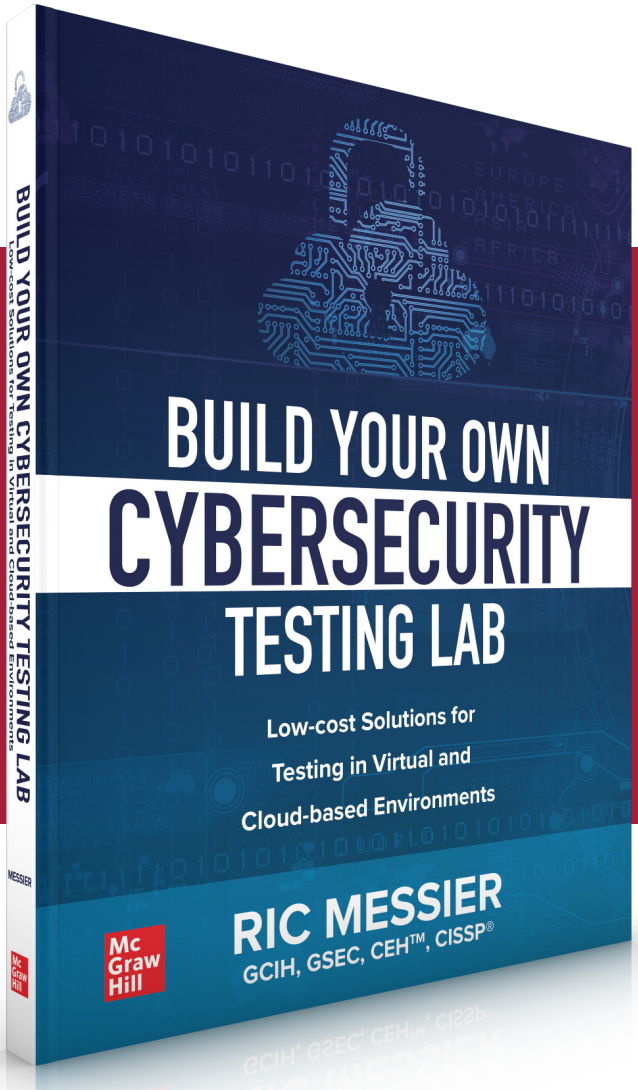
Mc Graw Hill

BUILD YOUR OWN CYBERSECURITY TESTING LAB

Sample Chapter

CHAPTER 1:
Why Perform Security Testing?



LEARN MORE

BUY NOW

Because learning changes everything.®

CHAPTER 1

Why Perform Security Testing?

In this chapter, we will cover:

- Compliance
- Types of security testing
- Identifying goals

People who hear security testing may well think it means the same thing as penetration testing. Since I bring it up, it shouldn't come as a big surprise that they are not the same thing. In fact, penetration testing is a small and potentially insignificant element of the broader field of security testing. There are number of ways of approaching security testing, and the approach should, ideally, be guided by what the overall objectives you have and what your attitude is toward security testing. Security testing is a very broad idea that we'll talk about in more detail later on. In short, security testing should encompass the entire life cycle of software or systems from requirements through deployment and maintenance. Penetration testing is focused on post-deployment.

The reason security testing is essential is because more and more we have adversaries who are organized, funded, and persistent. We are no longer facing the so-called script kiddie as a primary adversary—the nuisance that you can protect against through common defense-in-depth strategies like firewalls. We need to rethink how we address security. The company FireEye, through its Mandiant Consulting division, has been tracking a statistic they call dwell time. This is the number of days these adversaries are staying in networks before they have been detected. Figure 1-1 shows the trend of this dwell time over the last several years since this statistic was tracked. That number has been coming down, but it's still more than two months. That's the median number, by the way. That means half of the businesses that were investigated had dwell times longer than that.

Without people doing testing—across the board—we continue to open the door to these attackers, allowing them to remain in our systems and networks, stealing our data, using our resources to perpetrate additional attacks, or just making money off corporate resources. This is why we test. Because we need to identify and close holes and vulnerabilities as best as we can in the hope of at least making life for our adversaries harder.

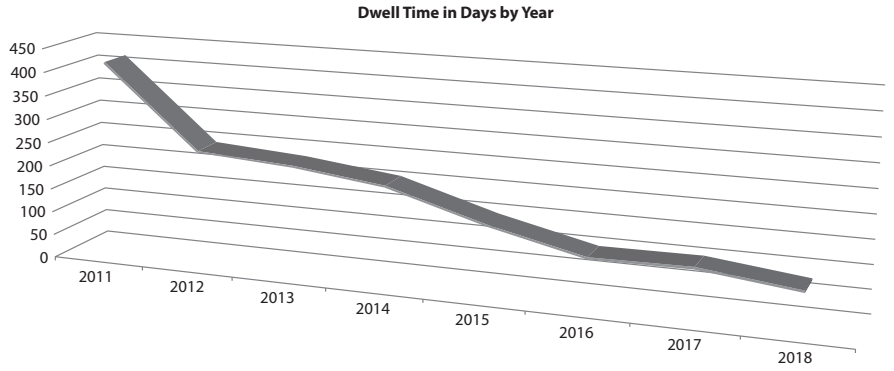


Figure 1-1 Median dwell time in an enterprise network by year

Security is commonly defined by the three properties: *confidentiality, integrity,* and *availability* (CIA). Security testing is an attempt to demonstrate that a system, enterprise, or application can withstand attempts to compromise those three properties. One way to help ensure security is covered within an organization is through regulations. When organizations have to adhere to regulations, they usually need to have a program in place to make sure they are complying with the regulations. This is true whether they are industry requirements or legal requirements. Companies will often implement security testing programs simply because they have to comply with a regulation. We'll talk about different types of compliance that could lead to the need for security testing.

This chapter covers the reasons for performing security testing as well as the different types of security testing. There is far more to security testing than many people realize so we're going to set the table for subsequent chapters. Additionally, any time you are doing any testing, you should be clearly understanding the goals and objectives of your testing. This helps make sure you are ultimately successful because you stated your objectives up front. Most importantly, and you'll hear this a lot, you need to make sure you have permission for the types of testing you are doing. Many of the activities that fall under the umbrella of security testing are illegal without permission. So, let's get started with the types of security testing you may want to do.

Compliance

Let's start at the top. Repeat after me. Compliance is not security. Compliance is compliance. This is an important point, though. Regulations, no matter where they come from, are guidelines that are usually about ensuring that a minimum level of security controls are in place and, often, that the processes as well as the results are documented. Documentation ensures that everything is repeatable, and every time you run through a process, you can compare the results cleanly against previous runs of the process.

You are doing the same thing. How do the results compare? You can determine whether the environment or application has improved, stayed the same, or gotten worse.

NOTE A control is a process, procedure, or technology used to implement the detection, response, avoidance, or remediate a security risk. You may also see or hear of controls referred to as safeguards.

Compliance comes in a variety of packages, and even if you aren't currently involved in security testing, you may be familiar with some of the common regulations or standards that companies have to be compliant with. Some of the most common are described here.

- **PCI DSS** The Payment Card Industry (PCI) has the Data Security Standards (DSS), currently at version 3.2.1. PCI was formed in 2006 by some of the major credit card companies. The DSS is a document outlining a basic set of security controls that anyone who handles branded payment cards must adhere to. This set of standards has continued to evolve since they were first introduced, but some of the standards that are most relevant here are vulnerability testing and management and testing security systems and processes. Additionally, PCI DSS requires such controls as firewalls and encrypting all credit card data, both in transit and also when the card data is at rest.
- **ISO 27000** The International Standards Organization (ISO) maintains a set of security standards that encompass several documents in the 27000 series. The requirements provided by ISO are perhaps best summarized in Annex A of ISO 27001. This is a list of more than 100 controls that anyone looking to be certified as compliant with ISO 27000 must adhere to. These controls include some of the same ones that PCI requires, including vulnerability testing and management.
- **HIPAA** The Health Insurance Portability and Accountability Act (HIPAA) is a law, unlike the other two discussed so far. HIPAA has a set of five categories of safeguards. These five are administrative, physical, technical, organizational, and policies and procedures. There are multiple controls that are recommended, spread across those categories.
- **NIST 800-53** The National Institute of Standards and Technologies (NIST) generates special publications, some of which are related to information security. The 800 series are the ones most relevant for our purposes and specifically, 800-53 which is a set of controls recommended for information systems implemented by the federal government. Since the publications are open for anyone to make use of, they are frequently used by organizations looking for guidance to get them started, as well as a set of requirements they can follow, developed by a group of security professionals. Not everyone in the field is an expert, after all, since everyone has to start somewhere. May as well build on the shoulders of those who have come before so the NIST publications make good framework documents.

- **GLBA** Gramm-Leach-Bliley Act is a federal law enacted two decades ago requiring public companies to let consumers know how they collect and protect information from those consumers. This law is a forerunner of other privacy laws like the General Data Protection Regulation (GDPR) in Europe and the California Consumer Privacy Act (CCPA). Other laws will come in other jurisdictions to help mandate appropriate controls over consumer information. Sometimes, ensuring protection means performing testing to verify that controls in place work as expected and actually protect consumer information.

These are just a handful of laws, regulations, and requirements that organizations may have to adhere to and are often audited against. Auditing is the act of verifying that what you are doing follows expectations, whether those expectations are your own documentation or whether the expectations are from a set of standards from an industry body like PCI. Auditing ensures you are doing what you are either supposed to be doing or what you are saying you are doing. As noted earlier, this is not a guarantee of security but a guarantee that you have done what you are supposed to be doing. For organizations that don't have the expertise to implement fully robust and resilient security programs with experienced security professionals, achieving compliance with these standards is a good start and may protect them from easy attacks.

Testing for compliance is a common activity, though. This book is about developing labs, but it's unlikely you are going to be testing for compliance in a lab unless you are testing applications for deployment. However, everyone who wants to perform this type of testing or even learn auditing needs to learn somewhere and labs are a good place to do that. Regardless, one reason organizations perform security testing is because of compliance with regulations and laws.

Security Testing

Let's talk security testing. Perhaps before we do, we should set some parameters. As mentioned earlier, security is a concept propped up by the three legs of confidentiality, integrity, and availability. These properties are collectively known as the CIA triad. Ideally, you are familiar with these concepts, though if you aren't, it's not the end of the world. They are not especially arcane, even if they are essential to information security, which may seem arcane and complex if you don't have a lot of experience with it (and sometimes even if you do have a lot of experience). One great thing about these concepts is they really are everyday sorts of ideas. All you need to do is grab a dictionary and look them up. We haven't twisted them at all. Should you have a good grasp on these concepts, you can feel free to skip ahead a bit.

- **Confidentiality** When you are sharing a secret with someone, you expect that secret to remain protected by the person you are sharing it with. You don't want the secret to be exposed to the outside world. When you tell your best friend in junior high school that you like Kelly, the girl on the other side of homeroom, you don't want your friend to run to Sarah, Kelly's best friend, to let her know. That would be hugely embarrassing. While you were in the room.

And maybe even standing next to them when it happened. This is an example of confidentiality. Information you don't want shared broadly should remain accessible and intelligible by those for whom it is intended. In the information security space, you can ensure confidentiality through controls like strong authentication and encryption.

- **Integrity** You may think of this as something a person has. Politicians are sometimes discussed as having integrity. This means that person has a sound moral character or foundation. In simple terms, someone who has integrity does what they say they are going to do. A person with integrity matches their actions and their words, meaning they don't do one thing and say something else. They are not "do as I say, not as I do people." When it comes to information security, it means there is a soundness of systems and information. Information that has been stored should be exactly the same as when it is retrieved. This is matching the walk and the talk. The same holds true for information in transit. You can think of the old telephone game here. How many *generations* (people) does the information go through before it becomes corrupted? Once the data being transmitted has changed, it no longer has integrity.
- **Availability** This is probably the most straightforward of the three principles. Systems that can't be reached when they are wanted are not available. The same is true for information. If it gets deleted, it is not available. While it seems very straightforward, this is also probably the most far-reaching. I have a couple of servers at home that I use for building up virtual machines for demonstration purposes or just for tinkering around. If the power at my house goes out, my laptop that I work on will continue to function but the systems where my virtual machines are housed will no longer be available. This may impact my ability to work.

It's worth noting that actions don't have to be malicious in nature for these properties to be compromised, resulting in an issue with security. The power example above, for instance, is likely not malicious. While power lines are buried where I live and power outages are not very common, they do still happen from time to time. They are as likely to be a result of construction in the area than they are a result of someone standing outside my house cutting the power. I don't live the kind of life that would inspire someone to want to cut power on me. The same is true with the other properties. You could have a failing hard drive or a bad driver and that could cause corruption of data either while it's being stored or retrieved. This is not malicious and yet the data is still corrupted. Ultimately, the reason doesn't matter when it comes to determining whether there is an impact to a system's security.

Because we can have security implications even when there is not malicious intent, security testing is essential. It's not just about protecting against bad people doing bad things. Security testing has to ensure software and systems are capable of being resilient to any failure. This is why there are so many different types of software testing. Different people and different organizations will have varying ideas about which of the three CIA properties are most important and the importance may vary from one situation to another. Because of this, you may find the different approaches to security testing valuable.

Software Security Testing

It's common for software to undergo quality assurance testing. Of course, there are no rules as to how that happens or how detailed it is. It's a long-held maxim that fixing bugs is cheaper the earlier in the process they are identified. Figure 1-2 is a graph showing one estimate of the cost difference between finding a problem in the requirements stage and all the way up to post-deployment/release. Different sources will provide different multiples for the different stages. The baseline is in the requirements stage, though you wouldn't fix a bug in the requirements, though you could identify bad requirements that could introduce problems. Given that, the best place is to identify bugs during the coding or development stage.

The rationale is that it costs more in testing because of the rework involved. A programmer has to get the bug report, validate the bug, identify where the bug actually is in the code, then fix the bug before handing it back to the testing team, where it all has to be tested again. The reason it's more expensive post-deployment in a traditional software release model where the software is packaged up and has to be sent back to the customer again is because the bug not only has to make it all the way back to the developer, following the same process outlined above, but the package has to be built and redeployed to the customer. These costs are coming down a little, since it's not like CDs have to be shipped like they may have a decade or more ago. However, there is still a lot of cost involved in the people who have to be involved in the redevelopment, retesting, and redeployment. As well as the cost of any shipping, including hosting it on a web site where it can be downloaded.

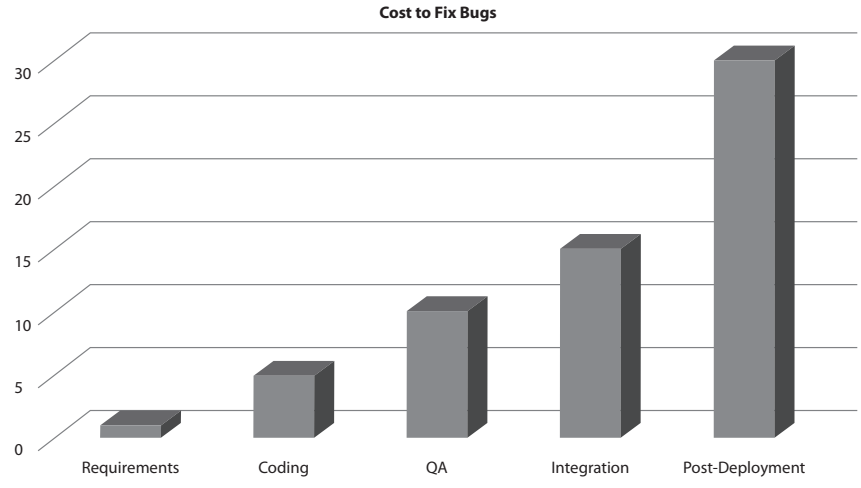


Figure 1-2 Cost of fixing bugs by stage

Bugs can fall into a security bucket. When it comes to security vulnerabilities, the costs may increase significantly, and they may not be limited to the software company. A security vulnerability may expose any customer that uses that software to attack, which could incur damage or losses to that customer. A study by the Ponemon Institute released in 2018 suggests that the cost of a data breach averages $3.86 million. The study says this is an increase of 6.4 percent over the year before. Given the number of breaches that occur each year, this is a significant amount of money globally. The software company that distributes vulnerable software may not be liable today in the case of that software being part of the reason an attacker gained access, but that may change over time.

This brings us to the first kind of testing that you may be doing. Software security testing looks to try to root out security-related vulnerabilities within software. Ideally, this testing is being done early in the development stream, but it may not be. In fact, some companies have something called bug bounties. Independent testers may perform testing on released software at home in order to identify vulnerabilities that they will get paid for telling the software company about. If you are one of these independent testers, you need a place to perform this testing, which means you need a lab.

The Open Web Application Security Project (OWASP) maintains a list of common vulnerabilities that are known to exist in software applications. This is a list that is updated as the commonality of vulnerabilities increases or decreases year by year. The current list at the OWASP web site (www.owasp.org) is from 2017, which doesn't make it out of date because there have not been significant changes to the list over years, aside from some consolidation of multiple vulnerability types into a single type, which has opened up some space for some new vulnerabilities. The current list is as follows:

- **Injection** Injection vulnerabilities stem from a lack of input validation. This allows programmatic content like structured query language (SQL) or operating system commands or even Javascript through the application to be handled by something on the back end. These injection attacks expose critical elements of infrastructure, potentially allowing access to sensitive information or just plain giving an attacker access to issue commands to the system. This could mean a years-long foothold in an environment for an attacker where they can do pretty much anything they would like. While the dwell time numbers referenced above show attackers today being in networks for something over two months, the reality is there are plenty of cases like Marriott, where the attacker was in the network for more than four years. These injection attacks are not only highly common, but they are also an easy attack vector for adversaries to get into the network.
- **Broken Authentication** Passwords are a common way for attackers to gain access to environments. If it's not people just giving up their passwords to the attacker in a social engineering attack, it's programmers doing bad things with the password. This may include storing the password in clear text somewhere. It may include not using good practices with hashes. It could also be default credentials in use or programs handling authentication allowing weak, easily

guessed passwords. This may also mean allowing multiple guesses of the password for services. Computing power is cheap so brute force attacks on passwords are easy. Protecting authentication mechanisms can be done but far too often isn't and attacking authentication is often trivial.

- **Sensitive Data Exposure** Sensitive data, including passwords or other credentials, is often not protected well enough. There may be a period of time, for example, that data is passing through an application's memory space in plaintext. If it's in memory, it can be captured, which means it's exposed. Sometimes, once a user has been authenticated, there is a session identifier that is used to maintain the session, especially in a web application, which has no ability to maintain the state of a user as part of the protocols used in web applications. If these session identifiers are not properly generated, they can be used to replay credentials, making it seem like an attacker has been logged in, when in fact the authentication was done by the actual user. Session identifiers should also be protected, where possible. This can be done, easily enough, using any number of encryption ciphers. There are libraries for many common ones that can be integrated into any program.
- **XML External Entities** If an attacker can upload data formatted with the eXtensible Markup Language (XML), they may be able to get the system to execute code or retrieve system data. This is a form of an injection attack. An XML External Entity attack uses a feature of XML, allowing a system call, which refers to either a program or a file in the underlying operating system. Attackers may be able to perform reconnaissance against an internal network using this attack.
- **Broken Access Control** Access control is determining who should have access to a resource and then allowing the user to have that access. There are many ways that access control may be broken in an application. A direct object reference is one way. An application may be developed in such a way that there is an assumption that if a function gets called, it has to have been called from another function where the user has already been authenticated. If the attacker goes directly to this page or function in the application rather than the page or function that would normally have been a predecessor, the attacker can make use of the functionality without ever having to authenticate. This can mean the attacker can gain unauthenticated access to a function that provides either administrative functionality or sensitive information.
- **Security Misconfiguration** All manner of ills falls into this category. Any configuration setting that impacts the security of the application might be set incorrectly, allowing an attacker to take advantage of the misconfiguration. It could be permissions that are left wide open so anyone who has access to the system can view or even change information stored. This can also mean the use of default usernames and passwords. Additionally, it can mean something like a typographic error. Let's say you are configuring a firewall, and you need to block 172.0.0.0/8 because you are receiving what appears to be attack traffic from a large number of addresses in that block. Instead, you enter 171.0.0.0/8.

This means you will continue to get attacked while you are, instead, blocking potentially legitimate customers from getting through. There are a lot of areas where either by design or mistake, systems are misconfigured.

- **Cross-Site Scripting** Cross-site scripting is also an injection attack, using a scripting language injected into the normal web page source. While this is a case of commands being injected, the difference here is that the target of the attack is nothing on the server-side. Instead, a cross-site scripting attack targets the user's system. Any script that is injected is executed within the user's browser. Generally, this means the attacker is looking to gain access to data that may be available within the browser. This may be credentials or session information. It's unusual that a script run within a browser would be able to get access to files in the underlying operating system, but it may be possible with a weak browser or a vulnerability.
- **Insecure Deserialization** Deserialization is where you take a stream of bytes and put them back into an object that has been defined within a program. Imagine a complex data type consisting of a name, a phone number, a street address, a city, a state, and a zip code. The name, street address, city, and state are all character strings. The phone number may be represented as a string of integer values, and the zip code may also be represented as a string of integer values. When you take all of these values and serialize them, you are converting them just to a long string of bytes. There may be some framing information in the byte stream to indicate where each of these values start and stop, especially if each of the character strings are not fixed size. Strings may be null-terminated, which means the last byte in the string has a numeric value of 0. This can be the framing. Deserialization will take a stream of bytes and put the bytes back into variables in the object. The problem with this is that if the byte stream does not align well with the object, you can end up with failures. If you are trying to put integer characters into variables but the value of that character is way out of bounds, what happens to the program? Insecure deserialization is when the deserialization happens without any checking to make sure what you are getting in is what you expect and it will fit into the object where it's supposed to.
- **Using Components with Known Vulnerabilities** It is common for complex systems to make use of supporting software. If a system developer or designer uses available components to speed the development of the application or overall system, it is up to that developer/designer to ensure the software underneath what they are developing is kept up-to-date with fixes and updates. There is a well-known example of this. Experian was broken into, in part because of a security flaw in the Apache Struts framework. This is a library that web applications are built on top of by those who are writing Java applications. When you use existing components like libraries or even services like the Apache or Nginx web servers, you are responsible for keeping those components up-to-date. This can be challenging because not all projects make announcements when they have new versions or, especially, when there are security-related updates.

However, having out-of-date software packages on a production system is a very common occurrence. Even in cases where downstream packages are updated and there are updates available, some organizations don't have processes in place to keep all software up-to-date. Even if they do, often there is a long delay between the software being available and when it is deployed.

- **Insufficient Logging and Monitoring** Operating systems and applications often have a limited amount of logging enabled by default, reducing visibility when problems occur. When incidents arise and attackers are in your environment, you want as much data as you can get. There are two standard logging protocols/systems, one for each Linux/Unix and Windows. This means any application that runs on both has to either be able to write to either or else they write their own logs. Even when the facility to generate logs exists, it's still up to the application developer to write logs. This is often overlooked, especially in cases where rapid application development is used. Methodologies like extreme programming, agile development, and rapid prototyping often eschew documentation, which may include logging, because this is a feature that isn't typically requested by users. Operational and security staff may have logging requirements, but end users wouldn't. This means the need for logging is often not addressed. Without logging, you can't monitor behaviors of services and applications.

This sort of testing should be handled as part of some sort of quality assurance process during the software development life cycle. However, security testing of software may not be done at all. It's not entirely uncommon for software to be developed without any security testing. Even software that does not offer up network services can have vulnerabilities that may be useful to attackers. Therefore, it's not helpful to believe that because it's a native application that only interacts with files on a local system so there is no chance it could be compromised, lessening the need for security testing. Everything is fair game.

Stress Testing

Most software development projects do not have to worry so much about how much they need to handle, in terms of volume of data or requests. However, any mission critical application should be stress-tested. Keep in mind that not all software runs on general purpose computers. A general-purpose computer is one that is designed to run arbitrary code, meaning if you can write a program for the computer, the computer will run it. Some computers don't have all the relevant pieces that would allow for these arbitrary programs to be run. The software that runs on these types of devices is commonly called firmware because there is no chance to alter it. It's burned onto a chip where it can't easily be changed. It may not be able to be changed at all, though that wouldn't be as common today, considering the prevalence of security vulnerabilities even in hardware devices.

NOTE These general-purpose computers follow something called a Von Neumann architecture. John Von Neumann outlined this computer architecture in a paper in 1945, describing the components necessary for these computing devices. According to John Von Neumann, and you'll recognize this explanation, a computer needs a processor, which includes an arithmetic logic unit and processor registers; a control unit that can track the instruction pointer in memory; memory that stores the data and program instructions; storage devices; and input/output mechanisms.

Special-purpose devices, especially those of the variety called Internet of Things, are often built using lower-capacity processors. These lower-capacity processors, including the processor necessary for handling network traffic, may fail when there is too much for them to do. Failure in the field wouldn't be such a good thing. You may have an analog telephone adapter, such as one from Cisco or Ooma or some other Voice over IP provider, just as an example. If one of these devices happened to fail because of a large volume of traffic being sent to it, you'd have no phone service. If you were running a business on one of these devices, you may be without phone service. Thermostats, like those from Nest, Honeywell, and other manufacturers, may be subject to failure on stress, without proper testing, which may mean you don't have heating or cooling.

This is not to say that only large volumes of traffic are how you would stress-test devices. Another way to stress-test any piece of software or device is to send it data that it doesn't expect. These malformed messages may cause problems with the application or device. There were a number of ways to crash operating systems 20 years or so ago using these malformed messages. A LAND attack, for example, was when you set the source and destination to be the same on network messages. This would cause stress to the operating system as it tried to send the message to itself over and over, in an endless loop. A Teardrop attack resulted in an unusable operating system when fragmented messages were sent in such a way that the network stack was unable to determine how best to reassemble the fragments. These sorts of unexpected messages lead to unreliable behavior. While most of these sorts of attacks were long ago addressed in operating systems, they are just examples of the types of stress testing that could be done. Just because the networking parts have been ironed out doesn't mean the applications are always best able to deal with bad behavior.

As an example, there are strategies to take down web servers that have nothing to do with unexpected activity or malformed anything. Instead, you can use slow attacks with legitimate requests to a web server. As the intention is to hold open buffers in the web server until no other legitimate request can get through, this is also a form of stress testing. You can see in Figure 1-3 a run of a program called **slowhttptest**, which is used to send both read and write requests in a manner that the web server can't completely let go of the existing connections.

Stress testing is not about performing denial of service attacks, since the objective is not to deliberately take a service offline. The objective is to identify failures of an application. Filling up a network connection so no more requests can get through is not especially useful when it comes to security testing. This is where clearly understanding

```
Sun May 19 20:54:08 2019:
        slowhttptest version 1.6
 - https://code.google.com/p/slowhttptest/ -
test type:                        SLOW HEADERS
number of connections:            50
URL:                              http://192.168.86.1/
verb:                             GET
Content-Length header value:      4096
follow up data max size:          68
interval between follow up data:  10 seconds
connections per seconds:          50
probe connection timeout:         5 seconds
test duration:                    240 seconds
using proxy:                      no proxy

Sun May 19 20:54:08 2019:
slow HTTP test status on 10th second:

initializing:         0
pending:              1
connected:            49
error:                0
closed:               0
service available:    NO
```

Figure 1-3 Slow HTTP stress testing

the application and the goal of the testing is essential. You can't test anything without understanding what the objective of the testing is. You'll be spinning your wheels without anything useful to show at the end of the day.

Penetration Testing

You may think this is the be-all and end-all of security testing. From my perspective, and you and many others will have their own perspectives, penetration testing is perhaps the least interesting and least useful type of security testing. In order to make clear distinctions so there is some context for future discussion, we're going to draw a line in the sand about what penetration testing is, so it's not a wide-open door that you can shove everything into. From my experience, the point of penetration testing is often limited in scope and can be more compliance-driven. This means there is some regulation somewhere that says they need to have their environment tested. It may be said that the testing has to be done by a third party. This is meant to get a fresh set of eyes on the environment and also skip any preconceived notions about what may be in place. There are no assumptions made in the case of a third-party test.

When it comes to third-party testing, you could do black box testing, meaning the person doing the penetration testing has no awareness of what the environment or application looks like. They have zero knowledge going in. This means they need to figure everything out as they go. This is thought to be more realistic since that's how an attacker is expected to operate. On the other end of the spectrum is white box testing. This means

the tester has complete access to everything. They have knowledge of the environment going in as well as access to insiders for additional information as the testing goes on. In reality, the tester may be working somewhere in between. You could consider this gray box testing.

When it comes to penetration testing, there are some well-known methodologies that can define behaviors. In part, they are thought to mimic the behavior of an attacker. When it comes to penetration testing, one simplified approach is demonstrated in Figure 1-4. A tester will need to do some preparation. This is essential when it comes to any form of security testing because part of the preparation is identifying the scope with the target. It's important to keep in mind that what you are doing is illegal without permission. This is even more true in some parts of the world than others, but even in the United States, we have the 18 U.S. Code § 1030, called the Computer Fraud and Abuse Act of 1986. It has been amended several times in the intervening years. In short, it says that if you knowingly gain unauthorized access or exceed authorized access, you are in violation of the law. This is, you may note, a very broad definition. While not all violations of this law are prosecuted, of course, considering the amount of illegal activities in this arena, you need to be aware of laws surrounding your actions to best protect yourself. At a bare minimum, without permission, what you are doing is unethical.

As mentioned above, there are a few testing methodologies. One of these is the Penetration Testing Execution Standard (PTES). This is a methodology developed by a number of security practitioners. According to the PTES, there are seven steps to a penetration testing engagement. They are as follows:

1. Pre-engagement interactions
2. Intelligence gathering
3. Threat modeling
4. Vulnerability analysis
5. Exploitation
6. Post-exploitation
7. Reporting

It's beyond the scope of this book to cover any of those steps in any detail. They should be fairly straightforward, though. You are preparing, understanding your target, identifying vulnerabilities that may be exploited, exploiting them, potentially moving through the environment, and then reporting to your target/client. As always, the objective should be to provide specific feedback to an organization so they can improve their overall stance on security. When you are finished with them, they should have actionable information that can protect them from malicious actions in the future.

Figure 1-4 Basic penetration testing steps

PTES is not the only penetration testing framework that could be used. PCI provides penetration testing guidance since DSS requires penetration testing as part of getting PCI certified, which is required if you do any handling of any payment cards. The Federal Risk and Authorization Management Program (FedRAMP) also provides guidance on how to perform penetration testing. FedRAMP ensures cloud-based services offer security if government agencies are to procure those services.

If you were to look up penetration testing frameworks, you will find a number of methodologies that don't specifically refer to penetration testing. Instead, the more generic security testing phrase is used. This is a phrase that covers all manner of sins. The National Institute of Standards and Technology (NIST) provides security testing guidance in special publication 800-15. OWASP has security testing guidance in addition to their common vulnerabilities list. Another open source project that maintains comprehensive security guidance is the Open Source Security Testing Methodology (OSSTM). The current version of the manual describing the methodology (OSSTMM) is version 3, but version 4 is in draft at the time of this writing.

All of this is to say that there is no one standard or definition for penetration testing. Additionally, some of the methodologies that may be commonly used for penetration testing are really referred to as security testing standards. Before we add another type of testing to consider, let's clean this up. For the purposes of this book, penetration testing is time bound, usually with a very small testing window, on the order of days. The scope is well defined ahead of time, as you'd expect, and too often the really sensitive pieces are left out of the scope because they are mission critical or there are fragile devices in those subnets. This, to me, is the real problem with penetration tests. When they follow the rules just outlined, the organization is left with a false sense of their preparedness for bad things to happen. The only way to get a decent sense of how protected you are is if you take the cuffs off the testers.

Red Teaming

Red team activities are different from penetration testing for our purposes, even if you may find the two terms conflated or swapped in the real world. One simple difference between a red team and a penetration test is that a penetration test can be performed by those who are inside the organization. You can have an internal team performing penetration tests to continue to probe for weaknesses. A red team, in a technical sense, is an independent group, which should make it a third party rather than one that is internal. It's difficult for an internal group to maintain the objective perspective necessary for a red team engagement. Whether we are talking about security testing or even a thought exercise, the job of a red team is to challenge the target. You may bring in a red team if you have a new cryptographic algorithm, just as another way of thinking about it. The job of that red team is to pick apart the algorithm in order to identify any hole that may exist. The same is true for red teaming in a security context.

One way red teamers help to improve security is to approach their target as though they were real attackers. This isn't necessarily the "follow an attacker's methodology" that a penetration test may claim to follow. This should be more of a "there are no rules" approach.

The objective is to get in. Not just to find vulnerabilities, though that should be the ultimate outcome. A red teamer probably isn't going to follow a set pattern of enumeration, scanning, exploitation, and so on. A red teamer may just jump straight to tactics they either know will work or strongly suspect will work. Again, this isn't so much about enumerating all possible vulnerabilities. It's about gaining entry to the organization to identify holes along the way—not theoretical vulnerabilities but actual, exploitable holes.

Red teaming may encompass activities like social engineering or password stealing. These activities may not be done in a penetration test since some organizations may believe their password policies are strong (they'd be wrong) or that their people know not to click on bad e-mail (they'd be wrong). And they are really only engaging in this exercise to identify technical vulnerabilities because that's what's important (they'd be wrong in that assumption). In spite of the fact that roughly 90 percent of attacks today happen through a human channel and not a strictly technical channel. Often organizations spend a lot of time shoring up the defenses at the edge of their networks, assuming that the attacker is going to come trundling up to attack the large wall that has been constructed. The idea of defense in depth that is commonly implemented on the edges of networks really misses the reality of the attacks modern adversaries are using. Testing social engineering strategies to identify weaknesses in the human element is important. This allows organizations to develop controls to prevent attackers from crawling up through the sewers while all the defenses are focused on the perimeter walls. At a minimum, the organization can implement additional visibility so they can see the sewer grate being cracked open.

While at least some of the actions performed by red teamers are based in a form of social engineering where you have to engage with your target, it's still necessary to practice the technical skills needed to perform these social engineering attacks. Most of the time, you'd be developing pretexts (the story that will be presented to the user, which would encourage them to do what you want them to) as well as the infrastructure and systems needed to support the attacks. You might develop a story that you are from the security operations center and are checking on something on the user's system, but you still need the technical ability to move off that system once you are on it. It's not all about conning people into doing things they shouldn't be doing.

Blue Team/Operations Testing

Alongside red teaming, you may also perform blue team testing. The blue team is the defensive team if the red team is offensive. The purpose of blue team testing is to identify places where the operations team is unable to detect the existence of security events. This may be done in conjunction with the red team. While this is commonly done in a production setting, it does potentially expose all production systems to anything the red team can throw at it. Doing a blue team test in a lab requires emulating production systems in a lab setting. This can be costly. However, in order to test operational capabilities, it may be useful to establish a minimal lab with enough systems to generate alerts so new detection capabilities can be tested. After all, assuming that protecting the border will be sufficient to keep people out, so constantly testing the edges of the network or even testing the hardening of systems has to be the right answer is a bad approach.

The bad guys are going to get in. It's inevitable. What's important is being able to detect when they do get in. When new detection capabilities are introduced, they need to be tested somewhere to ensure they function as expected. This should be done in a lab setting first and then re-verified in production. However, you might need a minimal amount of lab network in order to test detection capabilities. You can use the same techniques to establish a lab for blue teaming as you would for penetration testing or red teaming or even software testing.

Goals

No matter what you are doing, you should have clear goals and scope identified before you start. This is certainly true of security testing. Too often, perhaps, people performing security testing lose sight of the goals. If they are engaged in penetration testing, for instance, they may be focused on how many systems they can compromise or how deep into the environment they can go rather than thinking about what the company that hired them was looking for. Ultimately, the objective of any type of security testing should always be to identify weaknesses that can be exploited. Identifying a vulnerability in systems is good, and exploiting it hundreds of times to obtain access to those hundreds of systems is probably a waste of time since it adds no new knowledge to the outcome. Once you have identified one and exploited it, you can put it into your report. Running some checks to demonstrate that other systems are similarly vulnerable supports the importance of getting it fixed. Exploiting it over and over is time consuming when you likely have very limited time available. Moving on to other vulnerabilities is smart.

Your goals will similarly feed into the type of testing you want to accomplish as well as where you should be performing that testing. By way of example, let's say you are performing security testing on a web application. Your goal is to identify potential security issues within that application. You may assume that the organization you are testing for has a lab in place for their own testing. With some luck, the lab environment mirrors the production environment, meaning all the software and versions are identical between the two. In that case, you may consider performing your testing in the lab. Even in cases where your actions are not intended to be destructive, you may find that your testing causes bad things to happen on the systems under test. As an example, I was performing some web application testing on a nonproduction system several years ago. In preparation for a full application assessment, I was performing a spider of the site—pulling all the pages on the site in order to get an exhaustive inventory of them—and ended up taking the site down. It shouldn't have happened, but it did. You can't ever tell for sure.

Goals are best written down. This is especially true when it comes to any security testing you are performing on behalf of someone else, no matter if you are an employee or if you are being contracted to perform the testing. Because some types of security testing are illegal in many jurisdictions without approval, it's absolutely essential to be very specific about what you are doing if you are touching any devices that are not yours.

Even better than writing down goals is to develop test plans. A test plan is documentation indicating the function being tested, the expected outcomes, and steps used to achieve the testing. The more detail you provide in your test plan, the better you'll be able

to duplicate it later if you need to. Additionally, if you are working with anyone else, having a test plan means anyone should be able to pick up the plan and perform the testing.

NOTE It's essential to get permission before you perform any security testing if you are testing systems or applications that don't belong to you. The more detailed you can get with your permission, the better off you are going to be. Keep in mind that many types of security testing are illegal in many areas if you don't have explicit permission. This is not a case where it's better to seek forgiveness after the fact rather than get permission. If you don't own it, get explicit permission and be as detailed as you can.

Isolation

Security testing can be an issue because of the very nature of what you are doing. No matter if you are stress testing or learning penetration testing or some other activity, it's likely a wise idea to have your systems isolated. This is especially true if you are working in an enterprise since you don't want to negatively impact any production systems. There are several ways to achieve this isolation. If you are using physical systems, you can see an example of a way to isolate testing systems in an enterprise network in Figure 1-5. While it's possible to use entirely air-gapped devices, it's not always practical to do so. The best way to isolate and protect is to implement a firewall, as you can see in between the testing systems on the left in Figure 1-5 and the enterprise network on the right.

Of course, this is not the only way to achieve isolation. Another approach is to use entirely virtual machines. This would allow you to have some additional control by

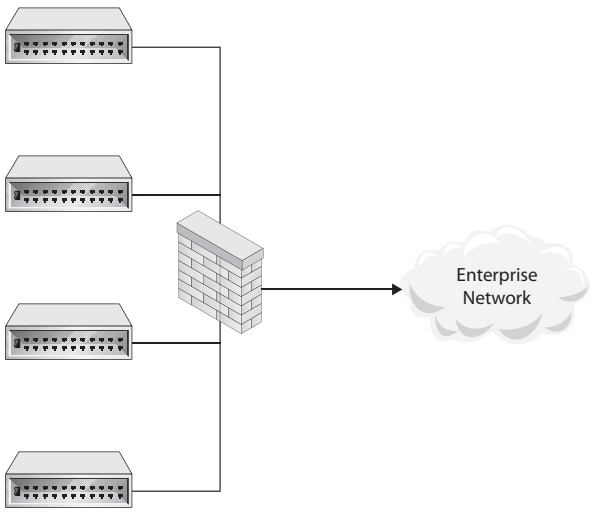


Figure 1-5
System isolation

putting all of your testing systems into an entirely isolated virtual network. All of the systems you are either testing or working from will reside entirely within a single hypervisor. You can gain remote access to the systems you are testing with through your hypervisor. There are a number of techniques that can be used to accomplish the remote access even in the case where you have isolated your systems into a virtual space. Some virtual machine environments also allow you to access the virtual machines using a mobile device, meaning you can operate your testing environment from just about anywhere. Just use your imagination.

Isolation comes down, in part, to network topology. Using the right network design and appropriate controls, you can really isolate systems under test. In the next chapter, we'll cover some of the foundations of networking and network design so you can better understand how to put your systems together in ways that will help protect both your systems under test as well as the systems you are using to perform the testing.

You vs. the Enterprise

To be completely honest, this book is going to feel a little bifurcated at times, simply because one reason for establishing a testing lab is to learn how to perform security testing—to explore without fear of causing problems elsewhere. There was a time when it may have been okay, at least legally, to explore and learn on systems you didn't own. However, that was decades ago. Today, there are laws in place that require you not to go trampling through someone else's computer garden. It's not only not legal, it's neither polite nor ethical. It's just not done, as they say. This sort of activity—developing a testing lab for learning—is generally one that is performed by individuals rather than businesses. You are expected to know your job before you get hired for your job, leading to the reason for the lab to begin with—people want to learn how to test and break things so they can get jobs requiring those skills.

One aspect of security testing that's perhaps worth mentioning is the bug bounty. Individuals often spend time trying to break application software or even network defenses because the company that is responsible for the software or network will pay money for any flaw that is found and verified. This is another reason why people may be interested in having a lab—so they can have a place to work on this testing. People will have their own opinions as to the ethics involved here, though as someone who spent time at a large company working on application security, I've run across several people who had not been asked to investigate anything and felt it was their right or duty. Often, it was for the notoriety of being able to announce a bug. This made it far less altruistic as far as I was concerned. It also made it potentially unethical since they had not been invited to dig into someone else's systems. Ethics are often highly fungible so your mileage may vary here. Bug bounty programs seem to be something else—it's a way of parceling out the enormous task of security testing to a bigger group of people than the company can afford to pay in direct employees. In this case, it's *you* being a single individual, but maybe you have the needs of a larger business if hunting bugs is something you make money from.

When some forms of testing are performed by a business, however, they have to be done in production because the point is to ensure the production systems are adhering to appropriate standards and aren't loaded with easily exploited vulnerabilities. In a business environment, labs rarely completely mirror the production systems and networks. You can test in a lab but you won't be able to fully identify issues that may impact the business in a lab.

This is not to say that businesses don't have need of lab environments for testing. Smaller organizations can have the same needs as larger organizations, but they have fewer resources to bring to bear. This is why this book is useful for them. We're going to cover how anyone, whether they are a business, large or small, or an individual can build a lab they need, regardless of what the purpose of the lab is.

Summary

The term security testing can mean a number of different things and encompass several different activities. You may hear the term and think about something very different than I do. Security testing may be performing quality assurance testing focused on security requirements. You may be working on testing either a native application or a web application. Software security testing may be focused on the top ten vulnerabilities that OWASP defines. While the top ten generally focuses on web applications, the programming fixes for remediating these vulnerabilities often apply just as well on native or mobile applications. This means developing testing plans against those vulnerabilities is often a good idea, no matter what the target of your testing is.

Compliance with regulations or laws is one driver for security testing. Some organizations may have to adhere with laws like HIPAA if there is any personal health information that gets handled within the organization. If they handle credit cards, they may need to follow the PCI DSS. While auditing against requirements is common, some of these requirements also specify different types of security testing, including application testing or penetration testing. Sometimes they even specify that the testing has to be done by third parties.

You may be looking at penetration testing. While penetration testing is generally done in a production environment, learning the tools and techniques to be able to perform penetration tests should be done in a safe environment. This is where a lab can be essential. Fortunately, there are many ways to create a lab that you can use for learning in. Considering the power of most tools you would use for penetration testing and the capability for causing damage, it's essential that you get permission since many testing actions are illegal without permission.

Red team and blue team are also types of security testing. You can practice your red teaming skills in a lab but it's unlikely you'd really be performing a red team assessment in a lab and less likely you'd be doing blue team testing in a lab. Of course, anything is possible and, as I said, you can certainly practice both sets of skills in a lab. It's easy enough to set up and configure in a lab the same sorts of tools you'd use for operational monitoring in production.

Writing test plans is a good idea. Even if you are penetration testing, at least having a documented methodology is a good idea. Fortunately, you don't have to spend a lot of time developing methodologies from scratch. There are a number of testing methodologies that can be used. Some are specifically geared toward penetration testing, while others are focused on security testing as a more general idea. These may be good starting points. Of course, test plans and methodologies are different things. Test plans are more specific and detailed. No matter which way you go, it's helpful to have your goals identified up front.

Isolating testing systems is important. Even if you are setting up a small environment at home for testing, you don't want to have an unexpected impact on the other systems on your network. After all, you could inadvertently find yourself testing and damaging systems that may house important personal documents like tax records or digital photos with all of your memories on them.

Whether you are an individual or a company, you can benefit from developing a security testing lab. You don't have to have the same objectives or needs to have the desire to set up a testing lab. Testing labs can become complex, and there are a lot of different ways to build one up. The objective of this book is to present you with a lot of detail in one place so you don't have to do all the hunting I've had to do over the years to build my own labs at home.

In the next chapter, we will start to lay the groundwork for creating your lab. Every lab has to have a network, after all. It's not only about systems. You need to have a medium for those systems to communicate with each other.