create code that compensates for this issue and presents a message for the user. Less easy to fix is the issue of slow connections, data mangling, and other issues that could reflect the condition

of the connection or hackers at work. In this case, user frustration becomes a problem because the user could add to the chaos by providing errant input in order to receive some sort of application response.

A way around the reliability issue in this case is to time calls. When the application detects that the calls are taking longer than normal, it's possible to respond by telling the user about the slow connection, contacting an administrator, or reacting to the problem in some other way. The point is to ensure you track how long each call takes and act appropriately.

Working with Frameworks

Frameworks represent a middle ground of reliability between libraries and APIs. For the most part, frameworks are compressed code. One of the most commonly used frameworks is MooTools (http://mootools.net/). However, there are many other frameworks available out there and you need to find the right one for your needs. In most cases, frameworks can run on the server, the client, or a combination of both. Depending on how you use the framework, you can see the same reliability issues found with libraries, APIs, or a combination of the two.

Differentiating Between Frameworks and Libraries

Even though a framework like Dojo and a library like jQuery look quite a bit alike—and you use them in the same manner, for the most part—Dojo is a framework and jQuery is a library. From a security, reliability, and speed perspective, the two entities are different.

A library is pure code that downloads and runs as part of your application. You call functions directly and the source for those functions is sometimes available so that you can change the function behavior.

Frameworks provide a means of interacting with a behavior, which means that some tasks aren't visible to the developer—the developer requests that the framework perform the task, and the framework determines how to accomplish it. Code still downloads from the vendor and still becomes part of your application, but the underlying technology is different. Some people define a framework as a packaged form of library that provides structure as well as code.

The interesting part about products such as MooTools is that you can perform many of the same tasks that you do when using libraries. For example, you can create a version of the accordion example using MooTools. The following code is much simplified, but it gets the point across.

```
<!DOCTYPE html>

<html>
<head>
   <title>MooTools Accordion Demo</title>
   <script src="MooTools-More-1.5.1.js"></script>

   <style>
      .toggler
      {
```

```
        color: #222;
        margin: 0;
        padding: 2px 5px;
        background: #eee;

        border-bottom: 1px solid #ddd;

        border-right: 1px solid #ddd;

        border-top: 1px solid #f5f5f5;
        border-left: 1px solid #f5f5f5;

     }


  </style>

  <script type="text/javascript">
     window.addEvent('domready', function()
     {
        var accordion = new Fx.Accordion('h3.atStart', 'div.atStart',
           {
              opacity: false,

              onActive: function(toggler, element)
              {
                 toggler.setStyle('color', '#ff3300');
              },

              onBackground: function(toggler, element)
              {
                 toggler.setStyle('color', '#222');
              }
           }, $('accordion'));
     });
  </script>

<body>

  <h2>MooTools Accordion Demo</h2>
  <div id="accordion">

     <h3 class="toggler atStart">Section 1</h3>
     <div class="element atStart">
        <p>Section 1 Content</p>
     </div>

     <h3 class="toggler atStart">Section 2</h3>
     <div class="element atStart">
        <p>Section 2 Content</p>
     </div>

     <h3 class="toggler atStart">Section 3</h3>
     <div class="element atStart">
        <p>Section 3 Content</p>
     </div>
```

```
    </div>

</body>
</html>
```

You obtain the `MooTools-More-1.5.1.js` file from the Builder site at http://mootools.net/more/builder. Make sure you include the core libraries. Google does offer a hosted site at https://developers.google.com/speed/libraries/, however, this site doesn't include the additional features, such as `Fx.Accordion`.

The framework requires that you set up a series of headings and divisions to contain the content for the accordion as shown in the HTML portion of the example. How these items appear when selected and deselected depends on the CSS you setup. The script defines two events: `onActive` (when the item is selected) and `onBackground` (when the item is deselected). In this particular case, MooTools behaves very much like a library and you see the output shown in Figure 5-4.

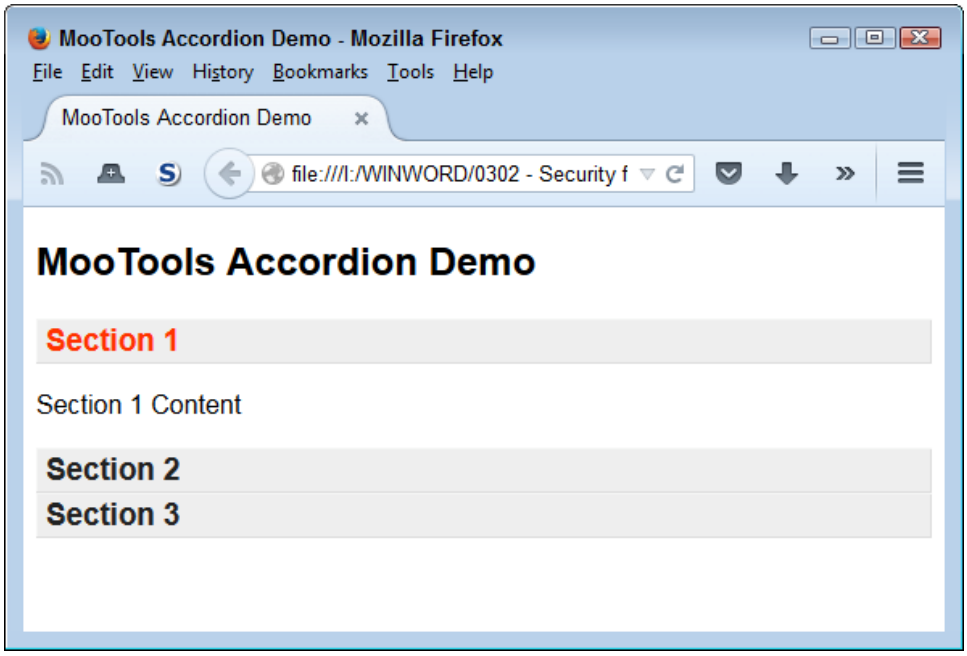


Figure 5-4. MooTools is a framework that provides both library and API functionality.

Calling Into Microservices

In many respects, a microservice is simply a finer grained API. The concept of using smaller bits of code so that it's possible to rely on whatever resources are needed to perform that one given task well makes sense. Because of the way in which microservices work, much of what you know about APIs from a reliability perspective also applies to microservices. For example, calls to a particular microservice may take longer than expected, fueling user frustration. The way around this problem is to provide application timeouts. However, in the case of a microservice, you can always try making the call using a secondary source when one is available.

However, microservices also differ from APIs in some respects simply because they are smaller and more customized. With this in mind, here are some reliability issues you need to consider when dealing with microservices:

- Using more microservices from different sources increases the potential for a lost connection and lowers application reliability.
- Choosing microservices that truly do optimize the service provided for the specific need you require will increase application reliability because there is less risk of major errors.
- Relying on a consistent interface for microservice calls reduces potential coding errors, enhancing application reliability.
- Having more than one source that can perform the same service reduces the number of single failure points, increasing application reliability.
- Keeping services small means that the loss of a single service doesn’t mean all services become unavailable as it would when using an API, so reliability is higher in this case as well.

6

Incorporating Libraries

It would be hard to find a web application that doesn't depend on a library—at least, not one of any consequence. Libraries are the epitome of not redefining the wheel. In fact, like wheels, there are libraries of every shape, size, and color out there. Yes, even color comes into play when you consider that many libraries include themes you can use to dress up your site. Consequently, it's likely that you've already spent a fair amount of time working with libraries in some way. In fact, a number of the book examples in earlier chapters relied on libraries simply because rewriting the code from scratch doesn't make sense.

This chapter doesn't try to persuade you about the value of using libraries. You've likely already sold yourself on their benefit. However, you might not have considered all the ramifications of using someone else's code in your application. There are always consequences to using libraries. The goal is to use libraries safely so that you get all the benefits of relying on someone else's code without creating any major security holes in your application. (Be assured, you do create some security holes, it's impossible to avoid it.)

Getting the Best Library Speed

You may find sites that tell you not to use libraries at all because they incur such use speed penalties (for example, see http://www.giftofspeed.com/dont-use-javascript-libraries/). The fact is that libraries can slow your site down, especially when you don't use them wisely. The speed hit can be large enough to make people perform the click that takes them to the next site, which means that your site loses out on providing a service or offering some other resource to the people who visit it.

This book generally uses the uncompressed versions of the available libraries to make it easier for you to work with the examples and perform tasks such as using a debugger to examine the applications with greater ease. However, from a speed perspective, you normally want to use the compressed version of the library. The compressed version contains the same code, but it loads faster and sometimes you get a bit of a boost when making calls as well.

It's also important to realize that you won't use all of the library code. Many libraries, such as jQuery UI (http://jqueryui.com/download/), provide a builder that lets you create a personalized version of the library that loads. After you discover which parts of a library you actually will use, use a builder to create a version with just those parts in it. The library will load faster and you also reduce the library's attack surface, which means an improvement in security as well.

Considering Library Uses

How you use a library directly affects how secure it remains—at least, to an extent. For example, direct involvement of the library with your JavaScript code tends to open more potential security holes than using the library to format the page. Of course, any use of a library could open potential security gaps that will cost your organization time, effort, and resources. The following sections discuss how you might use libraries to perform various tasks and consider how the form of usage could create certain types of security holes.

The best way to work with the examples described in this chapter is to use the downloadable source, rather than type it in by hand. Using the downloadable source reduces potential errors. You can find the source code examples for this chapter in the \S4WD\Chapter06 folder of the downloadable source.

Enhancing CSS with Libraries

CSS is all about formatting page content. Given the current state of CSS, it's amazing to see the sorts of pizzazz you can add to a site. Web developers use CSS to make a boring page pop out. Of course, there are mundane uses for CSS as well. For example, imagine trying to read multi-columned text without relying on CSS to format the content. What you'd end up with is a mess. Using just HTML tags to format the content would make the pages nearly impossible to use on the wide range of devices that people rely on today.

CSS alone could probably do everything you need to create fantastic pages. However, it's important to realize that even the best web developer doesn't have time to create all that CSS code by hand. There are a number of ways in which to pack CSS code into a library, but one of the more interesting is the `css()` function provided with jQuery UI as found in the `Transform.html` file.

```
<!DOCTYPE html>

<html>
<head>
   <script
      src="http://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="http://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <script
      src="jquery.transform.js">
   </script>
```

```
   <link
      rel="stylesheet"
      href="http://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
   <title>jquery.transform.js Demonstration</title>
   <style type="text/css">
      #TransformMe
      {
         border: double;
         border-color: Blue;
         border-width: thick;

         position: absolute;
         width: 120px;
         height: 40px;
         top: 90px;
         left: 75px;
      }

      #TransformMe p
      {
         margin: 0;
         padding: 0;
         height: 40px;
         font-family: "Comic Sans MS", cursive, sans-serif;
         font-size: large;
         color: Red;
         background-color: Yellow;
      }

      #Rotate
      {
         position: absolute;
         top: 190px;
         left: 75px;
      }
   </style>
   <script type="text/javascript">
      $(function()
      {
         $("#Rotate").click(function()
         {
            $("#TransformMe").css("transform", "rotate(45deg)");
         });
      })
   </script>
</head>

<body>
   <h1>jquery.transform.js Demonstration</h1>
   <div id="TransformMe">
      <p>Some Text</p>
   </div>
   <div>
      <input type="button"
             id="Rotate"
             value=" Rotate 45 Degrees " />
   </div>
```

```
</body>
</html>
```

The actual functionality found in this example relies on the `jquery.transform.js` script. However, before you get into the scripting, notice that the page does rely on standard CSS to place the content. In most cases, applications you create will rely on a combination of both local and library formatting sources. The content includes a paragraph formatted as a label and a button.

Clicking the button triggers the `click` event, which executes a single line of code. The CSS for `TransformMe` is modified to include a `transform` of `45deg`. The text goes from being straight across to having an angle like the one shown in Figure 6-1.

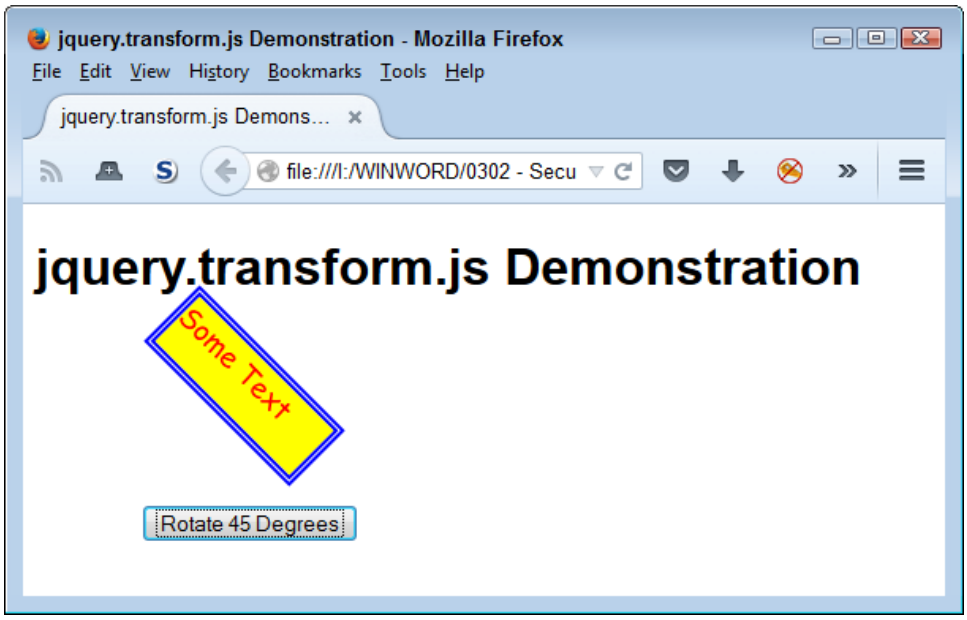


Figure 6-1. Using a transform can modify the appearance of text on screen.

The problem with allowing a script to affect page formatting is that you can't be sure how the page will look once the script completes. The page could contain hidden elements or simply not present information as expected. Hidden elements could contain code to redirect the user to an infected site or perform other deeds that you don't want associated with your application.

Interacting with HTML using Libraries

Some library functions interact directly with the HTML of your application in two common ways. The first is to interact with an existing element. In this case, content appears within the element based on the library function called used. The second is to create new elements and attach them to the existing document. The new elements can contain any sort of content, including tags that could potentially cause problems for the end user.

A major concern when using a library to interact directly with the HTML on your site is that the library could fill various elements with misleading, incorrect, or outright

contaminated data. For example, instead of creating the link you expected, it might actually redirect a user to another site—perhaps one that downloads a virus or other nasty piece of software to the user's machine. Libraries that work with HTML can cause subtle issues that you might not even know about immediately because everything will appear to work correctly. Unless a user complains about the misdirection (and given how URLs today they probably won't), you won't know about the issue until enough machines are infected to cause severe problems.

The adding of new elements to a page is a common practice. The example found in `ViewJSON.html` adds new elements to the existing document based on the content of a JavaScript Object Notation (JSON) file named `Test.json`. The following code shows how the example performs this task.

```
<!DOCTYPE html>

<html>
<head>
   <title>Viewing JSON Files</title>
   <script
      src="http://code.jquery.com/jquery-latest.js">
   </script>
   <script language="JavaScript">
      function ViewData()
      {
         // Obtain the data from disk.
         $.getJSON("Test.json",
            function(data)
            {
               // Create an array to hold the data.
               var items = [];

               // Parse the data by looking at
               // each entry in the Users object.
               $.each(data.Users,
                  function(key, value)
                  {
                     items.push("<li>" +
                        value.Name + "<br />" +
                        value.Number + "<br />" +
                        (new Date(
                           parseInt(value.Birthday.substr(6)))
                        ).toDateString()
                        + "</li>");
                  });

               // Place the result in an unordered list.
               $('<ul/>', {html: items.join("")}).
                  appendTo('body');
            });
      }
   </script>
</head>

<body>
   <h1>Viewing JSON Files</h1>
   <input id="btnView"
          type="button"
```

```
          value="View JSON Data"
          onclick="ViewData()" />
</body>
</html>
```

In this case, the library opens the `Test.json` file and reads data from it. The code removes the JSON file formatting and adds HTML tag formatting. It then uses the `join()` function to add the new elements to the page. Figure 6-2 shows the output from this example.

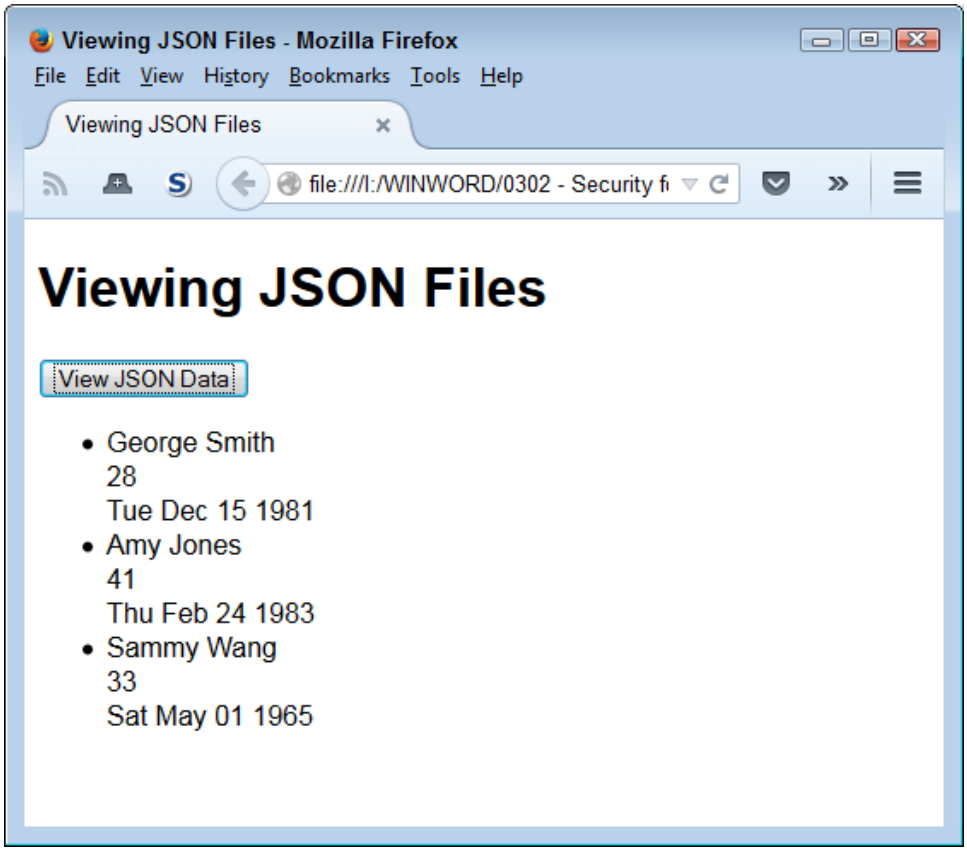


Figure 6-2. Injecting data from an external source is always problematic.

Notice that the example avoids the use of the `document.write()` call, which could expose users to all sorts of unwanted content. Using the `join()` function instead provides a safer alternative that should keep the page free of security issues.

Extending JavaScript with Libraries

Most people think about the ways in which libraries extend either CSS or HTML. The presentation of information is at the forefront. However, libraries can affect the underlying scripting as well. For example, most people are familiar with how to use `alert()` to display simple messages. Most libraries contain alternatives that a hacker could use to display other sorts of information on screen or affect the application in other

ways. For example, the `DialogBox.html` file contains an example of creating a custom dialog box like the one shown here.

```
<!DOCTYPE html>

<html>
<head>
    <title>Creating a Simple Dialog Box</title>
   <script
      src="http://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="http://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <link
      rel="stylesheet"
      href="http://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
   <style type="text/css">
    .Normal
    {
        font-family: Arial, Helvetica, sans-serif;
        color: SaddleBrown;
        background-color: Bisque;
    }
    .Emphasize
    {
        color: Maroon;
        font-style: italic;
        font-size: larger;
    }
   </style>
</head>

<body>
    <h1>Creating a Simple Dialog Box</h1>
    <div id="DialogContent"
         title="Simple Dialog Example"
         hidden>
        <p class="Normal">
            This is some
            <span class="Emphasize">interesting</span>
            text for the dialog box!
        </p>
    </div>
    <script type="text/javascript">
        $("#DialogContent").dialog();
    </script>
</body>
</html>
```

In this case, the example displays the dialog box immediately when you open the application as shown in Figure 6-3. The threat offered by such functionality is that a user could open your application expecting one experience and getting something completely different. Because the dialog box appears as part of the application, the user will provide any sort of information the dialog box requests or do anything it wants the user to do.

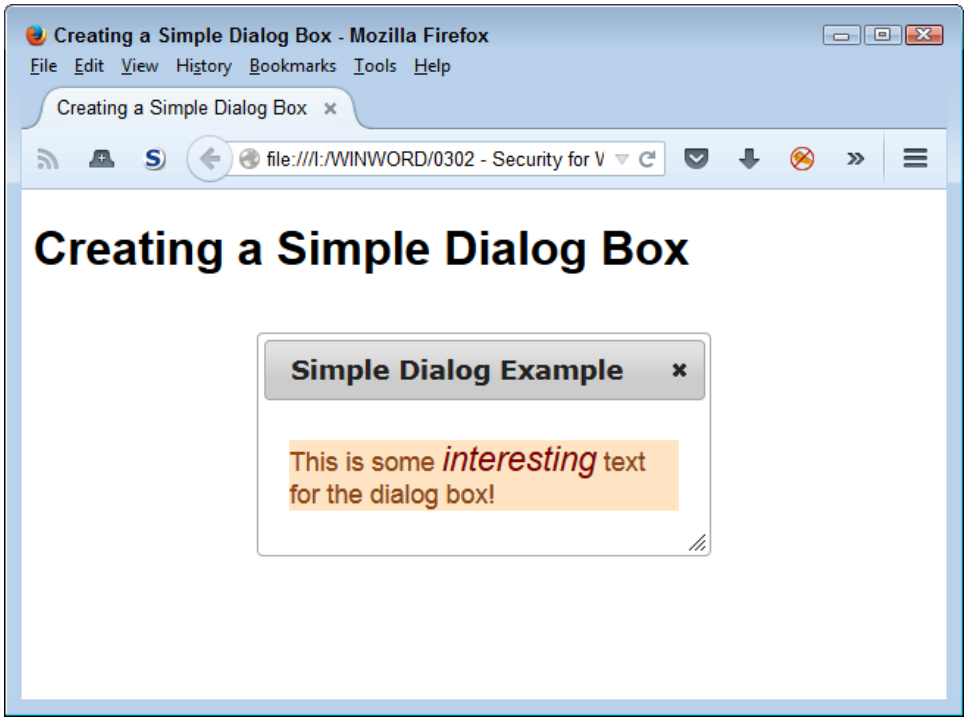


Figure 6-3. The dialog box functionality of jQuery UI could be used to create a security hole.

A hacker could enable such functionality by using something like script (code) injection. For example, the hacker could hide the code as a comment on a blog page (which is why manual moderation of comments is such a good idea). The underlying attack is one of social engineering—putting the user at ease by presenting the dialog box as part of the application and then using that situation to do something nasty.

Differentiating Between Internally Stored and Externally Stored Libraries

You may get the idea that all JavaScript libraries appear on a third party site. It's true that many popular JavaScript libraries, such as jQuery, do appear on someone else's site. However, many organizations have their own personal libraries as well. The factors that affect internally stored libraries are:

- They reside on systems owned and maintained by the organization
- The organization controls the source code and can modify it as needed to address issues that developers encounter or are brought to the organization's attention by others.
- Outside parties can't access the library without the organization's permission.
- The connectivity to the library relies on internal lines, rather than access through the Internet.

Internally stored libraries have many advantages not found in those stored externally on a third party site. For the most part, given the same level of coding, an internally stored library is faster, more reliable, and more secure than an externally stored library. In addition, because the library contains only the code an organization actually needs, it's quite possible that the internally stored library will use fewer resources because it will be smaller than its generic counterpart will. Of course, these statements make a lot of assumptions about the internally stored library that might not be true. Here are the counterpoints you need to consider:

- Internally stored libraries are expensive to build and maintain, so the organization may not keep them updated as required to keep them secure.
- Few organizations can field a development team equivalent in skills to a third party vendor, so the quality of the library will likely suffer.
- Third party libraries receive testing from a substantial number of testers, so that even small errors come out. Internally stored libraries generally receive a modicum of testing that may not even weed out some major flaws.

The bottom line is that internally stored libraries are successful when they target special functionality not provided by third party counterparts and the team putting them together takes the care required to do the job well. When putting your application design together, you must weigh the benefits that internally stored libraries provide against the risks they present. In addition, you must further define the risks presented by third party libraries.

Defining the Security Threats Posed by Libraries

Any JavaScript code you create is likely to contain some error that a hacker can exploit for various purposes. Libraries aren't special in this regard. However, because you're incorporating code that your organization didn't create into an application your organization did create, there is a chance that code that would normally behave correctly will end up having a security flaw due to assumptions on both sides. With this in mind, the two tools that every developer needs to know about when it comes to libraries are testing tools specifically designed for security needs and a good security expert.

> It isn't ever possible to say that the library you maintain is completely secure. You can say that you've written the code to meet security best practices and that a security expert has reviewed the code, but even with the best intentions and robust tools, it simply isn't possible to guarantee that a library (or any other code for that matter) is secure. This issue comes up relatively often when speaking with non-programmers—especially management staff of an organization that wants to use your library for development purposes. It's important that everyone understand that you have taken every appropriate security measure, but that no one can guarantee any code is completely secure.

Even after testing, you have to realize that there is a potential for security flaws in a library, the application it supports, or as a result of combining the two. With this in mind, here is a list of the most common sorts of threats that you could encounter when working with libraries (with no emphasis on the actual source of the flaw).

- Cross-Site Scripting (XSS): The most common security problem that developers face is XSS. There are three easy ways to get past XSS:
  - Never transmit untrusted data in the same HTTP response as HTML or JavaScript. In fact, it's best if the main HTML document remains static.

    You might wonder how your server would even end up sending untrusted data to anyone. It's easier than you think. The paper at http://www.cs.berkeley.edu/~prateeks/papers/scriptgard-ccs11.pdf describes just how hard it is to ensure data integrity even if you use the right sanitizers. The safe assumption is that any data you didn't personally create is untrusted.

  - When you must transmit untrusted data from the server to the client, make sure you encode it in JSON and that the data has a `Content-Type` of `application/json`.
  - Once the data is ready for display, use the `Node.textContent()`, `document.createTextNode()`, or `Element.setAttribute()` (second parameter only) calls to ensure the page presents it properly.
- Dangerous Function Calls: Just because JavaScript supports a particular call, doesn't mean the call is safe. Using `setInnerHtml()` or `.innerHtml =` can inject unwanted script code. Rely on the `setInnerText()` call instead.
- Modifying the Document Directly: Even though you may see the `document.write()` appear a few times in the book for the sake of expediency, using this call in a production environment is an invitation to disaster. The code could write anything, anywhere. Use calls that add, remove, or update Document Object Model (DOM) elements instead.
- Creating Scripts on the Fly: Any time you turn a string into a script, you're inviting a hacker to provide the string. Using calls such as `eval()`, `setTimeout()` with a string argument, `setInterval()` with a string argument, or `new Function()` makes your code significantly less secure.
- Code that Executes, but Causes Security Flaws: Some JavaScript calls give you all sorts of rope to hang yourself with. In order to avoid this situation, use JavaScript strict mode to ensure that only safe calls will actually work.
- Content that Purports to be Something It Isn't: Hackers love to send you content that isn't quite what you think it is. The best way to avoid this problem is to follow a Content Security Policy, which means including the appropriate content tags, such as `script-src 'self'` and `object-src 'self'`.

  Scanning a library using a product such as JSLint (http://www.jslint.com/) can help you ensure the quality of your code. High quality code is less likely to contain errors that will cause security issues. However, it's important to realize that JSLint (and tools like it) don't scan specifically for security issues. In fact, you can't scan for security issues in your code. In order to check for security issues, you must begin by testing your code specifically for security issues. If the code requires a higher level of confidence than testing will provide,

then you must also employ the services of a security expert to check to the code for potential problems manually.

Enabling Strict Mode

Newer browsers provide support for ECMAScript 5, which includes the JavaScript strict mode. In order to use this security feature, you need to ensure that your users aren't hanging on to that fossil of a browser that's been on their machine from the first time they used it. For example, IE 9 doesn't support this feature, but IE 10 does. You can find which browsers support strict mode at http://kangax.github.io/compat-table/es5/. It's essential that you develop an organizational policy that requires users have certain browsers in order to work with the application.

Strict mode makes many obvious and subtle changes to the way JavaScript operates. The important thing is that it makes it easier to debug JavaScript applications because instead of silently failing or acting oddly, your JavaScript application will now raise errors when it encounters a problem. This means that those odd library calls that used to die without doing anything, will now tell you something about the problems. Here are the major problems that strict mode helps you deal with.

- Eliminates the Use of `with`: Using the `with` statement can open security problems in your application. The latest versions of JavaScript have deprecated this feature because it's so dangerous. Strict mode will raise an error whenever you try to use `with` in your code.
- Prevents Unwanted Variables: A major problem in JavaScript is that you can accidentally create new variables by assigning a value to a variable with a mistyped name. In some cases, this error can create unwanted global variables that can result in security holes. Strict mode forces you to declare every variable before you use it, so it's not possible to create variables accidentally any longer.
- Disallows Coercion Using `this`: Some existing code coerces local variables to the global state by assigning a value to them when they're in the unassigned or null state. For example, you can't assign a value to a variable within a constructor without first instantiating an object by calling `new`.
- Precludes Duplicates: It's quite easy to create duplicates of properties in objects or named arguments in functions. Strict mode throws an error if you try to create a duplication in either situation.
- Notifies of Immutable Value Change Attempts: It isn't possible to change an immutable value in JavaScript. However, in the past, the attempt would fail silently, so it was possible to assume the code was in one state when it was really in another. Strict mode throws an error after any attempt to change an immutable value.

Strict mode comes with ECMAScript 5 and above. You don't have to install anything special to get strict mode support. However, you do have to include the `"use strict";` string. Older versions of JavaScript ignore the string, so you won't see any changes in how older code runs. The `StrictMode.html` file contains the following example.

```
<!DOCTYPE html>

<html>
<head>
```

```
    <title>Working with Strict Mode</title>
    <script language="javascript">
        function testStrict(Name)
        {
            "use strict";

            try
            {
                this.Name = Name;
                alert(this.Name);
            }
            catch (e)
            {
                alert(e);
            }
        }
    </script>
</head>

<body>
    <h1>Working with Strict Mode</h1>
    <button id="Test" onclick="testStrict('George')">
        Test
    </button>
</body>
</html>
```

Without strict mode checking (you can simply comment the statement out to try it) the code displays the value erroneously assigned to `this.Name` of George. However, with strict mode checking in place, you see the error message shown in Figure 6-4.

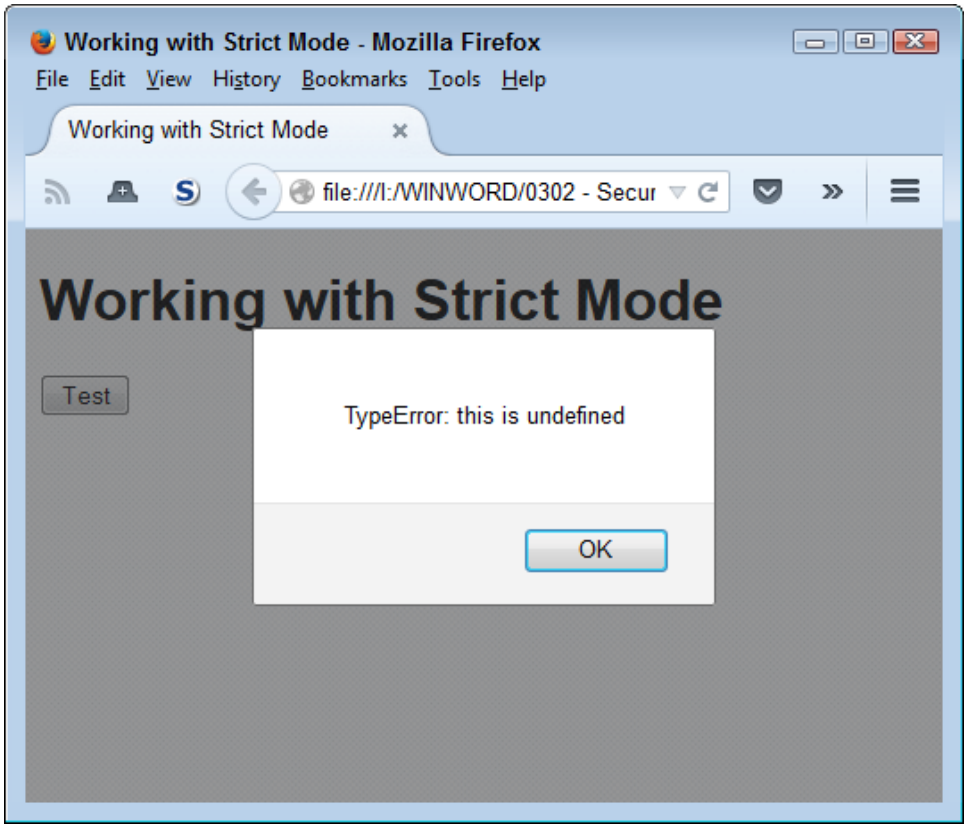


Figure 6-4. Strict mode checking prevents errant assignments.

> Never use strict mode globally. It might seem like a good idea at first, but you'll quickly find that some library code will fail. The idea behind strict mode is to ensure you have firm control over the code you create. Using validation checks, you can monitor the behavior of libraries that you use.

Developing a Content Security Policy (CSP)

One of the biggest problems that developers face is creating applications that aren't susceptible to various kinds of script and content injection attacks such as XSS. The problem defies easy resolution because of the way in which browsers work. Browsers effectively trust any content coming from a specific origin, so that the browser trusts anything injected at the same level as the original data. CSP seeks to remedy this problem by creating a whitelisting facility of a sort.

For browsers that support CSP (see the table at http://caniuse.com/contentsecuritypolicy for a listing of compatible browsers), CSP provides the means for keeping browsers from recognizing scripts and contents from unsupported sites. The policy appears as a series of headings that you add to the top of the page. When a browser sees the headings, it uses them to determine which scripts and content is safe to load.

When a browser doesn't provide support for CSP, it ignores the headings you provide and acts as it normally would. This means that users must rely on browsers that support CSP if you want to use CSP as a means of blocking unwanted content. Otherwise, the browser will work as it always has and continue to load content from unsupported sites.

A header consists of three basic elements: policy name, data type, and data source. Here is an example of a CSP heading that contains just one data type:

```
Content-Security-Policy: script-src 'self'
```

In this case, the CSP states that the browser should only execute scripts found embedded within the page. Let's say that you want to provide support for jQuery as well. In this case, you'd extend the policy as shown here:

```
Content-Security-Policy: script-src 'self' 'http://code.jquery.com'
```

You don't need to provide the specific location of the source code file, just the host site information. Each entry is separated from the next using a space. A CSP can define all sorts of content types. For example, you may decide that you want to support scripts found on the local page, but you don't want to support objects at all. In this case, you use the following CSP header:

```
Content-Security-Policy: script-src 'self'; object-src 'none'
```

Notice that the content types appear separated by a semicolon (;). CSP supports a number of different content type directives. You can find a quick reference guide containing a list of these directives, sample values, and descriptions at http://content-security-policy.com/. The important thing to remember is that using CSP can save you from that subtle hole in your code that a hacker could otherwise exploit.

Incorporating Libraries Safely

Some organizations simply add a library to their toolbox because it provide some functionality that the organization needs without much thought about how that library is put together or even if it's a good idea to use it. In at least some cases, organizations that don't put a lot of thought into the use of a library end up regretting it later because the library will contain a lot of security holes, run unreliably or slowly, or actually end up costing the organization more time than if it had simply created its own library in the first place. The following sections how you determine whether incorporating that interesting library you found into your application is actually a good idea.

Embracing the Sandbox Solution

It's quite likely that even if you check the third party JavaScript and clean the data you use thoroughly that something will still try to sneak by. Remember that it's easier to overcome the walls you build than to build them in the first place. When it comes to JavaScript security threats, you must consider that hackers will simply come up with new ways of circumventing your security every time you have a new bulletproof strategy in your arsenal.

> A potential way around this problem is to rely on sandboxing. Essentially this means that the third party library runs at a lower privilege level than your code does. Using sandboxing doesn't guarantee that your application will remain safe, but it does reduce the potential for security breaches. You can read about sandboxing techniques at http://www.slideshare.net/phungphu/a-twotier-sandbox-architecture-for-untrusted-javascript. The techniques described in the slideshow are complex, but they do work. As with everything, it comes down to determining what level of risk you're willing to tolerate when running an application. Chapter 10 discusses the use of sandboxes in detail.

Researching the Library Fully

Whenever you choose to add a library to your application, you're expressing trust in the creator of that library. Even with the help of a competent security expert, taking a library apart and examining every bit of code it contains (assuming the code is even fully available) will prove more time consuming that creating the library from scratch. Consequently, you must consider whether the library poses few enough risks to be part of an application solution that you create. When researching a library, consider these questions as part of your research:

- Are there any trade press stories that discuss the library in a negative way? (If the library is already known to have unpatched security holes, you may want to reconsider using it for your application.)
- Are other organizations using the library without problem?
- Is the company that created the library proactive in answering questions about library integrity?
- Does the company that created the library provide consistent bug fixes and updates?
- What sorts of questions are people asking as part of the online support provided for the library?
- How long as the library been around? (Libraries with long lives tend to have fewer problems and the longevity speaks of good support on the part of the creator.)

Defining the Precise Library Uses

No one uses every feature provided by every library found in an application. Some research sites state outright that many developers use a library for just one or two of the features it contains. Documenting how you use a particular library helps you mitigate risks associated with using the library by focusing your attention on those areas. More importantly, keeping track of specific usage areas tells you when you must perform an update and how you can decrease the size of a library by removing unused features from it. The more you know about how you use a library to perform specific application tasks, the more you reduce the risk of using it.

Keeping Library Size Small and Content Focused

It's essential to keep third party libraries small and focused on specific needs whenever possible. This means removing code that you won't ever use because such code tends to:

- Slow download times

- Increase security issues
- Reduce reliability
- Cause unnecessary updates

Fortunately, the best third party libraries are on your side. They want you to create a custom version of the library to use on your site. For example, when you use jQuery UI, you can create a customized version using the special builder shown in Figure 6-5.

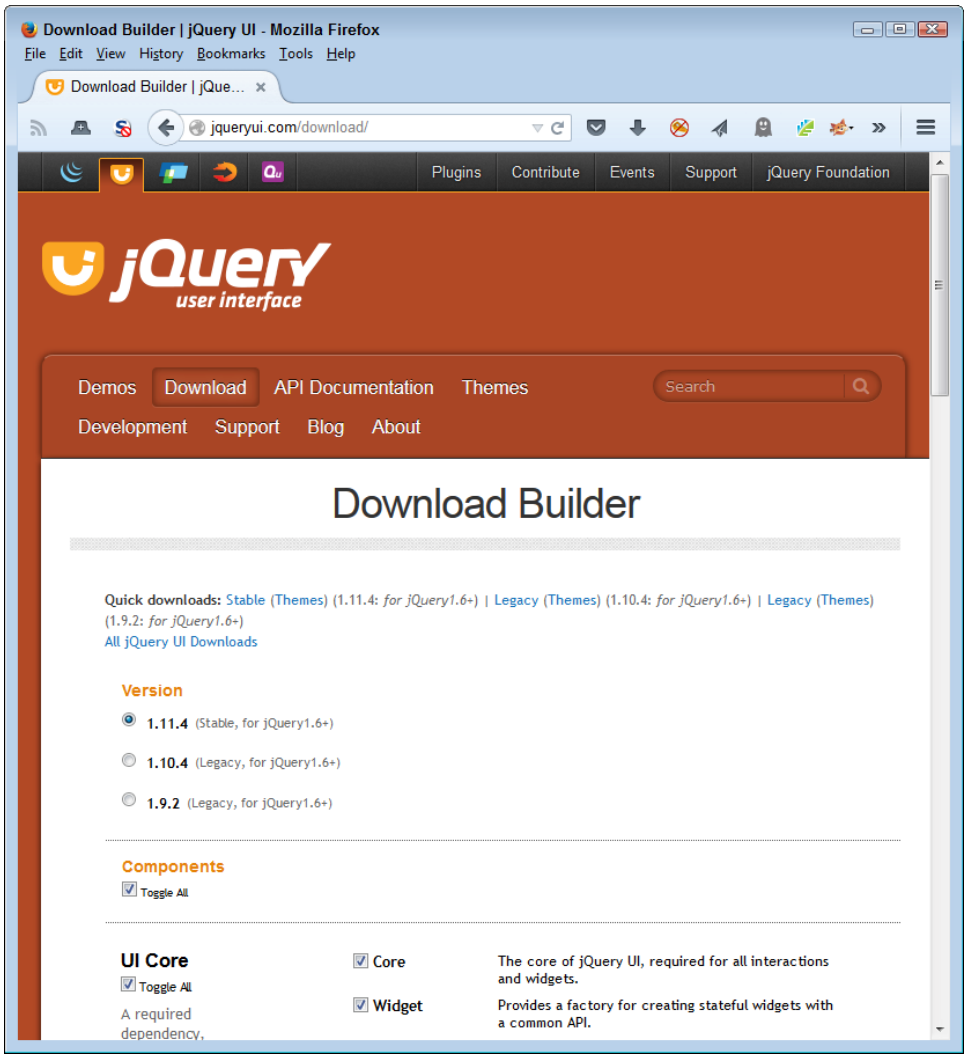


Figure 6-5. Using the jQuery UI builder helps you create a customized version of the library.

Using a builder means that the library you create will contain only the library elements you actually need. Of course, the down side is that you must provide hosting for the library, which means you won't get automatic updates as the creator makes them available. If you use an option like this, it's essential to check for the need to update your copy of the library on a regular basis.

Performing the Required Testing

Before you put any library into a production environment, you must build a test suite to determine whether the library works as you hope it will. You can start by using a vendor-specific test suite, but it's important to check using a test suite of your own design, as well, that mimics how your application will use the library. The testing process should include both valid and invalid data input checks. It's important to ensure that you get the expected output with valid inputs and some sort of exception when using invalid inputs. Chapters 11 and 12 provide details on how to perform various levels of testing on your application and the libraries it uses.

Differentiating Between Libraries and Frameworks

The "Differentiating Between Frameworks and Libraries" sidebar in Chapter 5 gives you an brief overview of how libraries and frameworks differ from a reliability perspective. Of course, there is more to the difference than simply reliability concerns. It's important to have a better understanding of precisely how the two differ and why you should care.

Frameworks provide templating for a site in a manner that libraries don't. When using a library, an application is simply relying on another source for code. Framework usage implies another level of participation, where the application now uses external sources for additional support. Because a framework integrates more fully into an application and provides more resources, you must also hold it to a higher standard than when working with a library (although you should hold both to a high standard anyway).

Trying to figure out whether a framework is safe to use can be quite hard. In fact, testing can become nearly impossible. With this in mind, third party security experts are attempting to quantify the safety level of frameworks and templating libraries. One of the better places to look for information is mustache-security (https://code.google.com/p/mustache-security/). This site provides you with seven levels of checks for frameworks as shown in Figure 6-6.

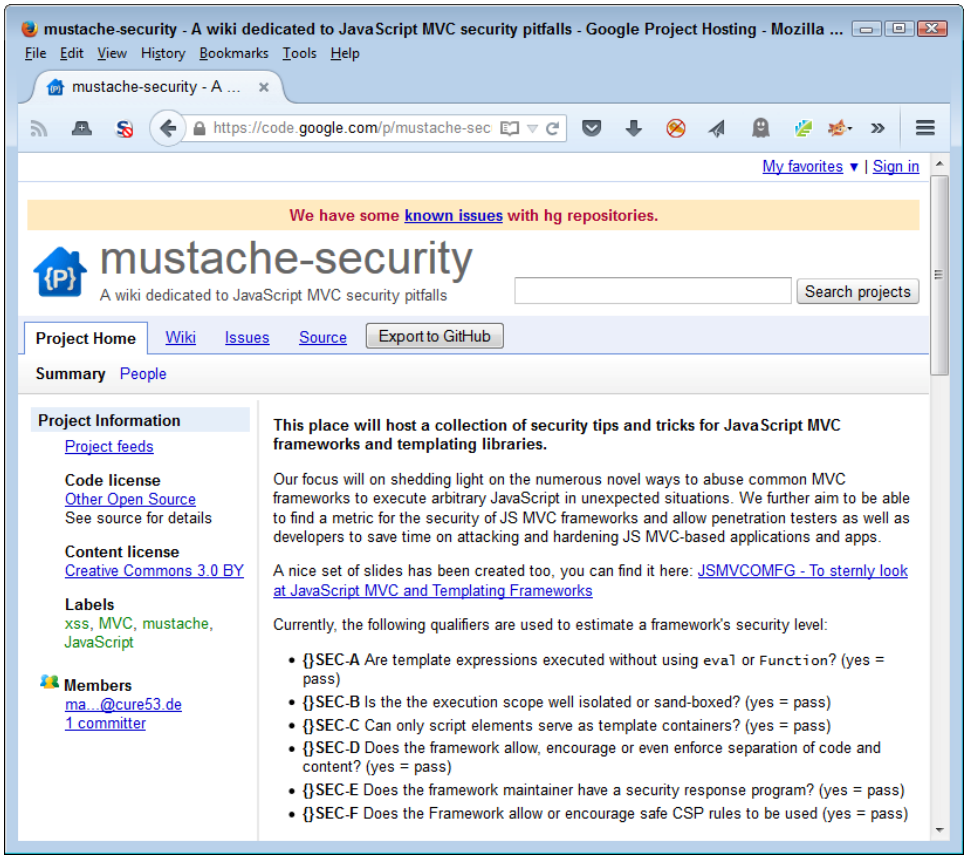
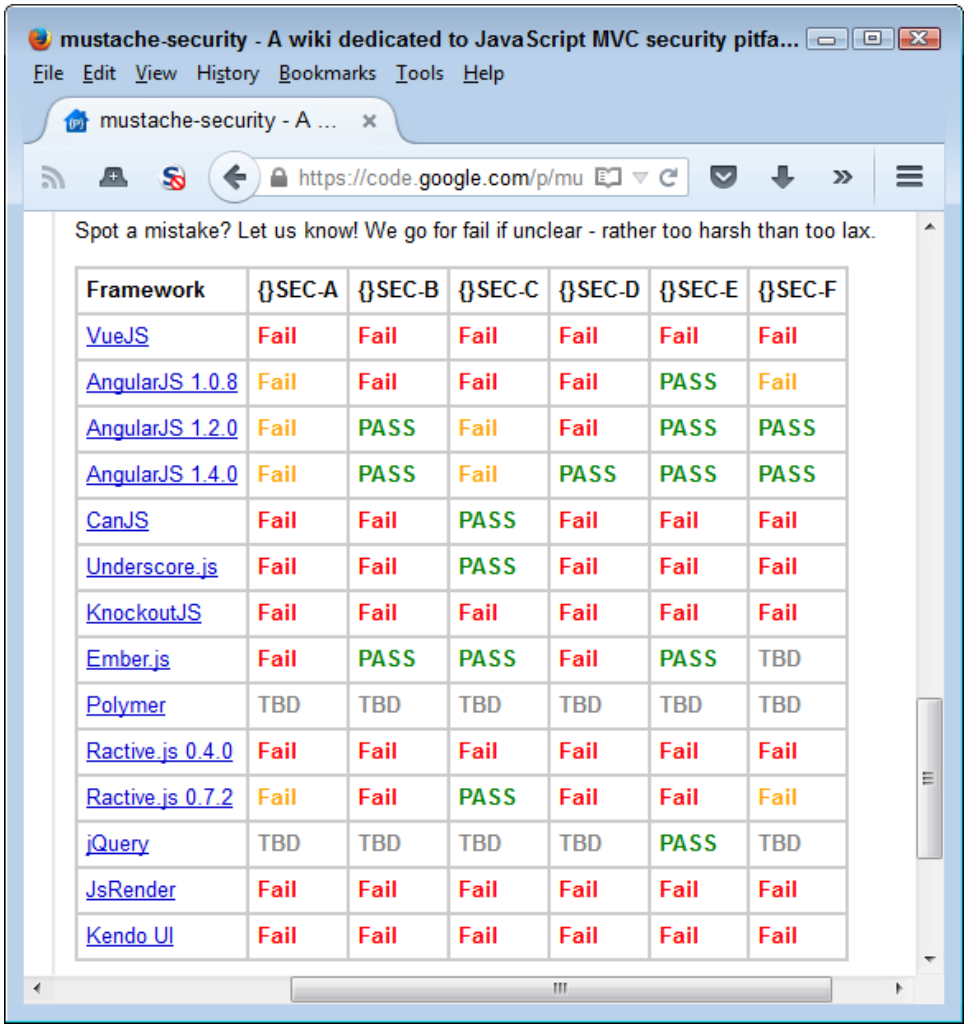


Figure 6-6. Sites such as mustache-security provide seven levels of checks for frameworks.

Looking a bit further down the page, you can see that the security news for frameworks isn't good. Most frameworks today contain serious security problems that you need to consider before using them for application development. Figure 6-7 shows the results for some of the most popular frameworks and templating libraries available today.



Framework	{}SEC-A	{}SEC-B	{}SEC-C	{}SEC-D	{}SEC-E	{}SEC-F
VueJS	Fail	Fail	Fail	Fail	Fail	Fail
AngularJS 1.0.8	Fail	Fail	Fail	Fail	PASS	Fail
AngularJS 1.2.0	Fail	PASS	Fail	Fail	PASS	PASS
AngularJS 1.4.0	Fail	PASS	Fail	PASS	PASS	PASS
CanJS	Fail	Fail	PASS	Fail	Fail	Fail
Underscore.js	Fail	Fail	PASS	Fail	Fail	Fail
KnockoutJS	Fail	Fail	Fail	Fail	Fail	Fail
Ember.js	Fail	PASS	PASS	Fail	PASS	TBD
Polymer	TBD	TBD	TBD	TBD	TBD	TBD
Ractive.js 0.4.0	Fail	Fail	Fail	Fail	Fail	Fail
Ractive.js 0.7.2	Fail	Fail	PASS	Fail	Fail	Fail
jQuery	TBD	TBD	TBD	TBD	PASS	TBD
JsRender	Fail	Fail	Fail	Fail	Fail	Fail
Kendo UI	Fail	Fail	Fail	Fail	Fail	Fail

Figure 6-7. Most frameworks and templating libraries today have a long way to go before they become secure.

The interesting thing about sites like these is that they often provide specific examples of how the framework fails. For example, click the AngularJS link in the table shown in Figure 6-7 and you see specifics on how the framework failed in the security area as shown in Figure 6-8. The information also includes workarounds you can use to help mitigate the problems, which is the main reason you want to spend some time researching the frameworks and ensuring you understand precisely where they fall short.

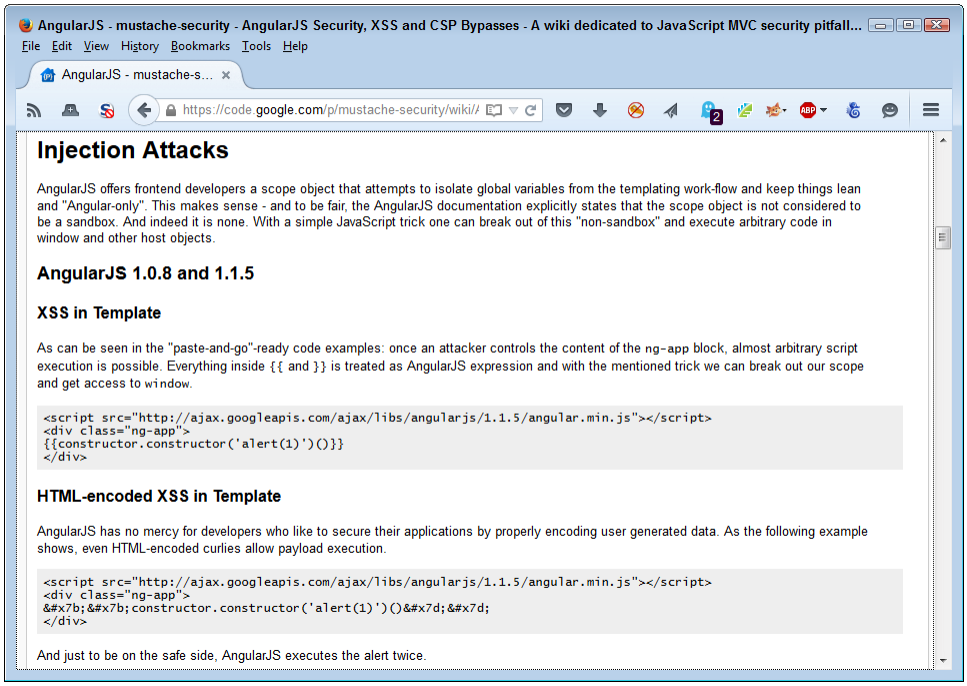


Figure 6-8. Some security sites provide example code demonstrating security issues in frameworks.

7

Using APIs with Care

Application Programming Interfaces (APIs) provide a means of using full-blown applications on someone else's server to meet your specific needs. Working with an API means sending a request to a black box and receiving a response of some sort back. The request can be as simple as a function call, but often requires that you supply data. The response can be in the form of data, but could also be some other service. For example, there are APIs for performing physical tasks such as opening doors. In fact, APIs can likely meet any generic need you might require. This chapter discusses the security implications of using APIs, including how APIs differ from libraries.

Part of discovering the security threats that APIs represent is viewing coded examples. This chapter uses a simple example as part of the explanation process. Even though you could use the target API for a significant number of application tasks, the point is to create some to use for discussion purposes and then look at how the use of an API could create problems for any application. View the example in this chapter as a point of discussion, rather than an actual programming technique.

> Even though this chapter does discuss specific APIs in both coded examples and as examples of where APIs can go wrong, the intent isn't to single out any particular API. Every API you use could have serious flaws that enable a hacker to gain access to your system or cause other sorts of problems for your application. The specific examples help you understand how all APIs can go wrong in general. In short, you shouldn't see the specific examples in this chapter as an indication of particular APIs to avoid using.

Of course, the result of all this exploration is to determine how you can use APIs safely as part of your application development strategy. Because APIs are so beneficial, you won't see them go away anytime soon as a bad idea from a security perspective. In fact, you can count on the use of APIs to expand because developers have constant time crunch issues that APIs solve. As more people use web-based applications and those applications become more complex, you can count on the use of APIs to expand to meet specific needs. As usage increases, so does the appeal for creating exploits by hackers who want to access your data and use it in ways you never intended. In short, if anything,

the security footprint of APIs will increase, so your diligence in using them must increase as well.

Differentiating Between APIs and Libraries

Some developers think that all sources of outside code are the same. However, APIs and libraries differ in some significant ways. For example, APIs represent an out-of-process because they run on another server, while libraries represent an in-process call because you include them as part of your application. Just the matter of out-of-process versus in-process significantly changes the way in which you must view security issues. The following sections help define the differences between APIs and libraries better so that you get a better understanding of the security implications of using one over the other.

> Some people use library and API interchangeably, but the two have specific meanings in this book. Chapter 6 defines libraries in details, but essentially, a library is external code you incorporate into an application using whatever technique the host language requires. You use libraries directly in your code as part of the application. As a result, a library is always an in-process coding technique, even when the library source resides on a third party server. In order to use the library, the code on the third party server downloads onto your server and becomes part of the application, running in the same process as the application.
>
> This book also differentiates between APIs used to perform tasks and those used as part of an application development effort. For example, the JavaScript Web APIs described at http://www.w3.org/standards/webdesign/script define the JavaScript language elements. This sort of API defines a language used for scripting purposes. Yes, this is a legitimate API, but not in the context used in this book. The book context is more along the lines of a Web service, rather than a language-specific specification.

Considering the Differences in Popularity

Libraries and APIs both provide benefits to the developer by saving development time. In fact, most web-based applications today use a combination of libraries and APIs to perform tasks. However, APIs appear to be gaining in popularity faster than libraries for some good reasons:

- Reduced Resource Usage: APIs run on someone else's server, so you don't need to worry about obtaining the required resources. However, the fact that the code runs on someone else's server also means that you lose control of your data at some point and never see the actual API code so that you can assess the security threats it represents.
- Decreased Coding Requirements: Using an API is often easier and requires less code than using a library. Of course, you also have less control over an API. The tradeoff is flexibility and the price is complexity. Given that developers are increasingly overloaded, the reduced coding requirements of APIs can make them quite attractive.

However, the reduced flexibility also represents a potential source of security issues because you can no longer tailor code usage to meet your particular requirements.

- Smaller Learning Curve: It's usually easier to learn how to use an API than it is a library because the API is generally in a client/server form. The biggest issue is determining how to format the data so that the API understands what you want it to do. It's in the data formatting that the greatest security issue exists. A hacker could cause malformation of your query that causes the server to crash or force the server to respond in unexpected ways. Because the query source is traced to your application, the host will blame your application, rather than the hacker that caused the problem.

Defining the Differences in Usage

It isn't possible to use APIs in every case that you use a library (and vice versa). Some situations call for an integrated approach. For example, you can't easily code a user interface using an API—it really does call for a library. (You can obtain the data for the user interface, as described for the Google API example later, but this is different from actually creating the interface elements.) The following list describes typical uses for APIs and their security implications:

- Data Query: The most common use for APIs is to make a request and obtain data as a result. The request commonly contains the parameters of the requested data. A data source can include databases, sensors, calculations, and analyses. Whenever you deal with data, man-in-the-middle attacks are common. However, a hacker could simply choose to modify the data in order to mislead you or to use the received data as a method for gaining access to your network.
- Supervisory Control and Data Acquisition (SCADA): The Internet of Things (IoT) relies on web interfaces to perform a number of SCADA tasks. For example, with the right thermostat, it's relatively easy to adjust the temperature of your house using a web application from your smartphone, wherever you might be. A hacker could easily plug into the same interface and lower your house temperature until the pipes freeze or make it so hot that you need to open the doors when you get home. Of course, SCADA encompasses all sorts of things, including industrial controls. In this case, a breach of security not only affects data, but has physical effects as well.
- Monitoring: A middle ground between data query and SCADA is the act of monitoring systems. For example, the police may want to monitor certain street cameras during a protest or other event. APIs make it possible to create applications that can monitor any camera, anywhere, as needed to provide protection for the general populace. A hacker could trick the same API into providing false information.
- Multimedia: An API need not work only with textual or other abstract data. It can also perform tasks related to the other senses, with visual and audio being the most common. The control and manipulation of multimedia creates an environment in which users interact with applications in non-traditional ways and the results are often surprising. By combining multimedia with data science, it's possible to recognize new patterns and objects in everyday situations, enhancing a user's ability to perform tasks, such as visualizing scenes in ultraviolet, which might otherwise be impossible. However, it's important to realize that multimedia is simply another

form of data and data of all sorts is easy for hackers to intercept, corrupt, and otherwise manipulate in unwanted ways.

- Location: Even though geolocation is the most common form of location acquisition, query, and manipulation, it's important to understand that APIs exist to work with all sorts of location information. Location, like multimedia, is a special kind of data. In this case, it provides a two- or three-dimensional point in space relative to a particular target space, such as the earth. For that matter, some APIs add a fourth dimension to the equation, time. Imagine what would happen if hackers chose to modify location data such that two objects, such as cars, attempted to occupy the same space at the same time (resulting in a crash).

Extending JavaScript Using APIs

It's important to consider security from a real world perspective. The following sections describe some issues in locating appropriate APIs for testing and application development purposes. You then look at an example for the Google Maps API in action. This simple application demonstrates the need to consider issues such as malformed requests, server errors, and errant data return. The example is interesting because it does demonstrate how to create an application that works with the browser canvas to draw part of the user interface based on API return data.

> The best way to work with the example described in this section is to use the downloadable source, rather than type it in by hand. Using the downloadable source reduces potential errors. You can find the Google Maps API example in the \S4WD\Chapter07\GoogleAPI folder of the downloadable source.

Locating Appropriate APIs

JavaScript provides an extensible programming environment that supports both APIs and libraries. Developers have taken advantage of both forms of extensibility to create some astounding sharable code that performs tasks that you might not think possible. For example, it's possible to use a JavaScript application to interact with the vibrate feature of a mobile device to create special effects (check out the technique at https://developer.mozilla.org/en-US/docs/Web/Guide/API/Vibration). With the right resource, you can find an API to perform just about any task you can think of. The following list provides you with some ideas of where to look for that API of your dreams:

- {API}Search (http://apis.io/)
- 40 useful APIs for web designers and developers (http://www.webdesignerdepot.com/2011/07/40-useful-apis-for-web-designers-and-developers/)
- API Directory (http://www.programmableweb.com/apis/directory)
- API For That (http://www.apiforthat.com/)
- APIs Dashboard (http://www.programmableweb.com/apis)
- APIs Data.gov (https://www.data.gov/developers/apis) (click the "browse the current catalog for APIs" link)

- Guide to Web APIs (https://developer.mozilla.org/en-US/docs/Web/Guide/API)
- MashApe (https://www.mashape.com/)
- Public APIs (https://www.publicapis.com/)
- Web API Interfaces (https://developer.mozilla.org/en-US/docs/Web/API)

This is just a sampling of the API search sites online. Most of them provide write-ups on the APIs; some sites also provide reviews. Before you select a particular API, make sure you read as much as you can about it. Perform tests with the API on a test system. Make sure you verify the sorts of data that the API actually transfers between your test system and the host by using a packet sniffer. Even though it may sound a little paranoid to test APIs in this way (see the "Accessing APIs Safely from JavaScript" section of this chapter for additional details) performing the tests helps reduce your risk. Remember that security is always a balance between risk and the benefits you obtain from taking the risk.

> Some API providers have so many APIs that they provide their own listing. For example, Google provides a wealth of APIs and you can find them listed on GData API Directory at https://developers.google.com/gdata/docs/directory. It's important to realize that many host-specific listings contain biased information and reading about the capabilities and deficiencies of the API on another site is always recommended.

Creating a Simple Example

Creating a secure application that relies on APIs is harder than you might think because even the best APIs rely on a certain amount of trust. You trade the risk of trusting the host for the data or other resources the host provides. To see how the security issues come into play, this section works through a simple example that uses the Google Maps API. In order to make the API usable, it also relies on the jQuery (http://jquery.com/) and jQuery UI (http://jqueryui.com/) libraries to display the data and associated user interface. Rarely will you use an API without also relying on libraries to interact with the data in some way.

> In order to use this example, you must obtain a developer key. Google provides two kinds of keys: paid and free. You only need the free key for this example. The paid key does provide considerably more flexibility and you'll likely need it for any full-fledged application you create. However, for experimentation purposes, the free key works just fine. You can obtain this key at https://developers.google.com/maps/licensing. Make sure you understand the terms of service fully before you begin working with the Google Maps API. You can also find some additional assistance in using the Google Maps API with JavaScript at https://developers.google.com/maps/documentation/javascript/tutorial.

It's best to create the code for this example in several steps. The first is to add the required jQuery references shown here.

```
<script
   src="http://code.jquery.com/jquery-latest.js">
</script>
```

```
<script
   src="http://code.jquery.com/ui/1.9.2/jquery-ui.js">
</script>
<link
   rel="stylesheet"
   href="http://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
```

Even though the code makes a reference to an external library, the code from the library is included as part of the application as an in-process element. Chapter 6 describes all of the security issues regarding library use. In addition, you also need to add a reference to the Google Maps API as shown here.

```
<script type="text/javascript"
   src="https://maps.googleapis.com/maps/api/js?key=Your Key Here&sensor=false">
</script>
```

> This example won't work at all unless you replace the words, Your Key Here, with the key that you receive from Google. Consequently, this particular step is important because it's the one step that you must perform even if you're using the code downloaded from the book's site.

Even though both the library and the API references rely on the `<script>` tag, the manner in which they use the tag is different. Notice that the library accesses a file without any additional information. The API requires additional information in the form of a key and a sensor configuration option in this case. As the section progresses, you'll see other differences.

Now that you have all of the required references in place, it's time to create a canvas to draw the map. The canvas is simply a `<div>` as shown here.

```
</div>
   <div id="MapCanvas">
</div>
```

You must provide style information that gives the `<div>` size or else the map won't appear on screen, even when Google sends it to you. The example uses the following style information:

```
#MapCanvas
{
   height: 90%;
   width:100%;
}
```

In addition to the canvas, the example provides two textboxes for input and a button you can use to request a new map. There isn't anything too complex about the interface, but it gets the job done (the example is preset to show the location of Milwaukee, Wisconsin in the center of the map—you can change the longitude and latitude to any other location you wish).

```
<div id="Input">
   <label for="longitude">
      Longitude:
   </label>
   <input id="longitude"
          value="-87.95"
          type="text" />
   <label for="latitude">
```

```
      Latitude:
   </label>
   <input id="latitude"
          value="43.04"
          type="text" />
   <input id="submit"
          value="Get Map"
          type="button" />
</div>
```

The code for this example uses many of the jQuery and jQuery UI tricks you've seen in other applications. For example, the application creates a spinner for the longitude and latitude controls to make it easier to move the center of the map incrementally. Moving an entire degree at a time wouldn't make the application very useful, so the two spinners change the inputs by a tenth of a degree at a time (even this setting may be too large and you might want to change it). Notice the use of the `step` option to perform this task. Of the code that follows, the `GetMap()` function is the most important because it actually displays the map on screen.

> Latitudes range from 90 degrees north to -90 degrees south, so the example reflects this requirement. Likewise, longitudes range from 180 degrees west to -180 degrees east of Greenwich, England. You can read more about latitude and longitude at http://geography.about.com/cs/latitudelongitude/a/latlong.htm.

```
$(function()
  {
      // Track the current latitude using a
      // spinner control.
      var Latitude = $("#latitude").spinner(
         {
            min: -90,
            max: 90,
            step: .1,

            change: function(event, ui)
            {
               if (Latitude.spinner("value") < -90)
                  Latitude.spinner("value", -90);
               if (Latitude.spinner("value") > 90)
                  Latitude.spinner("value", 90);
            }
         });

      // Track the current longitude using a
      // spinner control.
      var Longitude = $("#longitude").spinner(
         {
            min: -180,
            max: 180,
            step: .1,

            change: function(event, ui)
            {
               if (Longitude.spinner("value") < -180)
                  Longitude.spinner("value", -180);
```

```
            if (Longitude.spinner("value") > 180)
               Longitude.spinner("value", 180);
         }
      });

   // This function actually displays the map on
   // screen.
   function GetMap()
   {
      // Create a list of arguments to send to Google.
      var MapOptions =
      {
         center: new google.maps.LatLng(
                     Latitude.spinner("value"),
                     Longitude.spinner("value")),
         zoom: 8,
         mapTypeId: google.maps.MapTypeId.ROADMAP
      }

      // Provide the location to place the map and the
      // map options to Google.
      var map = new google.maps.Map(
         document.getElementById("MapCanvas"),
         MapOptions);
   };

   // The example provides two methods of getting a
   // map: during page loading or by clicking Get Map.
   $(window).load(
      function()
      {
         GetMap();
      });

   $("#submit").click(
      function()
      {
         GetMap();
      });
})
```

The `GetMap()` function performs the actual task of obtaining the map. To do this, your application must create a list of map options. The example shows a simple, but typical list. The most important of these options is where to center the map. In this case, the map automatically centers itself on Milwaukee, Wisconsin, but you can change the settings to any location you want. The example uses a zoom factor of 8 and you'll see a road map. The Google Maps API actually provides a number of map types that you can try.

There are two times when `GetMap()` is called. When the application loads, you see Milwaukee, Wisconsin (unless you change the default settings). After you change the inputs, you can also click Get Map to display a new location. Figure 7-1 shows typical output from this application.

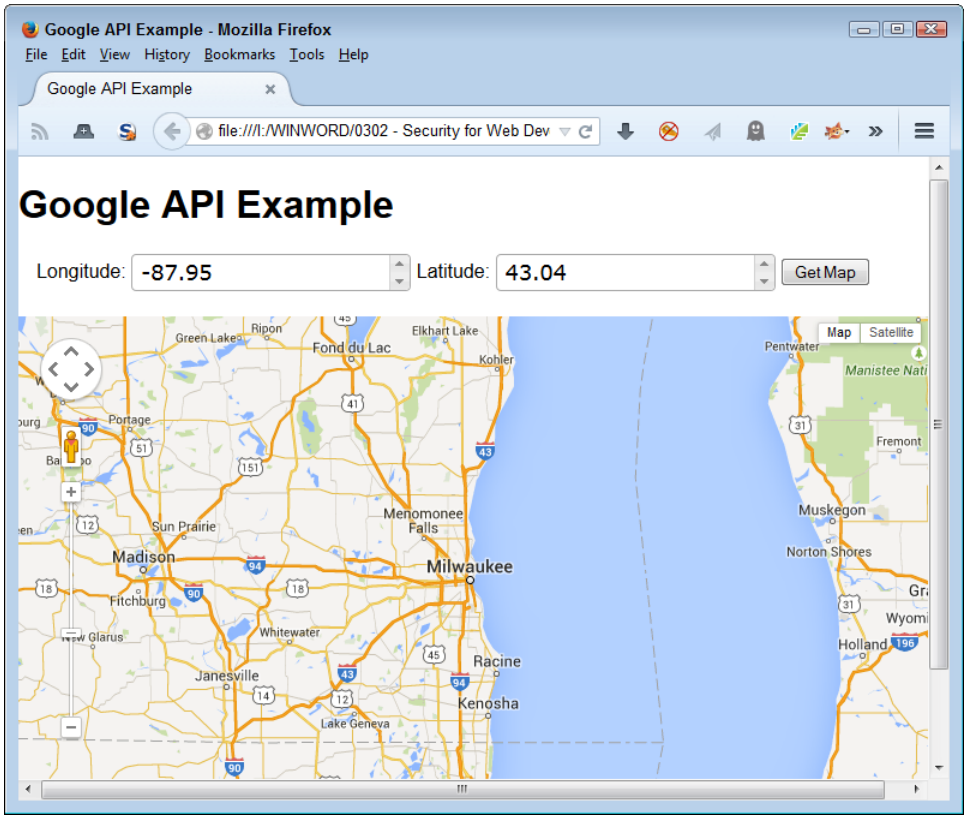


Figure 7-1. The example can show any location for which you have a longitude and latitude.

This chapter doesn't provide a detailed look at the Google Maps API, which is relatively complex. What it does provide is a simple look at something you could easily expand into a full-fledged application later. Many organizations today use maps for all sorts of interesting purposes. The article at http://blog.smartbear.com/software-quality/bid/242126/using-the-google-maps-api-to-add-cool-stuff-to-your-applications provides some additional ideas on how you can use the Google Maps API with your browser-based application.

Defining the Security Threats Posed by APIs

It's important to know what sorts of API threats you might face as a developer. Even though the following sections may not contain the most recent of API threats, they do represent various kinds of API threats and provide you with the kind of information needed to help research the latest threats to your application. These real world examples make it possible for you to understand what sorts problems you'll encounter.

Ruining Your Good Name with MailPoet

Blogging represents a major investment of time for many people. I use my blog to communicate with my readers and tell people about my latest self-sufficiency efforts. You can find a blog for just about any purpose and on just about any topic. People place their reputations in their blogs as a way to help the world as a whole. So, when a blogging software plugin, such as MailPoet, has a problem, it can affect the reputations of every blogger who uses it. That's precisely what happened on July 1st, 2014.

The vulnerability allows the attacker to place any PHP file on the system using a remote upload, which means that your blog may now serve up phishing attacks with the information it used to provide. At first the hack would cause the site to crash in some cases, but this flaw has since been fixed, so unless you're vigilant, you may not even know you've been hacked.

A problem with this particular vulnerability is that the entity that discovered the problem (Sucuri) didn't apparently give MailPoet enough time to fix and distribute a fix before revealing it, so there is lots of finger pointing going on. This particular plug-in has been downloaded 1.7 million times, but is used on multiple blogs by some individuals, so it significant potential to cause problems. The MailPoet threat is an example of the need to verify API security before you begin development and to perform constant checks as you continue to use the API.

Developing a Picture of the Snappening

SnapChat is a great way to exchange photos with other people in a way that isn't permanent. The purpose of the service is to allow a freer flow of information in an environment with fewer risks. It actually does work when everyone plays by the rules, but not everyone does (as explained in Security Lessons Courtesy of Snapchat).

The API is extremely easy to hack because of a flaw in a third party library. Instead of actually fixing the problem, SnapChat has responded recently by admonishing users not to download third party programs that allow them to overcome the built-in rules. The result of not providing a solid API in this case is the leaking of 90,000 plus pictures that are likely more revealing that anyone cares to know. Talk about a loss of reputation! Of course, the usual amount of extortion will take place and it's unlikely that anyone will ever be able to track down the monetary implications because no one will be willing to talk about it.

The short take on this particular API hack is that it just doesn't pay to use some services—no matter how appealing they might be. This particular threat demonstrates the need to check vendor reputation before you begin development using the API. In addition, it shows that you need to research the API before using it because many hackers knew of this flaw before the trade press started making an issue of it.

Losing Your Device with Find My iPhone

The Find My iPhone service is supposed to protect your iPhone should it get lost. You can locate your iPhone, see where it has been, and lock it to keep prying eyes away. However, Apple didn't secure access to iCloud, the service that supports Find My iPhone, very well. It's vulnerable to a brute force attack—one in which the attacker keeps trying passwords until one is successful.

Some hackers discovered this flaw and used it to lock people's iPhones. For a small sum of $100, the victim could get their iPhone unlocked. If the victim resisted, the hackers promised to erase all their data. The only way around the problem without paying is to restore the device to factory settings. Fortunately, this story does have a happy ending, the hackers have been arrested. No one is saying just how many people paid the fee, how many lost their data, and just how many phones were hacked, but to create this much press it has to be more than a few.

This particular threat is an example of the need to test inputs and outputs. By thinking through how the API worked, it's possible that developers could have figured the brute force approach out sooner than later and avoided potential problems.

Leaking Your Most Important Information with Heartbleed

You depend on Secure Sockets Layer (SSL) to provide secure communication between client and server. Unfortunately, like any other piece of software, SSL can contain vulnerabilities. In this case, the use of a feature called a heartbeat in OpenSSL versions 1.0.1 through 1.0.1f allows an attacker to gain access to user passwords and long term private keys used to encrypt communication.

According to InfoWorld, Heartbleed was to blame for the Community Health Systems breach that released patient records for 4.5 million patients. The price tag for just this one breach could cost up to $150 million. The interesting thing about Heartbleed is that it has serious staying power. According to ComputerWorld, as of June 24, 2014, there were still 300,000 servers that were susceptible to Heartbleed (which is a long time for applying a patch given that the vulnerability was discovered in April). The Heartbleed vulnerability is so serious that it has spawned its own information site, http://heartbleed.com/.

You should see this threat as a wakeup call not to trust any particular API to remain secure. Hackers are constantly testing the waters to see just what they can get by with. In this case, constant testing of the API might have yielded information about the vulnerability before it became a problem. Certainly, relying on just one means of securing information is probably not the smartest thing to do.

Suffering from Shellshock

Some pundits are calling Shellshock worse than Heartbleed. It's hard to say that one is worse than the other when both are incredibly bad and affect a large number of servers. In this case, a bug in the bash utility used by Linux administrators can be hacked by creating a malformed environment variable and then running a bash script. Access to this functionality can be provided through the Common Gateway Interface (CGI) or PHP.

The biggest limiting factor for this hack is that it isn't automatically exploitable—there has to be an outside interface to access the server. There are reports that larger organizations, such as Yahoo and WinZIP have been hacked using Shellshock. The potential for Shellshock hacks is huge because the number of affected servers is larger than those affected by Heartbleed. In addition, a viable patch was longer in coming; although, one is in place now.

This is an example of keeping your data localized under lock and key. By providing an outside interface to critical data, organizations open themselves up to data loss. Keeping sensitive data under wraps is the only way to ensure that it actually remains secure. Of

course, you need to make some data accessible, but doing so should be the option of last resort and you should make only the required data available.

Accessing APIs Safely from JavaScript

It's possible to access APIs safely using JavaScript if you perform some testing first. Of course, testing takes time and you may encounter resistance to spending the time to ensure the API works as advertised. All you really need to do is ask about the value of the organization's data compared to the time required for testing. The answer is usually obvious, the data is always worth the testing time. With this in mind, the following sections describe some measures you can take in using APIs safely.

Verifying API Security

The "Defining the Security Threats Posed by APIs" section of this chapter describes some of the threats that APIs can present to the developer. Unfortunately, the trade press often expose these threats after you implement a solution using the API, making it hard to change the API you're using without incurring a huge cost. However, verifying that the API is currently clean by looking for stories about it is always a good place to start.

It also pays to look at the vendor's reputation for security. New vendors are untested and therefore, unreliable. You use them at a much greater risk than a known vendor, one who has had products on the market and tested for some amount of time. Partially offsetting the new versus experienced vendor is the fact that hackers also target vendors with a larger market share in order to improve the odds of getting something truly useful in exchange for their efforts.

The manner in which the API is constructed, the documentation provided, and the amount of input requested for calls all point to the level of security provided. It may be a nuisance to sign up for a service provided by Google, but the fact that you must use a special key to access the API can reduce the APIs attack surface at least a little. In addition, the Google APIs don't ask you to provide sensitive data; although, an attacker could possibly derive some data by simply looking at your requests. For example, asking for maps at specific longitudes and latitudes could mean a new business venture in the area or some other reason for the increased attention.

The Google API is a good choice for the example in this chapter because Google has a relatively good reputation for security. In addition, it's easy to verify that you actually are interacting with Google in most cases and that the results you receive are of the right type. Google's documentation helps you perform the sorts of checks that make it easier to verify that the API is secure.

Testing Inputs and Outputs

A simple way to verify the usefulness and security of an API is simply to test known quantities. Check known inputs and outputs to ensure the API works as advertised. It may surprise you to discover that some APIs really don't work as advertised. In some cases, the problem isn't one of actual coding errors, but one of not understanding how the API works.

Of course, many organizations perform an input/output test during the initial design and development phase for an application. However, creating a test harness lets you perform

the tests at random. The benefit of this approach is that the tests can help you detect potential data corruption or other potential security problems. Running tests at intervals helps you ensure that the API remains viable. The tests also tend to trash any conclusions eavesdroppers may make about the queries you make against the API.

The Google Maps API example in this chapter should have also included code to verify both the input and output data. These checks were left out in order to make the application easier to understand. However, in a real application, you need to test the data before you submit it and verify that you get the kind of information you expected as a return value. Of course, it's not possible to check the data specifically because the map will change with each call.

Keeping Data Localized and Secure

The more data you send to an API, the higher the probability that someone will intercept that data. The data need not be intercepted en route to present a problem. If the data ends up residing on the host's server, then anyone accessing the server can likely access the data as well. The best way to keep a secret is not to tell anyone. Sharing as little data as possible with an API is a good idea.

The use of an encoded key with the Google API is a good security measure because it positively identifies your organization. However, the key is important for another reason. In some cases, other APIs use identifiers that a hacker could track back to your organization. Anyone intercepting the Google key won't know anything more about your organization than before.

One example of an API that used to require input that a hacker could track back to an organization was Amazon.com. Some APIs used to rely on the associate identifier that an organization also attaches to links for buying products. A hacker could discover this identifier with relative ease and could begin tracking an organization's API calls as well.

Coding Defensively

When writing your application, always assume the inputs and outputs are wrong, that the user is going to handle everything incorrectly, and that there is a hacker looking over your shoulder. Yes, developers need to develop a sense of true paranoia in order to avoid some of the problems prevalent in applications today. As the book progresses, you see the usefulness of techniques such as API sandboxing in order to determine just how bad, bad can be when it comes to interacting with the API. The important thing is to assume nothing about code—even the code you create.

8

Considering the Use of Microservices

Microservices are a relatively new technology that breaks huge monolithic applications into small components. Each of these small components acts independently and performs just one task well. Because of the technologies that microservices rely on and the way in which they're employed, microservices tend to provide better security than some of the other technologies described so far in the book. However, just like any other technology, microservices do present opportunities for hackers to cause problems. It's important to remember that any technology has gaps that hackers will exploit to accomplish tasks. The goal of the developer is to minimize these gaps and then ensure as many safeguards are in place as possible to help in the monitoring process.

Because microservices are so new, the chapter begins by spending a little more than the usual time explaining them. This book doesn't provide you with a complete look at microservices, but you should have enough information to understand the security implications of using microservices, rather than older technologies you used in the past. In addition, it's important to consider the role that people will play in this case. A hostile attitude toward microservice deployment can actually cause security issues that you need to consider during the development stage.

The chapter discusses how you might create a microservice of your own (but doesn't actually provide source code because this is a book about security and not about writing microservices). The example focuses on a combination of Node.js and Seneca to create a simple microservice, and then access that microservice from a page. The point of the example is to discuss how microservices work so that you can better understand the security information that follows in the next section. The reason for using the combination of Node.js and Seneca is that these applications run on the Mac, Windows, and Linux platforms. Other microservice products, such as Docker, only run on Linux systems at present.

The chapter finishes by reviewing the importance of having multiple paths for microservice access. One of the advantages of using microservices is that you can employ multiple copies of the same microservice to reduce the risk of an application failing. In

short, microservices can be both more secure and more reliable than the monolithic applications they replace.

> The best way to work with the examples described in this chapter is to use the downloadable source, rather than type it in by hand. Using the downloadable source reduces potential errors. You can find the source code examples for this chapter in the \S4WD\Chapter08 folder of the downloadable source.

Defining Microservices

Applications that work on a single platform will eventually go away for most users. Yes, they'll continue to exist for special needs, but the common applications that most users rely on every day won't worry about platform, programming language requirements, or any of the other things that applications need to consider today. Microservices work well in today's programming environment because they define a new way of looking at code. Instead of worrying how to create the code, where to put it, or what language to use, the developer instead thinks of just one task that the code needs to perform. That task might not necessarily even fit in the application at the moment—it may simply represent something interesting that the application may need to do when working with the data. In the new world of application development, applications will run anywhere at any time because of technologies such as microservices. The following sections provide you with a good overview of precisely what microservices are and why you should care about them.

Specifying Microservice Characteristics

Many developers are used to dealing with monolithic designs that rely heavily on Object-Oriented Programming (OOP) techniques. Creating any application begins by defining all sorts of objects and considering all sorts of issues. Current application design techniques require a lot of up front time just to get started and they're tied to specific platforms. Microservices are different. Instead of a huge chunk of code, you write extremely small pieces of code and make many decisions as you go along, rather than at the beginning of the process. Microservices have these characteristics:

- Small: Each microservice performs just one task.
- Language Independent: Every microservice relies on the language that best suits the task it performs without any consideration for the needs of any other microservice.
- Data Transfer Independence: Even though most microservices currently rely on JavaScipt Object Notation (JSON) to transfer data, you can use any method of data transfer that works best for the microservice.
- Queued Messages: Communication typically occurs using an asynchronous messaging system so that no one microservice can cause delays in the application as a whole.
- Dumb Pipe: A problem with many methods of communication today is that the intelligence resides in the pipe. Microservices rely on a dumb pipe and intelligent services. Most microservices rely on Representational State Transfer (REST) for communication purposes.

- Decentralized: Each microservice is separate from every other microservice and from the application as a whole. A failure of one microservice typically won't affect application operation. Each microservice receives separate monitoring.
- Platform Independence: Any application can make use of any microservice no matter what platform the application is running on and regardless of which platform the microservice uses.

You might wonder about the size of microservices—what performing just one task well really means. Think about a string for a second. When working with a monolithic application, you have a single object that can capitalize, reverse, and turn the string into a number. When working with a microservice, you create a single microservice to perform each task. For example, one microservice would capitalize the string, another reverse it, and still another turn it into a number. When you think about microservices, think focused and small.

From a developer perspective, microservices represent the ultimate in flexibility and modularity. It's possible to work on a single function at a time without disturbing any other part of the configuration. In addition, because updates are so small, it's not like an API where you have a huge investment in time and effort. When you make a mistake, correcting it is a much smaller problem.

Differentiating Microservices and Libraries

It's important to realize that microservices don't execute in-process like libraries do. A microservice executes on a server like an API. This means that you don't have the security risks with microservices that you do with libraries. It's possible to separate your application code from the microservice completely.

The calling syntax for a microservice also differs from a library in that you create a JSON request and send it to the server. The response is also in JSON format. The use of JSON makes it possible to work with data in a rich way without resorting to XML. Working with JSON is much easier than working with XML because JSON is native to JavaScript and it provides lighter weight syntax. You see how this works later in the chapter. For now, just know that microservices work differently than library calls do for the most part.

From a security perspective, microservices tend to be safer than libraries because they don't execute in-process and you can guard against most forms of errant data input using best practices approaches to working with JSON. Of course, hackers can thwart any effort to make things more secure and microservices are no exception.

Differentiating Microservices and APIs

APIs often require that you create an object and then execute calls against that object. Requests can take a number of forms such as REST, HTML request headers, or XML. Responses could involve direct manipulation of objects on the page (as in the case of the Google Maps API example shown in Chapter 7). The process is cumbersome because you're working with a large chunk of monolithic code that could contain all sorts of inconsistencies.

Like APIs, microservices do execute out of process. However, unlike APIs microservices aren't huge chunks of monolithic code. Each microservice is small and could execute in its own process, making it possible to isolate one function from another with complete

assurance. Data exchanges occur using just one approach, JSON, which is likely the best approach to use today because it's simpler than working with XML.

Considering Microservice Politics

By this time you reach this section of the chapter, you know that microservices have a lot to offer the developer, IT in general, and the organization as a whole. Using microservices makes sense because the technology makes it possible to create applications that work on any device in any location without causing hardship on the developer. Unfortunately, monolithic application development scenarios tend to create fiefdoms where a hierarchy of managers rule their own particular set of resources. Because microservices are small, easily used for all sorts of purposes, and tend not to care about where needed data comes from, they break down the walls between organizational groups—upsetting the fiefdoms that ruled in the past. As in any situation of this sort, some level of fighting and even sabotage is bound to happen.

The sabotage part of the equation is what you need to consider as a developer. It's unlikely that anyone that anyone will purposely spend time trying to kill a microservices project, but the subtle reluctance to get tasks done or to do them correctly can kill it just as easily. All organizations have a "we've never done it that way here before" attitude when it comes to new technologies—inertia has a role to play in every human endeavor, so it shouldn't surprise you to find that you have to overcome inertia before you can start your first project.

From a security perspective, flaws induced in the project during this early stage leave openings that hackers are full aware of and will almost certainly exploit if your organization becomes a target (or sometimes by pure random chance). With all this in mind, it often helps to follow a process when incorporating microservice strategies into your programming toolbox. You won't always follow these steps in precisely the order listed, but they do help you overcome some of the reluctance involved in working with microservices.

1. Form a development team that is responsible for microservices development that's separate from the team that currently maintains the monolithic application.
2. Create a few course-grained microservices for new application features to start.
3. Develop microservices that provide self-contained business features at the outset so that you don't have to worry about interactions as much.
4. Provide enough time for existing teams to discover how to use microservices and begin incorporating them into existing applications. However, don't move existing applications completely to microservices until you have enough successes so that everyone agrees that making the move is a good idea.
5. As development progresses with the initial microservices and you can see where changes need to be made, create finer-grained microservices to produce better results.
6. Standardize service templates so that it's possible to create microservices with a minimum of chatter between groups. A standardized template also tends to reduce security issues because no one has to make any assumptions.
7. Create enough fine-grained microservices to develop a complete application, but don't focus on the needs of an existing application—consider creating a new application instead.

8. Obtain the tools required to perform granular monitoring, log aggregation, application metrics, automated deployment, and status dashboards for data such as system status and log reporting.
9. Build a small application based solely on microservice development techniques. The idea is to create a complete application that demonstrates microservices really can do the job. Developing a small application tends to reduce the potential for failure for a development group that is just learning the ropes.
10. Slowly cross-train individuals so that the sharp divisions between skill sets diminishes.
11. Break down the silos between various groups. Start creating microservices that makes code, resources, and data from every group available to every other group without consideration of the group that originated the item.
12. Slide development from the existing monolithic application to one designed around microservices.
13. Beginning with a small monolithic project, move the monolithic project entirely to a microservices environment if possible. Perform the task slowly and use metrics after the addition of each microservice to ensure that the application truly does work faster, run more reliably, and stay more secure.
14. Prune older microservices from the system as you replace them with finer-grained and more functional replacements.

Making Microservice Calls Using JavaScript

The previous section helped you understand what a microservice is, but it doesn't show you how a microservice works. The following sections provide you with a simple example of how you might put a microservice together and use it in an application. Of course, you need a lot of microservices to create a fully functional application, but this example is a good way to get started.

Creating a Microservice Setup for JavaScript

Before you can begin working with microservices, you need a setup that supports them. Of course, you have all sorts of options for creating a usable installation, but one path is a whole lot easier than the rest—using a combination of Node.js and Seneca.

You begin by installing a copy of Node.js on your system if you don't have one installed. The download you need for the installation is at https://nodejs.org/download/. It helps to have installation instructions. You can find instructions for a Mac install at http://blog.teamtreehouse.com/install-node-js-npm-mac, a Windows install at http://blog.teamtreehouse.com/install-node-js-npm-windows, and a Linux install at http://blog.teamtreehouse.com/install-node-js-npm-linux. Make sure the folder you use allows developer access, which means not using the `C:\Program Files` folder on Windows systems, for example. (Install the product to `C:\nodejs` on Windows systems if possible.) Make sure you run the suggested tests after installation to ensure your Node.js setup is working properly.

This book uses Seneca for microservices because it works well on Mac, Windows, and Linux systems. Of course, you can use any API gateway that suits your needs. Once you have Node.js installed, you can use the Node Package Manager (NPM) to install Seneca using the instructions found at http://senecajs.org/install.html. Make sure you are in the folder you plan to use to create your code when you install Seneca so that the Seneca files are in the right location. The downloadable source code includes the Seneca files for you.

Using NPM (https://www.npmjs.com/) makes it easy to install all sorts of packages for Node.js and reduces the complexity of creating applications of all sorts. It's important to note that the Seneca installation instructions show a $ prompt. When working with platforms other than Linux, you'll see another sort of prompt, but the command, `npm install Seneca`, is the same across all platforms. Interestingly enough, the prompt will simply display a busy indicator during the installation process—you won't see any sort of output that indicates anything is really happening until the end when you see the Seneca directory structure.

Understanding the Role of REST in Communication

Microservices rely on REST, which is an architectural style of communication, because it's lighter weight than protocols such as the Simple Object Access Protocol (SOAP). Using SOAP does have advantages in some situations, but it presents problems in Internet scenarios such as significant use of bandwidth and the need for a more formal level of communication between client and server. Applications that rely on REST for communication are called RESTful applications. Using REST for microservices provides the following advantages:

- Decouples consumers from producers
- Provides stateless communication
- Allows use of a cache
- Allows use of a layered system
- Provides a uniform interface

You have a number of options for using REST with microservices. However, the easiest method (and the method used for the example) is to rely on a specially formatted URL. For example, http://localhost:10101/act?say=hello is the URL used for the example. In this case, you contact the localhost using a special port, `10101`. You send a message using `act`. The message is interpreted as a JSON name/value pair, `{say:"hello"}`. The example demonstrates how this all works, but the idea is that you send a request and then get back a JSON response. Using REST for communication makes things simple.

Transmitting Data Using JSON

Microservices rely on JSON for transferring both requests and responses. Yes, you can also send data using REST, but the information ultimately ends up in JSON format. There are three main reasons that you want to use JSON to transfer data:

- Clean Data: The data format for JSON is straightforward. The data appears in two forms: name/value pairs or as a list of values. Because the data format is so strict and

simple, there is less chance for error when transferring data and therefore, fewer reliability and security issues.

- Efficiency: Because JSON avoids the whole tagged appearance of both HTML and XML, it tends to be smaller than other sorts of data transfers. The information is still in text form, but the format itself is quite efficient, which means you waste fewer resources transferring the data.
- Scalability: The strict data format used by JSON means that data transfers are standardized, which makes it easier to expand your application as needed. Using a single data structure means that you can plug in your code anywhere that you need it.

JSON typically uses five distinct data forms. Unlike XML, you don't create complex data hierarchies that can follow just about any form imaginable. Here are the five forms that you rely on to transmit data:

- Object: An object is a name/value pair. The pair appears within curly braces ({}) and is separated by a colon (:). You can create complex objects by separating several name/value pairs using a comma. For example, `{say:"hello"}` is a name/value pair.
- Array: An array one or more values contained within square brackets ([]). For example, `["One", "Two", "Three"]` is an array containing three string values.
- Value: A value is a single item. JSON recognizes string, number, object, array, `true`, `false`, and `null` as values.
- String: A series of characters within quotes (most texts say you should use double quotes). JSON recognizes control characters preceded by the backslash (\). These characters are: backspace (`\b`), formfeed (`\f`), newline (`\n`), carriage return (`\r`), and horizontal tab (`\t`). You can also specify Unicode character using \u and a four digit hexadecimal value. For example, `\u00BC` is the one quarter (¼) symbol.
- Number: A number is an unquoted series of numeric characters with or without a decimal point. The plus and minus signs show positive and negative values. You can also specify numbers using scientific notation by adding an e or an E. For example, `-123e20` is a perfectly acceptable presentation of a value.

Creating a Microservice Using Node.js and Seneca

You can find a number of examples for using Node.js and Seneca to create a microservice online. Unfortunately, most of them are convoluted and difficult to use. Some are simply outdated. The best example appears at http://senecajs.org/. The source for the server works precisely as shown. However, an even simpler example is the one found in `service.js` as shown here:

```
require('seneca')()
   .add(
      { say:"hello"},
      function( message, done )
       {
         done( null, {message:'hello'} )
         })
   .listen()
```

In this example, `require('seneca')` loads the Seneca library into memory. The code then adds a match pattern of `{ say:"hello"}` as a JSON object. The `function()` associated with the match pattern outputs another JSON object, `{message:'hello'}`. The example purposely uses both single and double quotes when creating JSON objects to show that it is possible, even if the official specifications don't seem to say so. The final step is to tell the service to `listen()`. You can add a port number to the `listen()` function. If you don't provide a port number, the service listens at the default port of 10101. To start the service, you type node server.js and press Enter at the command prompt. You see startup messages like the ones shown in Figure 8-1.

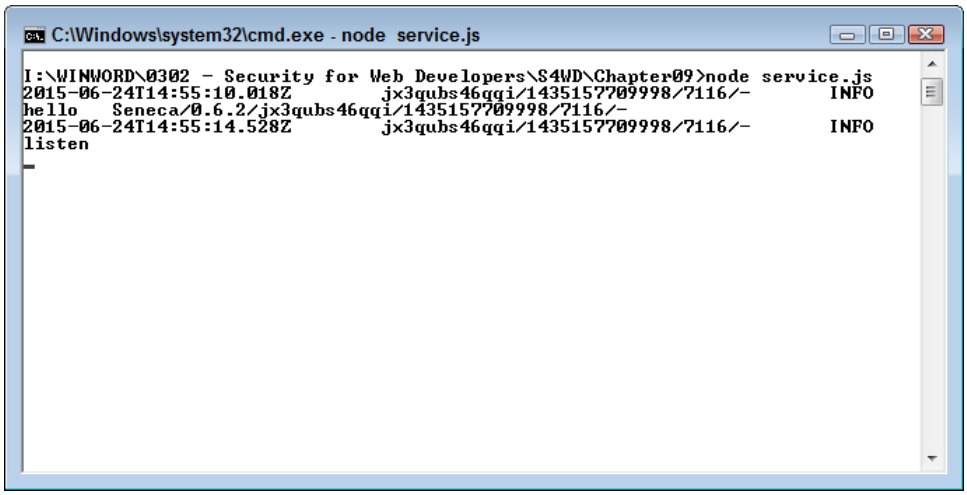


Figure 8-1. The microservice is listening for requests.

The startup process logs two steps. The first is the initialization process for Seneca (where Seneca says hello on the third line of the output in Figure 8-1). The second is placing the microservice in listen mode (as shown on the fifth line). Whenever the microservice makes a call or performs some other task (other than simple output), you see one or more log entries added to the window. From a security perspective, this makes it possible for you to track the microservice functionality and detect whether anyone is attempting to do something unwanted with it.

Of course, you'll want to test the microservice. Open your browser window and type http://localhost:10101/act?say=hello as an address. The microservice outputs a simple JSON object as shown in Figure 8-2.

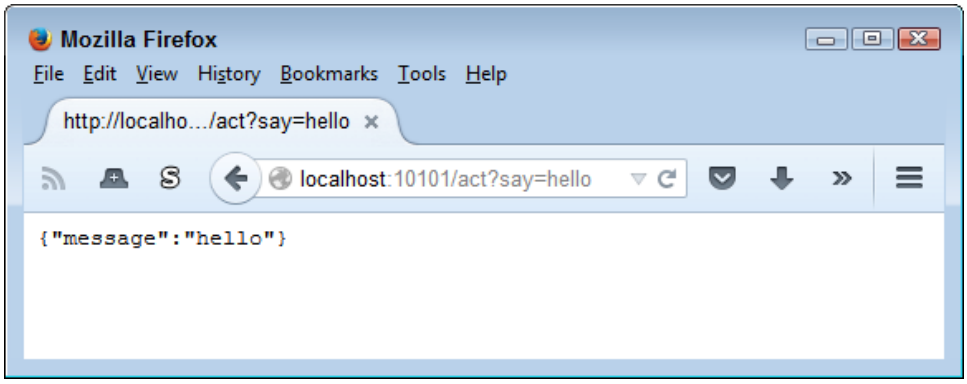


Figure 8-2. This simple example outputs a JSON object.

When you look back at the console window, you don't see anything. That's because the function output a simple JSON object and didn't make any calls outside the environment. However, try typing http://localhost:10101/act?say=goodbye as a request. Now you see some activity in the console window as shown in Figure 8-3.

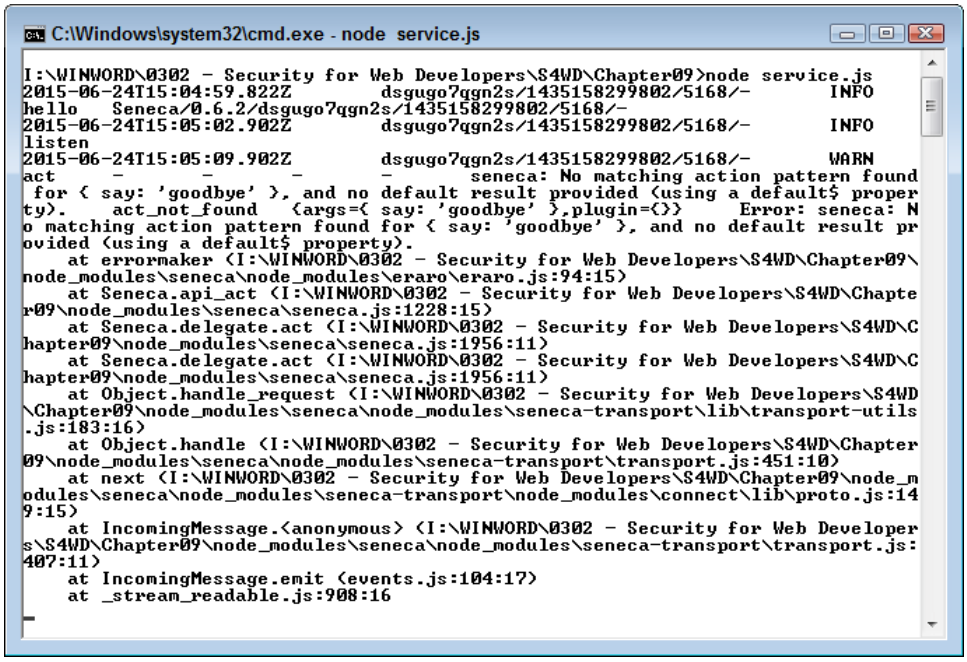


Figure 8-3. Errors produce copious output text.

The output includes a stack trace, which you can ignore in this case, but could prove helpful when working with complex microservices. The most important information appears at the top in this case. You see a warning that there is no matching pattern for `{ say: 'goodbye' }`. Notice that the REST request is translated into a JSON object. The error output tells you precisely what happened, so it's harder for someone to get by with an invalid request.

This is actually a good example for experimentation because you can see the results of trying to fool the REST communication part of the microservice functionality without

worrying about other elements covering up the results. When you finish working with the example, press Ctrl+C or Ctrl+Break to stop the service. The service will stop and you'll see the command prompt reappear.

Defining the Security Threats Posed by Microservices

In many respects, microservices mirror APIs when it comes to security concerns. For example, it's possible that a microservice could suffer from a man-in-the-middle attack. The ability of a hacker to truly benefit from such an attack is less than with an API because a microservice is small, self-contained, and only performs one task. However, the threat is still there and a hacker really only needs one avenue of attack to ruin your day. Even though you may see all sorts of articles telling you about the natural security that microservices provide, they do have security issues and you need to know about them too. The following sections provide a mix of benefits and problems that you need to consider when it comes to microservice security.

Lack of Consistency

The biggest potential threat posed by microservices is the lack of consistency that appears to haunt just about every library and API ever created. The library and API developers begin with the simple idea of creating an easy-to-use and consistent interface, but over time the library or API becomes a mishmash of conflicting strategies that makes a Gordian knot easy to untangle by comparison. Trying to fix either code base is incredibly difficult because developers use libraries and APIs as a single piece of code. These inconsistencies cause security issues because developers using the code bases think the calls should work one way when they really work another. The result is that a mismatch occurs between the code base and the application that relies on it. Hackers seize such errors as a means for gaining access to the application, its data, or the system it runs on.

Microservices can also suffer from a lack of consistency. It's essential that you create a template for describing precisely how to call microservices as early as possible in the development process. Just like libraries and APIs, hackers could use inconsistencies as a means for overcoming any security in have in place. Unlike libraries and APIs, the inconsistency would affect just one microservice, rather than the entire code base. Fixing the microservice would also prove easier because you're looking at just one call, rather than an entire API. Publishing a new microservice is also easier than publishing an entirely new library or API. Consequently, overcoming a microservice inconsistency is relatively easy.

Considering the Role of the Virtual Machine

Each microservice typically runs in its own Virtual Machine (VM) environment. This means that one microservice can't typically corrupt another. Even if a hacker does gain access to one microservice, the amount of damage the hacker can do is typically minimal. However, it's quite possible to run multiple microservices in the same virtual machine—at which point it would become possible for a hacker to try various techniques to obtain access to the entire API. To maximize security, you want to avoid stacking microservices

as shown by the `service2.js` example here (to access this example, you must use port 999, such as http://localhost:999/act?say=goodbye):

```
require('seneca')()
   .add(
      { say:"hello"},
      function( message, done )
       {
         done( null, {message:'hello'} )
         })
   .add(
      { say:"goodbye"},
      function( message, done )
       {
         done( null, {message:'goodbye'} )
         })
   .listen(999)
```

Best practice is to run each service in a separate virtual machine to ensure each service has its own address space and process. Separating each microservice presents fewer opportunities for error and also for issues resulting from code tricks hackers could employ.

Using JSON for Data Transfers

There are no perfect data transfer methodologies. Yes, JSON is lightweight, easy to use, and less susceptible to security issues than other technologies such as XML. However, hackers still have a number of methods for causing problems with JSON. The following sections describe the more prevalent issues.

Considering the Dangers of `eval()`

It's possible that someone could send a response to the client that contains a `<script>` tag with a script that could do just about anything. Fortunately, most browsers now detect such attempts and refuse to transfer the information. For example, try http://localhost:999/act?<script>alert('Hello');</script>=goodbye and you may see something like the output shown in Figure 8-4. In addition, the microservice itself refused to process the request. However, this particular script is simplistic in nature—a more advanced attempt could succeed.

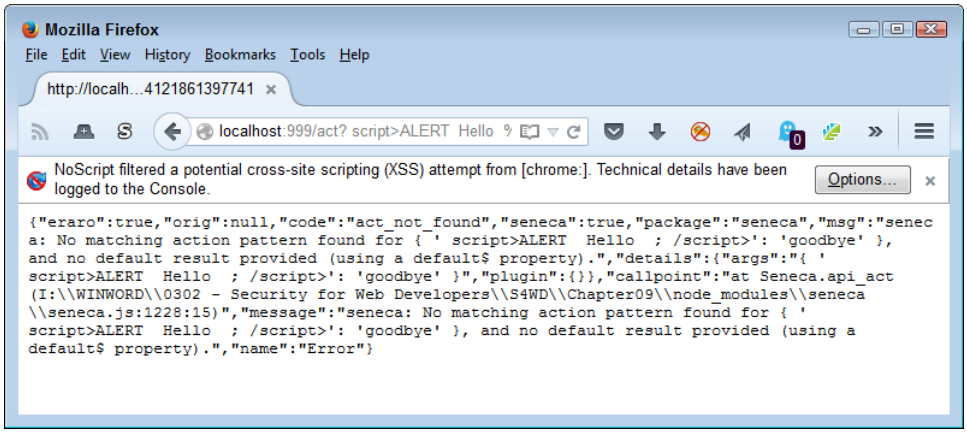


Figure 8-4. Scripting issues can plague microservices just as they do APIs.

Defending Against Cross-site Request Forgery

A Cross-site Request Forgery (CSRF or XSRF) is an attempt by an attacker to get a user to execute code unwittingly or unknowingly. The code executes at the user's privilege level and under the user's credentials, so it appears that the user has executed the code, rather than someone else. In most cases, this particular attack works against microservices when the following series of events occur:

1. A user logs into an application that relies on microservices.
2. The user performs various tasks, each of which relies on REST for communication in the background.
3. An attacker sends the user an URL that is specially formatted to look just like the other REST messages, but does something the attacker wants the user to do. The URL could appear as part of an e-mail message or some other communication, or even as a link on a web site.
4. The user initiates the request by sending the URL to the microservice just like normal.
5. The rest executes and the user may not even realize it has happened.

> A real world example of this exploit occurred in 2008 as a uTorrent exploit. The reason this particular exploit is so important to microservice developers is that it works on a system where a web server runs in the background. When the exploit occurs, it compromises the entire system, which is why you want to keep those microservices separated in their own VMs. You can read about it at http://xs-sniper.com/blog/2008/04/21/csrf-pwns-your-box/.

Because of the way in which this exploit works, what you really need to do is ensure that the application automatically logs the user out after a period of inactivity. In addition, you need to look for odd usage patterns and set limits on what a user can do without approval. For example, a user can transfer $1,000 at a bank, but transferring $10,000 requires a manager's approval.

Defining Transport Layer Security

The Achilles' heel of both microservices and APIs is that both function by sending messages back and forth, rather than performing processing on a single machine. The cornerstone of developing a fix for this issue is Transport Layer Security (TLS). It's essential to ensure the transport layer between client and server remains secure throughout the messaging process. This means using technologies such as HTTPS and REST to ensure that the communications remain as secure as possible.

An issue with wrapping absolutely every call and response in HTTPS and REST is that the application can slow to a crawl. The best method of overcoming this problem is to rely on load balancing to terminate client communications, while also keeping a channel open to backend processing needs. Keeping the backend processing channel open reduces overhead and helps reduce the effects of using HTTPS and REST.

> One of the issues with using HTTPS with public facing networks is that you must have a certificate from a Certificate Authority (CA)—an expensive proposition that may keep some organizations from using HTTPS. When you control both ends of the communication channel, it's possible to create your own certificate to achieve the same goal at a significantly reduced cost.

The use of HTTPS and bi-directional TLS ensures that both client and server establish each other's identity during each request/response cycle. Authentication reduces the chance that someone can successfully implement a man-in-the-middle attack to obtain unauthorized access to data. Most communication today takes place using unidirectional TLS where the client verifies the server's identity, but the server just assumes the client isn't compromised. Given the nature of microservice communication, you really do need to implement bi-directional TLS to verify the identity of both client and server.

Creating Alternate Microservice Paths

Something that many developers will have a problem understanding is the decentralized nature of microservices. Each microservice is separate. You don't have to think about the platform a microservice needs, what language it uses, or where it resides physically. It's possible to have two microservices written in two different languages residing on two different platforms in two different locations perform the same task. Because the environments used by the two microservices are so different, it's unlikely that an issue that affects one microservice will also affect the other microservice. Consequently, it's a good idea to keep both of them around so that you can switch between them as needed to keep your application running. That's what this section is all about—considering the implications of having multiple paths to access multiple microservices.

When thinking about microservices and the paths they employ, also consider things like ports. You can create microservices that work on different ports. Normally, you might rely on the microservice on port 999 to perform the work required by an application. However, if the microservice on port 999 becomes overloaded, compromised, or simply doesn't work, you can switch to the same microservice on a different port. Your code remains the same—only the port changes. Using this approach gives your application resilience, reliability, and flexibility. It also means that you have options should something like a Distributed Denial of Service (DDOS) attack occur.

III

Creating Useful and Efficient Testing Strategies

It isn't likely that you'll escape potential security problems with your application, but you can reduce them by ensuring you take time to think things through and perform proper testing techniques. This part of the book is all about helping you reduce the risk that someone will find your application inviting enough to try to break into it. Chapter 9 starts out by helping you think like a hacker, which is an important exercise if you actually want to see the security holes in your application. It isn't as if someone marks security holes in red with a big sign that says, "Fix Me!" so this thought process is incredibly important.

Chapter 10 discusses sandboxing techniques. Keeping code in a sandbox doesn't actually force it to behave, but it does reduce the damage that the code can do. Sandboxes make your system and its resources far less accessible and could make all the difference when it comes to security.

Chapters 11 and 12 discuss testing of various sorts. Chapter 11 focuses on in-house testing, which often helps locating major problems, but may not be enough to find subtle problems that could cost you later. Chapter 12 discusses third party testing, which is an especially important option for smaller businesses that may lack a security expert. No matter where you perform testing, ensuring you test fully is essential to knowing what the risks are of using a particular product with your application.

9

Thinking Like a Hacker

Most developers spend their time in a world where it's important to consider how things should work—will work when the code is correct. The whole idea of thinking about things as they shouldn't work—will break when the code is errant—is somewhat alien. Yes, developers deal with bugs all the time, but the line of thought is different. When you think like a hacker, you might actually use code that is perfectly acceptable as written—it may not have a bug, but it may have a security hole.

This chapter contains a process that helps you view code as a hacker would. You use tools to look for potential security holes, create a test system to use while attempting to break the code, and rely on common breaches to make your life a little easier. Hackers love the Bring Your Own Device (BYOD) phenomena because now you have all these unsecured systems floating about using operating systems that IT may not have much experience working with. Of course, there is always the ultimate application tester, the user. Users can find more ways to break applications than any developer would even want to think about, but user testing can be valuable in finding those assumptions you made that really weren't valid.

In fact, it's the need to think along these lines that drives many organizations to hire a security expert to think about all of the devious ways in which hackers will break perfectly functional applications in an effort to gain a tangible advantage they can use to perform specific tasks. However, this chapter assumes that you really don't have anyone wearing the security expert shoes in your organization right now. You may consider it when the application nears completion, but by then it's often too late to fix major problems without incurring huge costs. Thinking like a hacker when viewing your application can save your organization money, time, effort, and most importantly, embarrassment.

Defining a Need for Web Security Scans

The basic idea behind web security scans is that they tell you whether your site is currently clean and sometimes help you consider potential security holes. If you have a large setup, then buying a web security scanner is a good idea. However, smaller

businesses can often make use of one of the free web security scanners online. Of course, these products really aren't completely free. When the web security scanner does find a problem on your system, then the company that owns it will likely want you to engage it to clean your system up and protect it afterward. (There truly is no free lunch in the world of computing.)

> It isn't possible for any web security scanner to detect potential problems with your site with 100 percent accuracy. This is especially true of remote scanners—those that aren't actually running on a system that has the required access to the server you want to test. Scanners do a remarkable job of locating existing issues, but they can't find everything.
>
> You must also realize that web security scanners can produce false positive results. This means that the scanner could tell you that a security issue exists when it really doesn't.
>
> The best safeguard against either missing issues or false positives is to use more than one web security scanner. Counting on a single web security scanner really is a good start, but you'll likely end up working harder than needed to create a fully workable setup with the fewest possible security holes.

Before looking more at web security scanners, it's important to see what tasks it performs. In this case, the example will use Securi (https://sitecheck.sucuri.net/), which is a free scanner that can detect a number of security issues, mostly revolving around potential infections. The main page shown in Figure 9-1 lets you enter your site URL and click Scan Website! to begin the scanning process.

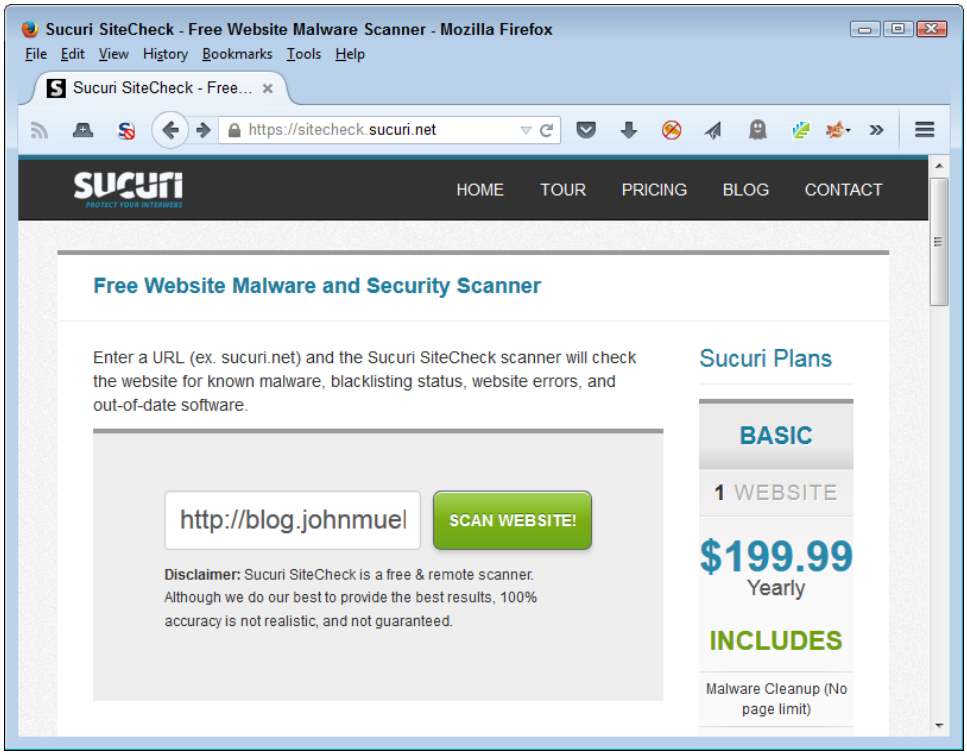


Figure 9-1. Starting the scan involves entering your site URL and clicking a button.

> If you do have a hosted site, make sure you check with the vendor providing the hosting services. Many of them offer low cost web security scanners. For example, you can find the offering for GoDaddy users at https://www.godaddy.com/security/malware-scanner.aspx. The vendor has a vested interest in ensuring your site isn't compromised, so the low cost offering benefits the vendor too.

After the scan is completed, you see an overview of what the scanner found and a more detailed listing of actual checks as shown in Figure 9-2. Of course, if you want a more detailed check, you can always buy one of the Securi plans listed on the right side of the page. However, for the purposes of seeing what a scanner can do for you, the free version works just fine.

Figure 9-2. The completed scan shows any problems with your site.

It can be interesting to view the site Website Details information. Figure 9-3 shows the information for my site. Interestingly enough, a hacker could employ a scanner just like this one to start looking for potential areas of interest on your site, such as which version of a particular piece of software you use. The web security scanner helps you understand the sorts of information that hackers have available. Seeing the kinds of information everyone can find is often a bit terrifying.

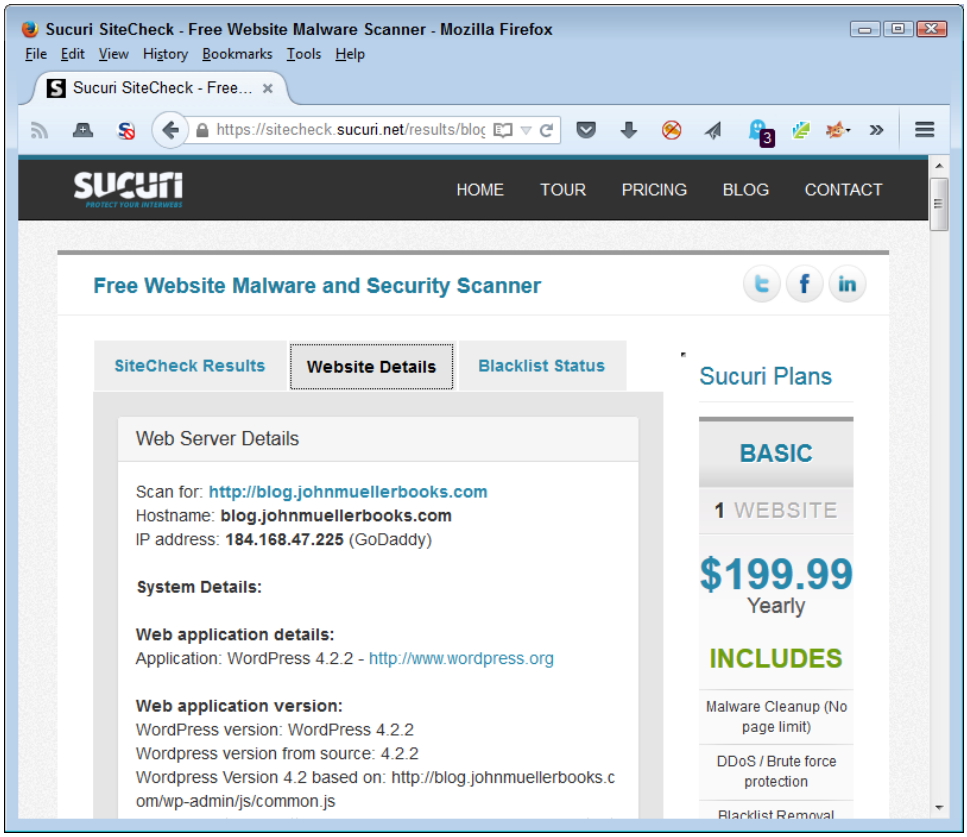


Figure 9-3. The detailed information can provide insights into what hackers can learn about your setup.

The detailed information can be helpful for another reason. When you use a hosted site, as many small businesses do, it's hard to know whether the vendor providing hosting services actually keeps the software updated. The Securi (and other) web security scanners probe the site carefully and come up with lists of version information you might not even see when using the dashboard for your site. For example, it's possible that even though a library still works fine, it's outdated and has a potential for creating a security breach on your system. A call to the hosting company can usually clear matters up quickly, but you need to know about the problem before you can call anyone about it.

> You can find lists of web security scanners online. One of the better lists is from Web Application Security Consortium (http://projects.webappsec.org/w/page/13246988/Web Application Security Scanner List). The authors have categorized the list by commercial, Software-As-A-Service (SAAS), and free open source tools. Another good list of tools appears on the Open Web Application Security Project (OWASP) site at https://www.owasp.org/index.php/Phoenix/Tools. You can find tools for all sorts of purposes on this site.

Building a Testing System

When thinking like a hacker, it's often a good idea to have a system that you don't really have to worry about trashing. In fact, it's more than likely that the system will get trashed somewhere along the way, so it's important to build such a system with ease-of-restoration in mind. The following sections provide you with a good overview of the things you should consider when building a test setup for your organization.

Considering the Test System Uses

You need a test system so that you can feel free to explore every possible avenue of hacking. Until you fully understand just how vulnerable most applications are today, you really can't fix your own application. Developers make all sorts of mistakes that seem innocent at the time, but really cause security issues later. Something as minor as the wrong sort of comment in your code can cause all kinds of problems. Using a production system to experiment is simply a bad idea because the techniques described in this chapter could potentially leave you open to a real attack. You want to use a system that you can trash and restore as needed to ensure the lessons you learn really do stick.

The idea is to test each application under adverse conditions, but in a safe manner that doesn't actually compromise any real data or any production systems. Giving a test system a virus will tell you how your application will react to that virus without actually experiencing the virus in a production environment. Of course, your test systems can't connect to your production network (or else the virus could get out and infect production systems). You can also use this environment to test out various exploits and determine just what it does take to break your application as a hacker would.

A test system need not test just your application, however. You can also use it to test the effectiveness of countermeasures or the vulnerabilities of specific configurations. Testing the system as a whole helps you ensure that your application isn't easily broken by vulnerabilities in other parts of the system setup. It's essential to test the whole environment because you can count on any hacker doing so.

Getting the Required Training

Before a developer can really test anything, it's important to know what to test and understand how to test it. You can begin the training process by using the OWASP Security Shepard (https://www.owasp.org/index.php/OWASP_Security_Shepherd), which illustrates the top ten security risks for applications. The Security Shepard can provide instruction to just one person or you can use it in a classroom environment to teach multiple students at once. The application provides a competitive environment, so in some respects it's a security game where the application keeps score of the outcome of each session. The application supports over 60 levels and you can configure it to take the user's learning curve and experience into account.

After you learn the basics, you need to spend some time actually breaking into software as a hacker would so that you can see the other side of the security environment. Of course, you want to perform this task in a safe and legal environment, which means not breaking into your own software. OWASP specifically designed the WebGoat application (https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project) to provide vulnerabilities that an application developer can use to discover how exploits work in a real application setting. The movies at http://webappsecmovies.sourceforge.net/webgoat/

take you step-by-step through the hacking process so that you better understand what hackers are doing to your application. The training includes movies on:

- General Principles
- Code Quality
- Concurrency
- Unvalidated Parameters
- Access Control Flaws
- Authentication Flaws
- Session Management Flaws
- Cross-Site Scripting (XSS)
- Buffer Overflows
- Injection Flaws
- Insecure Storage
- Denial of Service (DOS)
- Configuration
- Web Services
- AJAX Security

The final step is a WebGoat challenge where you demonstrate your new found skills by breaking the authentication scheme, stealing all the credit cards, and then defacing the site. The point is to fully understand hacking from the hackers viewpoint, but in a safe (and legal) environment.

It's important to understand how to perfect your security testing skills. The article at http://blogs.atlassian.com/2012/01/13-steps-to-learn-perfect-security-testing-in-your-org/ provides you with thirteen steps you can follow to improve your testing methods. This information, combined with the tools in this section, should help you test any application even if your organization lacks the resources to perform testing in depth.

Creating the Right Environment

Any environment you build must match the actual application environment. This means having access to each of the operating systems you plan to support, browsers the users plan to use, and countermeasures that each system will employ (which may mean nothing at all in some cases). Make sure the systems are easy to access because you may find yourself reconfiguring them to mimic the kinds of systems that users have. In addition, you must plan to rebuild the test systems regularly.

> Because of the kind of testing you perform using the test system, you must provide physical security for your lab. Otherwise, some well-meaning individual could let a virus or other nasty bit of code free in the production environment. You also don't want to allow access to the security setup to disgruntled employees or others who might use it to make your life interesting. The test environment should reflect your

production environment as closely as possible, but you also need to keep it physically separate or face the consequences.

As part of keeping production and test environments separate, you should consider using a different color cabling for the test environment. You don't want to connect a production system to the test environment accidentally or vice versa. In addition, it's essential to label test systems so no one uses them in the production environment.

Using Virtual Machines

Most organizations won't have enough hardware to run a single machine for each configuration of operating system and browser that users expect to rely on when working with the application. The solution is to use virtual machines so that you can configure one physical computer to host multiple virtual computers. Each of the virtual computers would represent a single user configuration for testing purposes.

Using virtual machines is a good idea for another reason. When the virtual system eventually succumbs to the attacks you make on it, you can simply stop that virtual machine, delete the file holding that computer, and create a new copy from a baseline configuration stored on the hard drive. Instead of hours to set up a new test system, you can create a new setup in just a few minutes.

Virtual machines can solve many problems with your setup, but you also need to consider the need for higher end hardware to make them work properly. An underpowered system won't produce the kind of results you need to evaluate security issues. The number of systems that you can create virtually on a physical machine is limited to the amount of memory, processing cycles, and other resources the system has to devote to the process.

It's also essential that the virtual machine operating software provide support for all the of the platforms you want to test. Products, such as VMWare (http://partnerweb.vmware.com/GOSIG/home.html) offer support for most major operating systems. Of course, this support comes at an additional cost.

As an alternative to going virtual, some organizations use older systems that are still viable, but not as useful as they once were in the production environment. Because of the way in which web applications work, these older systems usually provide all the computing horsepower needed and let an organization continue to receive some benefit from an older purchase. Of course, maintaining these old systems also incurs a cost, so you need to weigh the difference in cost between a virtual machine setup and the use of older systems (or perhaps create a combination of the two).

Getting the Tools

Unless you want to write your own security tool applications (a decidedly bad idea), you need to obtain them from someone else. Fortunately, sites such as McAfee (http://www.mcafee.com/us/downloads/free-tools/index.aspx) provide you with all the free tools you could ever want to perform tasks such as:

- Detect malware on the host system
- Assess whether the system is vulnerable to attack
- Perform forensic analysis after an attack

- Use the Foundstone Software Application Security Services (SASS) tools to make applications more secure
- Determine when an intrusion occurs
- Scan the system for various vulnerabilities
- Stress test the system

Configuring the System

Starting with a clean setup is important to conducting forensic examination of the system after attacking it. Before you install any software or perform any configuration, make sure you zero wipe it (write all zeroes to it). A number of software products lets you perform this task. Writing all zeroes to the hard drive ensures that any data you do see is the result of the testing you perform, rather than information left behind by a previous installation.

It's important to create a clean setup of each platform you intend to support at the outset. Make sure you make a copy of the setup image so that you can restore it later. The setup should include the operating system, browsers, and any test software needed to support the test environment. You may also want to include software that the user commonly installs on the system if the web application interacts with the software in any way.

> Products such as Norton Ghost (http://www.symantec.com/page.jsp?id=ghost) make it considerably easier to create images of each configuration. Make sure you have an image creation strategy in mind before you do anything to the clean configuration you create. You need clean images later to restore the configuration after you trash it by attacking it.

In addition to standard software, you may want to install remote access software so that you can access the system from a remote location outside the physically secured test area. External access must occur over a physically separate network to ensure there is no risk of contaminating the production environment. The use of remote access software lets more than one tester access the systems as needed from their usual workplace, rather than having to access the systems physically from within the secure area.

> Some organizations provide workstations that access the test systems using a KVM (keyboard, video, and mouse) switch. Using a KVM setup with a transmitter lets you easily switch between test systems from a remote location. However, the use of remote access software is probably more flexible and faster.

Restoring the System

It won't take long and your test system will require restoration. The viruses, adware, and Trojans that you use to test it are only one source of problems. Performing exploits and determining how easy it is to break your application will eventually damage the operating system, test application, test data, and the countermeasures used to protect the system. In short, you need some method of restoring the system to a known state quite quickly.

Restoration also includes reconfiguring the system to mimic another environment. It pays to use images to create various test environments quickly. As previously mentioned,

using virtual machines saves a lot of time because you don't have to rebuild the hard drive from scratch each time. However, make sure you also have each operating system image on a separate hard drive, DVD, or other storage media because the hard drive will eventually become corrupted.

Defining the Most Common Breach Sources

Every day sees the addition of new security breaches. It's not likely that anyone could keep up with them all. However, some security breaches require special attention and others are indicative of the sorts of things you see in the future. Of course, it's nice to have an organized method for locating the security beaches and the checklist provided by OWASP at https://www.owasp.org/index.php/Web_Application_Security_Testing_Cheat_Sheet is a start in the right direction.

Once you know about the potential attack vectors for your application, it helps to score them so that you know which vulnerabilities to fix first. An attack that lets someone access all the pictures of the company picnic is far lower priority than one that allows access to the credit card information of all your clients. One of the best systems you can use for scoring potential attack vectors is Common Vulnerability Scoring System (CVSS) (http://www.first.org/cvss/v2/faq). Using this system helps you create an organized list of problems that you can deal with to make your application more secure.

Of course, it also helps you know what to test. With this in mind, the following sections describe the most common breach sources as of the writing of this book. (These sections build on the information you already obtain about basic hacks from Chapter 1.) You'll likely find even more breaches by the time the book is released because hackers are nothing if not creative in their methods of breaking software.

> You can't categorize every potential breach source hackers will use. In fact, hackers often rely on misdirection (much like magicians) to keep you from figuring out what's going on. A recent news story about LOT Polish airlines serves to point out the results of misdirection (see http://www.computerworld.com/article/2938486/security/cyberattack-grounds-planes-in-poland.html). In this case, authorities have spent considerable time and resources ensuring that flight systems remain unaffected by potential attacks. However, they weren't as diligent about ground systems. Hackers managed to break into the ground systems and make it impossible to create flight plans for outbound passengers. Whether the attack grounded the planes by affecting their flight systems or by affecting ground control doesn't matter. What matters is that the planes couldn't take off. The hackers achieved a desired result through misdirection. The lesson is that you need to look everywhere—not just where you think hackers will attack based on the latest statistics.

Avoiding SQL Injection Attacks

There are many forms of the SQL injection attack. For example, you could mess with the form data in an application to determine whether the form is capable of sending

commands to the backend server. However, it's best to start with a popular method of checking for the potential for a SQL injection attack.

Let's say that you have an URL such as http://www.mysite.com/index.php?itemid=10. You have probably see URLs structured like this one on many sites. One way to check for a vulnerability is to simply add a single quite after the URL, making it http://www.mysite.com/index.php?itemid=10'. When you press Enter, the site sends back a SQL error message. The message varies by system, but the idea is that a backend server receives your URL as a SQL request that's formatted something like: `SELECT * WHERE itemid='10''`. The addition of another single quote makes the query invalid, which produces the error.

> When you do find a problem with your application that someone could potentially exploit, it's important to make the entire development team aware of it. Demonstrating the exploit so that everyone can see how it works and what you need to do to fix it is an essential part of developer training for any organization. Products such as the Browser Exploitation Framework (BeEF) (http://beefproject.com/) can help you find the attack vector and then demonstrate it to others in your organization.

You can now start playing with the URL to see what's possible. For example, let's say you want to determine how many columns that query produces and what those columns are. It's possible to use the SQL `ORDER BY` clause to perform this task. Change the URL so that it includes the `ORDER BY` clause like this: http://www.mysite.com/index.php?itemid=10 ORDER BY 1. This is the same as typing `SELECT * WHERE itemid='10' ORDER BY 1` as a command. By increasing the ORDER BY value by 1 for each request, you eventually see an error again. Say you see the error when you try `ORDER BY 10`. The query results actually have 9 columns in this case.

A hacker will continue to add SQL commands to the basic URL to learn more about the query results from the database. For example, using the `SELECT` clause helps determine which data appears on screen. You can also request special information as part of the `SELECT` clause, such as `@@version`, to obtain the version number of the SQL server (giving you some idea of what vulnerabilities the SQL server might have). The point is that the original URL provides direct access to the SQL server, making it possible for someone to take the SQL server over without doing much work at all.

You can see another type of SQL injection attack dissected at http://www.w3schools.com/sql/sql_injection.asp. The underlying cause of all these attacks is that a developer used data directly from a page or request without first checking it for issues and potentially removing errant information from it.

Understanding Cross-Site Scripting

XSS is similar to SQL injection in many ways because of the way in which the exploit occurs. However, the actual technique differs. The two types of XSS are non-persistent (where the exploit relies on the user visiting a specially crafted link) and persistent (where the attack code is stored in secondary storage, such as a database). Both attacks rely on JavaScript code put into a place where you wouldn't expect.

As an example of a non-persistent form of XSS, consider this link: http://www.mysite.com/index.php?name=guest. It looks like a perfectly harmless link with a name/value entry added to it. However, if you add a script to it, such as http://www.mysite.com/index.php?name=guest<script>alert('XSS')</script>, the user could see a dialog box pop up with the message, XSS. This example doesn't do any damage, but the script could easily do anything you can do in a script. For example, you could craft the script in such a manner that it actually redirects the user to another site where the page would download a virus or other nasty piece of software.

Production: Please do not change the % shown below to percent. –John M

> The example shows the `<script>` tag in plain text. A real exploit would encode the `<script>` tag so that the user couldn't easily recognize it. What the user would see is a long URL with an overly complex set of % values that appear regularly in valid URLs as well.

A persistent XSS attack is harder to implement, but also does a lot more damage. For example, consider what happens when a user logs into an application. The server sends a session ID as a cookie back to the user's machine. Every request after that uses the session ID so that the application can maintain state information about the user. Let's say that the attacker sends a specially crafted script to the server as part of the login process that gets stored into the database under a name that the administrator is almost certain to want to review. When the administrator clicks on the user name, the script executes and sends the administrator's session ID to the attacker. The attacker now has administrator privileges for the rest of the session, making it possible to perform any administrator level task. You can get more detailed information about persistent XSS at https://www.acunetix.com/blog/articles/persistent-cross-site-scripting/.

In both cases, the best defense against XSS is to sanitize any input you receive. The process of sanitizing the input removes any scripts, weird characters, or other information that isn't part of the expected response to a request. For example, if you expect a numeric input, the response shouldn't contain alphabetic characters.

Tackling Denial of Service Issues

The idea behind a DOS attack is relatively straightforward. You find an open port on a server and keep sending nonsense packets to it in an effort to overwhelm the associated service. Most servers have services that offer open ports, such as:

- DNS servers
- E-mail servers
- FTP servers
- Telnet servers
- Web servers

Of course, the more open ports you provide, the better the chance of overwhelming your server. A first line of defense is to close ports that you don't need by not installing services you don't require. An application server may only require a Web server, so that's the only service you should have installed. As an application developer, you can recommend keeping other services uninstalled (or at least inactive). When creating a

private application, using a non-standard port can also help, as does requiring authentication.

Hackers are usually looking for services that don't have a maximum number of connections, so ensuring you keep the maximum number of connections to a value that your server can handle is another step in the right direction. It's important in a DOS attack to give the server something to do, such as perform a complex search when working with a Web server. Authentication would help keep a hacker from making requests without proper authorization.

Still, it's possible to bring down any server if you have enough systems sending an endless stream of worthless requests. Some of the defenses against DOS attacks include looking for patterns in the request and then simply deny them, rather than expend system resources trying to resolve the request. You can find a host of DOS attack tools to use to test your system at http://resources.infosecinstitute.com/dos-attacks-free-dos-attacking-tools/. Beside looking for patterns in the attack and attempting to resolve them yourself, you can also try:

- Purchase specialized equipment designed to help mitigate DOS attacks
- Rely on your ISP to detect and mitigate DOS attacks
- Obtain the services of a cloud mitigation provider

Nipping Predictable Resource Location

It's possible to attack a system by knowing the location of specific resources and then using those resources to gain enough information to access the system. Some sites also call this type of attack as forced browsing. Of course, the best way to prevent this sort of an attack is to keep resources in unpredictable locations. In addition, you can ensure that the authentication scheme works and that you properly secure resources. However, let's look at how this particular attack works.

One example of this sort of attack is where a URL points out a valid resource and then you use that URL to access another resource owned by someone else. Let's say that you have your agenda for a particular day located at http://www.mysite.com/Sam/10/01/2015 and that you want to access Amy's agenda for the same day. Change the URL to http://www.mysite.com/Amy/10/01/2015 might provide the required access if the administrator hasn't configured the server's authentication correctly.

As another example, some servers place information in specific directories. For example, you might have authorized access to http://www.mysite.com/myapp/app.html. However, you could change the URL to see if http://www.mysite.com/system/ exists. If you get a response of 200 back from the server, the directory does exist and you can start querying it for useful information. Of course, this assumes that the administrator hasn't properly secured the system directory and that system is a standard directory location. The administrator could always change the name of system directory and also ensure that it only allows access by those with the proper credentials.

Overcoming Unintentional Information Disclosure

Unintentional information disclosure can occur in all sorts of ways. However, the exploit always involves the hacker gaining unauthorized access to a centralized database. At one time, the source of the information would have been something like the Network

Information System (NIS). However, today the information could come from any source that isn't secure or has vulnerabilities that a hacker can exploit. There are so many of these sources that it's not really possible to come up with a simple example that illustrates them all. You can overcome this type of hack by:

- Applying all required patches to the operating system, services, application environment, libraries, APIs, and microservices as required
- Configure the border routers and firewalls to block requests that could request information from a sensitive source
- Restrict access to all sensitive information sources
- Never hard code passwords or place them where someone could easily find them
- Use two-factor authentication for any sensitive information source
- Perform audits to look for potential breaches (even if you feel the system is completely secure)
- Use assessment tools to determine whether it's possible to access the sensitive information source from anywhere other than a designated location

Testing in a BYOD Environment

The BYOD phenomenon keeps building in strength. It's important to realize that BYOD isn't going to go away. In fact, you probably already know that BYOD is going to become the preferred method of outfitting users at some point. Organizations will eventually tell users to bring whatever device is needed to get their work done and leave everything in the hands of the user. It sounds like a disaster in the making, but that's where users are taking things now.

According to Gartner, Inc., by 2017 half of organizations will no longer supply any devices to users (see http://www.gartner.com/newsroom/id/2466615). In addition, by 2020 75 percent of users will pay less than $100 for a smartphone (see http://www.gartner.com/newsroom/id/2939217), so getting a device smart enough to perform most tasks won't even be that expensive. Creating and managing applications will become harder because you must ensure that the application really does work everywhere and on any device. Of course, it has to work without compromising organizational security. The following sections will help you provide some level of testing and isolation for the BYOD environment.

> It's also important to realize that users are now participating strongly in Bring Your Own Application (BYOA). The reason this new move on the part of users is so important is that it introduces yet another level of unpredictability to the application environment. You never know when another application will cause your application woe. In fact, the third party application could provide the conduit for the next major breach your company suffers. Users will definitely continue using applications such as Dropbox, Google Docs, and CloudOn because they're convenient and run everywhere. To ensure the integrity of the application environment, you need to continue viewing these applications as contamination just waiting to ruin your day.

One of the reasons that BYOA is such a problem is that the organization loses control over its data. If the user stores organizational data in a personal account on Dropbox, the organization can't easily retrieve that data in the event of an emergency. In short, BYOA opens serious security holes that could be a problem for any application data that you want to protect.

Configuring a Remote Access Zone

When working within the BYOD environment, the best assumption you can make is that the device environment isn't secure and that you can't easily make it secure. With this in mind, a BYOD session usually has four phases:

1. The client and server create a secure tunnel to make it harder for outsiders to listen in. The intent is to prevent man-in-the-middle attacks.
2. The user supplies two forms of authentication. A two-factor authentication process makes it less likely that someone will successfully mimic the user.
3. The client makes one or more requests that the server mediates using a service mediation module. The service mediation module only honors requests for valid services. The service mediation module automatically logs every request, successful or not.
4. A service separation module provides access to public data only. It disallows access to sensitive data. The client sees just the data that the organization deems acceptable for a BYOD environment.

The remote access zone is part of phases 1 and 2. It consists of an external firewall and a VPN and authentication gateway. The remote access zone provides a first level of defense against intrusion.

The information gleaned from the user must appear as part of your application strategy. A BYOD access is different from local access from a desktop system the organization owns in that you have no idea of where this access occurs or whether the device itself is secure. When a user accesses your application in such a manner, you need to provide a role that matches the device used. This means that you don't allow any sensitive information to appear on the device and could potentially limit access to other sorts of data. For example, your organization might decide that it's acceptable to obtain a listing of sales for a particular client, but the list is read-only, which means that the user can't make any changes to the sales list in the field. In order to make these changes, the user would need to log into the account from a local system.

The use of a remote access zone also implies that your organization configures Mobile Device Management (MDM). This is a set of products and services that help ensure the mobile device remains as secure as is possible. For example, the MDM could check mobile devices for patch requirements and ensure the user patches the device before you allow application access (you could simply patch the device automatically). Even with an MDM in place, you can't assume the device is secure when:

- The device reports values that imply a secure configuration, but your organization doesn't actually implement the specified values. A malware developer won't know which values to report and will simply report them all.

- Other authorities can override the device settings. For example, if a smartphone vendor can automatically patch the device outside your control, you must assume that some of those patches could contain viruses or other malware.
- It isn't possible to enforce settings on the mobile device. The device may not provide the required support, your application may not be configured to support the device, or malware interferes with the updates.

Checking for Cross Application Hacks

A cross application hack is one in which one application gains access to data or other resources used by another application. In many cases, the hack occurs when two applications have access to the same data source (cross application resource access or XARA). One such recent hack is for the OS X and iOS operating systems (read about it at http://oversitesentry.com/xara-an-old-way-to-hack-cross-application-resource-access/).

Another type of cross application problem is Cross Application Scripting (CAS). In this case, the bug causes a problem where JavaScript code executes within certain types of applications. You can read about one such exploit for Android at http://news.softpedia.com/news/Apache-Cordova-for-Android-Patched-Against-Cross-Application-Scripting-453942.shtml.

The best way to verify that your application isn't potentially exposing data through this kind of exploit is to stay on top of any news for the platforms you support. It takes a security expert to find potential problems of this sort. Unfortunately, barring a patch from an operating system or browser vendor, you can't really do too much about this particular hack except to remain vigilant in checking your data stores for potential damage. Of course, this particular kind of problem just lends more credence to creating a remote access zone (see the previous section of the chapter).

Dealing with Really Ancient Equipment and Software

Research firms love to paint pretty pictures of precisely how the future will look. Of course, some of those dreams really do come true, but they don't take into account the realities of the range of user equipment in use. For example, it might trouble some people to discover that (as of this writing) 95 percent of all ATMs still rely on Windows XP as an operating system (see http://info.rippleshot.com/blog/windows-xp-still-running-95-percent-atms-world). The US Navy is still using Windows XP on 100,000 desktops (see http://arstechnica.com/information-technology/2015/06/navy-re-ups-with-microsoft-for-more-windows-xp-support/). Yes, old, archaic, creaky software still exists out there and you might find it used to run your application.

According to NetMarketShare (http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0) Windows XP still powers upwards of 14.6 percent of the systems out there. It's important to realize that your main concern when creating an application may not be the shiny new smartphone with the updated, fully patched operating system and the latest in browser technology. The real point of concern may be that creaky old Windows XP system loaded with Internet Explorer 8 that your users insist on using. It's interesting to note that Internet Explorer 8 still commands about 25 percent of the desktop market share (see https://www.netmarketshare.com/browser-market-share.aspx?qprid=2&qpcustomd=0).

Of course, you can always attempt to force users to upgrade. However, if you've dealt with the whole BYOD phenomena for long enough, you know that users will simply ignore you. Yes, they might be able to show you a shiny new system, but they'll continue to use the system they like—the older clunker that's causing your application major woe.

About the only effective means you have of dealing with outdated equipment is to check the browser data during requests. Doing so lets you choose whether to allow the request to succeed. When the browser is too old, you can simply display a message telling the user to upgrade their equipment. The article at http://sixrevisions.com/javascript/browser-detection-javascript/ describes how to perform browser version checking. You can see your actual browser information at https://www.cyscape.com/showbrow.asp. The only problem with this approach is that it's possible to thwart the check in some cases and some browsers also report incorrect version information.

Relying on User Testing

Nothing can test software in a more violent and unpredictable manner than a user. Most developers have seen users try to make software do things that it quite obviously shouldn't because the user doesn't realize the software shouldn't do that. It's this disconnect between what the software appears to do and what the user makes it do that provides the serious testing that only a user can provide.

The important part of the process is the part that will cause the most pain to developers. Any attempt to create an organized approach to user testing will only result in testing failures. Users need to have the freedom to play with the software. Actually, play time is good for everyone, but it's an essential element of user testing. The following sections describe how you can get the user to play with your software and come up with those serious security deficiencies that only users (and apparently some hackers) seem to find.

> Interestingly enough, you don't actually have to perform your own user testing any longer. If you really want a broad base of test users, but don't have the resources to do it yourself, you can always rely on sites such as User Testing (http://www.usertesting.com/). Theoretically, the site will provide you with honest evaluations of your application in as little as an hour (although one quote provided on the main page said the results were delivered in 20 minutes). Most of these third party testing sites offer web application, mobile application, and prototype testing.
>
> A few third party testers, such as Applause (http://www.applause.com/web-app-testing), specifically offer "in the wild" testing where your application actually sees use on the Internet from unknown users, just as it would in the production environment. Of course, the difference is that that testing occurs without any risk to your equipment and Applause provides the tools required to obtain measurable results. In general, you would want to save this level of testing for a time when your application is almost ready for release and you want to perform a sanity check.

Letting the User Run Amok

Sometimes the best way to test your application is to give your user a set of printed steps and a goal. Watching how the user interacts with the application can tell you a lot about how the user perceives the application and where you need to make changes. Actually videotaping the user at work can be helpful, but you need to be aware that the act of videotaping will change the user's behavior. Keylogging and other techniques are also helpful in keeping track of what the user does with the application without actually standing over the top of the user to observe (which would definitely change user behavior and your testing would fail).

Fortunately, you don't have to rely on just the resources you have at hand. Sites such as Mashable (http://mashable.com/2011/09/30/website-usability-tools/) provide you with a wealth of testing tools you can use to check your web application for problems. The site documents the tools well and tells you why each tool is important. Most importantly, the site helps you understand the importance of specific kinds of testing that you might not have considered. For example, Check My Colours (http://www.checkmycolours.com/) verifies that people with various visual needs can actually see your site. Yes, using the wrong colors really can be a problem and testing for that issue can help you avoid potential security problems caused by user error.

Another good place to find testing tools is The Daily Egg (http://blog.crazyegg.com/2013/08/08/web-usability-tools/). Some of the tools on this site are the same as those on the Mashable site, but you obtain additional insights about them. A few of the tools are unique. For example, the list of page speed testing tools is better on The Daily Egg and the list includes GTMetrix (http://gtmetrix.com/), which can help you locate the actual source of slowdowns on your page.

Developing Reproducible Steps

Part of user testing is to obtain a set of reproducible steps. Unless you gain this kind of information, you can't really fix application issues that cause problems. This is the reason that you need specific testing software that records what the user does in detail. Asking the user to reproduce the steps that led to an error will never work. In many cases, the user has no idea of how the error occurred and simply feels that the computer doesn't like them. Trying to get the user to reproduce the steps later will likely lead to frustration and cause the user to dislike the application (which usually leads to more errors and more security problems). Therefore, you need to get the steps required to reproduce an error on the first try.

In order to create an environment where you can obtain the best level of testing and also ensure that the user has a good chance of finding potential errors, you need to perform specific kinds of testing. Letting the user run amok to see what they can do without any sort of input is useful, but the chaos hardly produces predictable results. When working through user testing, you need to consider these kinds of tests:

- Functionality: It's important to test all of the application features. This means asking the users to try out forms, perform file manipulation and calculation tasks, search for information using application features, and try out any media features your application provides. As the user tests these various features, make sure that the tests also check out the libraries, APIs, and microservices that your application relies upon to perform most tasks.

- User Interface and Usability: The user interface must keep the user engaged and provide support for anyone with special needs. As part of this level of testing, you need to check navigation, accessibility, usefulness from multiple browser types, error messages and warnings, help and any other documentation you provide, and layouts.
- Security: Even though you have tested the application to determine whether it suffers from any of the common security breaches listed in the "Defining the Most Common Breach Sources" section of this chapter, you need to have the user test for them as well. See if the user can get the application to break in the same ways that you did. Look for ways in which the user's method of interacting with the application creates new breach conditions.
- Load and Scalability: It's impossible for you to test an application fully to determine how it acts under load. You need to ensure that the application scales well and that its performance degrades gracefully as load increases. However, most importantly, you need to verify that load doesn't cause issues where a security breach can occur. It's important to know that the application will continue to work properly no matter how much load you apply to it. Of course, the application will run more slowly when the load exceeds expectations, but that's not the same as actually breaking—causing a failure that could let someone in.

Giving the User a Voice

Interviewing a user or allowing users to discuss the application in a gripe session after working with it is a good way to discover more potential problems with the application. In many cases, a user will be afraid to perform certain steps due to the perception that the application will break. This problem occurs even in a test environment where the user is supposed to break the application. You may not want to hear what the users have to say, but it's better to hear it during the development stage than to put the application into production and find out that it has major issues later.

As an alternative to confrontational approaches to obtaining user input, you can also rely on surveys. An anonymous survey could help you obtain information that the user might otherwise shy away from providing. It's important to consider the effects of stress and perceived risk on the quality of user input you receive.

Using Outside Security Testers

Penetration testing relies on the services of a third party to determine whether an application, site, or even an entire organization is susceptible to various kinds of intrusion. The attacker probes defenses using the same techniques that a hacker uses. Because the attacker is a security professional, the level of testing is likely better than what an organization can provide on its own. Most organizations use outside security testing services for these reasons:

- Locating the vulnerabilities missed by in-house audits
- Providing customers and other third parties with an independent audit of the security included with an application
- Ensuring the security testing is complete because the organization doesn't have any security professionals on staff
- Validating the effectiveness of incident management procedures

- Training the incident handling team
- Reducing security costs for the organization as a whole

> It's a bad idea to allow someone to penetration test your application, site, or organization without having legal paperwork in place, such as a Non-Disclosure Agreement (NDA). Don't assume that you can trust anyone, especially not someone who is purposely testing the security of your setup. Make sure you have everything in writing and that there is no chance of misunderstanding from the outset.

Considering the Penetration Testing Company

Of course, like anything, penetration testing comes with some risks. For example, it's likely that you expect the third party tester to be completely honest with you about the vulnerabilities found in your application, but this often isn't the case. In some cases, the third party fails to document the vulnerabilities completely, but in other cases, the penetration tester might actually be sizing your company up to determine whether a real intrusion would be a good idea. It's important to ensure you deal with a reputable security tester and verify the types of services rendered in advance. You should spend more time checking your security services if you experience:

- Disclosure, abuse, or loss of sensitive information obtained during a penetration test
- Missed vulnerabilities or weaknesses
- Availability of the target application or site is impacted
- Testing doesn't occur in a timely manner
- Reporting is overly technical and hard to comprehend
- Project management looks disorganized, rushed, or ill-considered

When it comes to penetration testing, you tend to get what you pay for. Your organization must consider the tradeoffs between cost, timeliness, and quality of testing when hiring a third party to perform penetration testing. The less you pay, the more you tend to wait, the less you get for your money, and the more likely it is that something negative will happen as a result of the testing.

In some cases, a tradeoff in amount of testing works better than other tradeoffs do. Testing just one application, rather than the entire site, will cost less and you'll be able to hire a higher quality security firm to perform the task. Of course, if you test just one application, you can't check for issues such as trust relationships between the target and another systems.

Managing the Project

Before you allow anyone to perform penetration testing, you need a proposal that outlines the scope, objectives, and methods of testing. Any penetration testing must include social engineering attacks because most hackers employ them. In fact, the people in your organization are the weakest link in your security. An application can provide nearly perfect security, but a single disclosure by the wrong person can thwart all your attempts at maintaining a secure environment.

Ensure that the testing methodology is well-documented and adheres to industry best practices. For example, many security firms adhere to the Open Source Security Testing Methodology (OSSTMM) (check out the Institute for Security and Open Methodologies, ISECOM, site for details at http://www.isecom.org/). If the tester uses some other methodology, make sure you understand precisely what that methodology is and what sorts of benefits it provides.

The proposal should state what type of feedback you receive as part of the testing. For example, it's important to decide whether the penetration testing includes full disclosure with a demonstration of precisely how the system is penetrated, or does the report simply indicate that a particular area is vulnerable to attack.

Defining the time of the attack is also important. You might not want to allow testing during the busiest time of the day to avoid risking sales. On the other hand, you may need to allow testing at less convenient times if you truly want to see the effects of load on the system.

Covering the Essentials

Any organization you engage for penetration testing should have a single point of contact. It's important that you be able to reach this contact at any time. The leader of your development team and the penetration team contact should be able to contact each other at any time to stop testing should the need arise. The contact should also have full details on precisely how the testing is proceeding and specifics about any potential issues that might occur during the current testing phase.

You should also know whether the tester has liability insurance to cover any losses incurred during testing. The liability insurance should include repayment of time invested in recovering data, down time for the system, and any potential loss of revenue that your organization might incur.

The testing team must also demonstrate competency. You don't want just anyone penetration testing your system. The team that appears in the proposal should also be the team that does the actual testing. Look for certifications such as:

- GIAC Certified Incident Handler (GCIH)
- Certified Ethical Hacker (CEH)
- OSSTMM Professional Security Tester (OPST)

Getting the Report

The results of any penetration testing appear as a report. To ensure you get a report that you can actually use, make sure you request an example report before testing begins. The report should include a summary of the testing, details of any vulnerabilities, a risk assessment, and details of all the actions the penetration testing involves. The report must contain enough information that you can actually fix the problems found during testing, but still be readable by management staff so you get the time and resources required to perform the fixes.

Along with the report, you should also receive log files of every action taken during the testing phase. The log files should show every packet sent and received during testing so that you can go back later to follow the testing process step-by-step.

The security company you hire should also keep an archive of the testing process. You need to know how they plan to maintain this archive. Part of the reason for checking into the archive process is to ensure your confidential data remains confidential. It’s important that the archive appear as part of off-line storage, rather than a network drive on the vendor’s system.