Hacker Highschool

SECURITY AWARENESS FOR TEENS

LESSON 10
WEB SECURITY AND PRIVACY

HACKING IS LEARNING
www.hackerhighschool.org

ISECOM

WARNING

Table of Contents

Contributors

Pete Herzog, ISECOM

Marta Barceló, ISECOM

Chuck Truett, ISECOM

Kim Truett, ISECOM

Marco Ivaldi, ISECOM

Bob Monroe, ISECOM

Greg Playle, ISECOM

Cor Rosielle, ISECOM

Mario Platt

Monique Castillo

Introduction and Objectives

What you do on the World Wide Web (the Web) is your own business, isn't it? You might think so, but it's just not true. What you do on the web is about as private and anonymous as where you go when you leave the house. You'd think your comings and goings are your own business, and the people of ISECOM would agree with you. But in most countries government and private investigators are free to follow you around, collecting data on where you go, whom you meet, and what you say. The focus of this lesson is learning how to protect yourself on the web and to do that, you will have to know where the dangers are.

Web security is a whole lot more than making sure your browser doesn't collect tracking cookies. In this lesson we hope to show you how deep the web actually goes. When most people think of the Internet they think of the web. Since you have already read some of the other lessons, you know that this isn't true. The world wide web is just a small fraction of the digital communication structure we know as the Internet. With web security we have to look at web servers, the software running on both the client and the server-side, as well as all the services we depend on. Our phones are the most connected devices ever made, yet few people understand the difference between Java and Javascript.

The code of most web applications is no longer just Hyper Text Mark Up Language (HTML), either. Entire operating systems are being used to enhance your online experience to sell you things or just collect personal information about you. Those pictures you post to your friends on social media are being used by other entities. You are unknowingly part of the largest privacy invasion ever created. And you are the one providing all the information.

Our focus here is not to scare you but rather educate you about how far web security really goes. We will cover the foundation of web applications, their uses and misuses, vulnerabilities, security practices plus some tricks of the hacking trade. As always, Hacker Highschool is not here to teach you criminal techniques. We endeavor to show you how to be security conscience. You must think on your own and learn on a continuous basis about the world around you.

Your phone, computer, tablet, vehicle and even home appliances are connecting to the internet whether you know it or not. Wouldn't you like to know what they are saying about you along with the data that is being collected? We hope to show you ways to protect your privacy and personal data. Perhaps you will take up the challenge and begin teaching others as the Institute for Security and Open Methodologies (ISECOM) does. Education is a powerful weapon.

Fundamentals of Web Security

The focus of this lesson is learning how to protect your privacy on the web and how to keep your own web sites safe against intelligent attacks. To do that, you will have to know where the dangers are and how to bypass those dangers. Our topic here is going to expose you to the massive security challenges facing web site owners. You probably heard about attacks such as SQL injections, buffer overflows, cross site scripting and more. This lesson is going to demonstrate what these attacks really mean to you as well as methods to exploit/defend each web based vulnerability.

First off, you need to understand that the original idea behind the Internet was to link academic organizations with certain **U.S. Department of Defense (DoD)** organizations. This basically meant that the Internet (**ARPANET**) was built on the premise that security wasn't needed, since you already had to be either a research university or a government entity to connect to the network. The original Internet linked a bunch of big computers all over the U.S. to each other so security wasn't really an issue, the availability and durability of these connections were some of the real concerns.

This is where the **Transport Control Protocol (TCP)** was first introduced to deliver packets of data from one big (really big, like we are talking about a computer the size of your garage) computer to another big computer far away. Way back then, nobody thought the Internet would ever get as large as it has today. **Internet Protocol** V4 (IP as in TCP/IP v4) addressing used a 32 bit number sequence which limited addresses to a specific amount of addressable devices (4,294,967,296 addresses, to be exact). Remembering that back then there were only so many universities and defense organizations that would need an IP address. The idea was to have a connection of computers that would provide information to anyone connected to it. The logic back then was "why would anyone ever want to limit the freedom of information exchange?"

Along came private companies that connected to the Internet. Soon, others followed and more and more became all part of the Internet we see today. However, since security wasn't part of the original plans for the Internet, we have many security challenges that keep us security experts nicely employed. Keep reading and you will have the skills you need to earn a good paying job as well.

You may have heard the phrase "Keep It Simple, Stupid", or KISS for short when it comes to planning or performing a task. One fundamental law of engineering is that "complexity causes failures" so you want to try and keep your design planning as simple as possible. In security, we just shorten the whole phrase to "Complexity Breeds Insecurity".

The **World Wide Web** has evolved into an extremely complex system of interconnected computers, weird wires & cables, sloppy software, and a few really bad people who want to take things that do not belong to them. This whole mess we think of as the web sits poorly balanced on top of an ancient architecture built when security was not an issue nor was it designed into the original plans. Think of the web as a single car train with millions of other train cars stacked onto of each other's roof, each using different construction material, with the original train half leaning off the tracks.

Yes, security professionals have our work cut out for us. You can be sure that employment in the security field will continue to rise in the years to come. Why? Because the Internet is a bullet train moving at light speed and expanding even faster. Too many people depend on the Internet each and every day to allow engineers time to rebuild the Internet from scratch. We can't repair the train's engine while the whole thing is still moving, so we lubricate the wheels a bit or we dust off some gauges. The Internet train has to continue running until the neighbors build some other new Internet using spare parts from the one we have now. Until then, work on your resume.

How the Web Really Works

The web seems pretty simple: you get onto the Internet, open a browser, type in a website URL, and the page appears. But the devil's in the details, and some of them can hurt you. So how does the web really work?

A quick trip to the fine folks who make standards for the web, the **World Wide Web Consortium (W3C, at http://www.w3.org)**, will teach you all you want to know about how the web works. Don't miss the history of the web at **http://www.w3.org/History.html**. The problem seems to be that definitions and standards might teach you how to be safe. But, probably not. The people who want to hurt you don't follow standards or laws.

Reality Check

So what really happens when you go to a website? Assuming you are already connected to the Internet, here's how it works step by step:

1. You open your browser (Explorer, Firefox, Chrome, Opera, Safari, or any number of other browsers out there).
2. You type in the **Uniform Resource Locator (URL)** into the browser's address line (the website address, like ISECOM.org; the real nerds often call them URI for **Uniform Resource Indicators**).
3. The website's URL is saved in the browser's history on the hard disk. Yes, there's a record of everywhere you've been, right there on your hard drive.
4. Your computer asks your default **Domain Name Server (DNS)** to look up the IP address of the website. The DNS connects the name www.ISECOM.ORG to the IP address of 216.92.116.13. Use www.Whois.net to locate detailed information on a web page and try a Reverse IP lookup to find the actual numbered address instead of the site names.
5. Your computer connects to the website's server, at the IP address it was given by the DNS, to TCP port 80 for "**http://**" websites, or TCP port 443 if you go to an "**https://**" secure web site. If you use https:// there are more steps like getting server certificates that we won't cover in this example.
6. Your computer requests the page you ask for (like History.html), or if you specify a folder the web server sends a default page, usually index.html. It's the server that decides this default file, not your browser.
7. Your IP address, the web page you are visiting and details about your browser are likely to be stored on the web server and/or proxy servers in between (see below).
8. The requested web page is stored in the browsers **cache**. Yes, there's a copy of every page you've visited, right there on your hard drive, unless the web server explicitly asks your browser not to store it (cache is often disabled for protected web pages served through HTTPS, or at least it should be).
9. Most web pages contain other elements, like pictures, ads, style sheets (instructions to your browser on how the page should be displayed), **Javascript** (little programs, to do checks or make the web page look fancy and smooth). All these elements are retrieved in a similar manner as typing a URL by yourself.
10. The browser nearly instantaneously shows you what it has stored in your browser's cache. There's a difference between "perceived speed" and "actual speed" during your web surfing. This is the difference between how fast something is downloaded (actual) and how fast your browser and computer can render the page and graphics to show them to you (perceived). Remember: Just because you didn't see web page elements doesn't mean it didn't end up in your browser's cache.

The World Wide Web (web) is a massive **client-server network**. Clients are typical users who run web browsers to display or capture Internet data. Servers are web servers, such as Internet Information Server (IIS) on Windows or Apache on Unix/Linux. So the browser asks for a page, and the web server returns content in the form of **Hyper Text Markup Language (HTML)** pages.

Security - Here to Save the Day

Where does security come in? Well, if you're running a public web server, it's sort of like a retail store window. The front window is a great place to advertise and display your goods, however you don't want that store window left open so passers-by can take things for free. Ideally, you'd make sure that if someone throws a brick, your window doesn't shatter, either! Unfortunately, web servers operate complex software, and you can't have complex software without getting some bugs as well. Less scrupulous members of society will exploit these vulnerabilities to access data they shouldn't have access to (like credit card information, your medical records, and those pictures of you with the black eye that your little sister gave you). More on these vulnerabilities later.

Your browser *will* have vulnerabilities as well, and to make things worse, the creators of your web browser has its own agenda. Some bad Web sites can inject malicious code into your browser, or employ malicious links that can infect your computer. And let's not forget social media sites that watch your every move as well as collect every picture you post and every word you type. Last year, one specific social media site sold every bit of data they collected on its users to seven different companies. That social media site didn't lose FACE with their users though, since this massive sale of personal data wasn't listed in many BOOKs at that time.

Exercises

10.1 Locate your browser's cache and temp files. Examine the directory and the files inside. What is in those directories? How could this data be used against you? Is there a way to prevent the browser from caching files, or to clear the temp and cache when you close the browser?

10.2 Where can you find your browser's history file? Look at that history. What does it tell you about yourself? Is there a way to prevent the browser from recording this history? How can you clear this history?

10.3 Most on-line businesses collect a lot of information about you, which may seem harmless yet it becomes a privacy concern over time. Install the Firefox add-on called Ghostery. What does it do? What does it tell you?

10.4 Locate your favorite web sites and try to learn as much as you can about them. Who owns those sites? Where are the web servers located (physical location)? How long have those IP addresses been registered and who is listed as the Registrant Name?

10.5 When does your favorite web site domain name expire and can you purchase that domain name when it expires?

Game On: Who Hacked The Website?

"It's not like you were going to be the only person with access to the police web site. I've known you for years now and I don't even trust you with my electronics," Mokoa reasoned with Jace. He subconsciously braced himself next to the junky old metal police desk just in case Jace decided to smack him again. That previous hit almost cost them their relationship. Luckily Jace had learned the value of their friendship then, and she wasn't about to screw that up again, ever.

Without looking up from the monitor Jace replied, "Agreed, but I didn't give anyone here any password information about their web site. I just set it up and figured that they wanted me to maintain it. Hell, I built their entire network here for free. Nobody ever asked me for admin rights or passwords except for individual user accounts. I am root."

As the two teens crowded into the former police interview room recently converted into a server area, the puzzled hacker behind the keyboard was more interested in recovering the web site post-attack. Mokoa was concerned about who had accessed the police web console and created a back-up in the public directory. Jace was working in the digital world while Mokoa was thinking in the real world. They were a good team.

A blast of fresh air hit the small room as Officer Hank pushed into the room with the teens. "Hey Jace, how's it coming along and who is this handsome guy next to you? Do I need to frisk him for weapons, like chocolate or flowers?" Hank asked in a fatherly manner.

Mokoa stood up in reaction to the unexpected police officer standing over him. Actually, Officer Hank was towering over Mokoa. Hank held out his meaty hands to either shake Jace's friend's hand or place it manly on his shoulder. Mokoa started to introduce himself, "Hi I'm Mo."

"Mokoa, yeah. I've seen you a few times. Jace tells me all about you. I think she likes you," Hank gave a light jab at Jace's shoulder to soften his joke about her liking Mokoa.

"Shut up you two, I'm trying to work here," Jace growled back. Both guys stepped back from the cranky girl and shared a laugh at her reaction.

Jace confronted Officer Hank while looking at the screen and not at him, "You wanna tell me who in this building has admin access to this network besides me? Or more to the point, who screwed up the web site? I worked my butt off trying to make this network and web content perfect for you cops and all I get is this mess."

The burly police officer interrupted Jace, "Hold on their Miss High and Mighty. First off, you forgot to give anyone here the passwords to update the site. Secondly, we needed to add more information to our network. And C, you haven't been around lately to help make those changes happen. So you need to just chill out and remember that I've got a wooden baton and you don't."

Jace, put in her place, Mokoa thought with a slim smile. *By a cop, too. I like this guy.*

The flustered teen flopped back into the creaky old desk chair. She explained what she knew about how the web attack happened, what she had to restore and how the issue needed to be corrected. Hank squatted down next to her listening with interest. Every now and then, Mokoa added additional details.

Officer Hank explained his situation, "The chief wants dynamic content as part of the public web site. He wants things like traffic web cams, weather reports, most-wanted criminals, community volunteer patrol details and all kinds of stuff. He wants our web to give the users real-time information about events as they happen."

Jace sat leaning against the dilapidated police interrogation desk to absorb all this.

She asked, “Okay, we can do all that. Now I need to know how you guys want me to implement the dynamic content. We could use Ruby on Rails, PHP, jQuery and HTML5 with push or pull data. What's your flavor?”

Hank just started back at Jace, blank look on his face.

A long pause, too long for Mokoa, was broken when Jace tilted her slender head and continued the conversation in a slower voice, “There are plenty of security concerns with any dynamic content we use. First, we have the top three application level issues. They sound like a crappy movie script.”

Mokoa looked away. Jace spun around to face the computer screen. “Our biggest three issues with dynamic content are SQL injection, cross-site scripting and leaving unnecessary services running inside web applications. SQL injection is passing SQL code into an application. Potential attack strings are built from fragments of SQL syntax. They get executed on the database server side if the Web application doesn't screen out potential code. For example, we can have problems if we don’t protect ourselves against malicious input like a single-quote character, which could close the SQL string and give the attacker unintended system and application access.”

Without taking a breath, Jace continued,” The easiest way to see how this works is to type this string for the e-mail address and password in a web form input:

```
' OR '1' = '1 
[note that there must be a space at the end]
```

“The condition will be satisfied by the always-true condition '1' = '1', the SQL statement will show us everything in the current table and the login will succeed.”

Jace tossed her hair out of her face and pressed forward: “In order to eliminate this vulnerability, we need to pass the name and password to the database in a way that special characters aren't interpreted as part of the SQL command. The easiest way to do this is to avoid constructing ad hoc SQL statements, and instead pass the e-mail address and password as parameters to a stored procedure. Basically we tell the SQL database what an email address should look like and only accept valid inputs.”

Hank and Mokoa hated feeling stupid around Jace but that was something they were both getting used to. The police office tried, “Great to hear that you know so much about these web security problems. But, can you fix all this stuff and give the chief the type of web site he wants?”

She squared up her shoulders like a soldier given a proper order. Jace answered back, “Hank, I’m on it. Don’t you worry. I’ll take care of everything, dude.”

Hank smirked and thought, *That’s what I’m worried about, dude.*

Game Over

Rattling the Locks

Assume you have put up your own web server to host your blog, personal website, photos and whatnot. How could all that information be exploited and used against you? Let's find out, shall we?

Standard HTML pages are transferred using **Hyper Text Transfer Protocol** (**HTTP**). It's a simple text-based system for connecting to a server, making a request and getting an answer. This means that we also can connect easily to a server using command line tools like **telnet** and **netcat**, and get information about what software is running on a specific server. Look at what you get when you run this simple two-line command:

```
netcat isecom.org 80      [now press  <Enter>]
HEAD / HTTP/1.0           [now press  <Enter> twice]
```

```
HTTP/1.1 200 OK
 Date: Wed, 29 Feb 2012 23:25:54 GMT
 Server: Apache/2.2.22
 Last-Modified: Tue, 07 Feb 2012 18:41:18 GMT
 ETag: "3dad-4b8641fe22f80"
 Accept-Ranges: bytes
 Content-Length: 15789
 Identity: The Institute for Security and Open Methodologies
 P3P: Not supported at this time
 Connection: close
 Content-Type: text/html
```

Every web server and version will return different information at this request – an IIS server will return the following:

```
netcat www.microsoft.com 80
HEAD / HTTP/1.0
```

```
HTTP/1.1 200 OK
 Connection: close
Date: Fri, 07 Jan 2005 11:00:45 GMT
Server: Microsoft-IIS/6.0
P3P: CP="ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo
OUR SAMo CNT COM INT NAV ONL PHY PRE PUR UNI"
 X-Powered-By: ASP.NET
 X-AspNet-Version: 1.1.4322
 Cache-Control: public, max-age=9057
 Expires: Fri, 07 Jan 2005 13:31:43 GMT
 Last-Modified: Fri, 07 Jan 2005 10:45:03 GMT
 Content-Type: text/html
 Content-Length: 12934
```

Getting More Details

You can take this further and obtain more information by using the "OPTIONS" modifier in the HTTP request:

```
netcat isecom.org 80     [now press <Enter>]
OPTIONS / HTTP/1.0       [now press <Enter> twice]
```

```
HTTP/1.1 200 OK
 Date: Fri, 07 Jan 2005 10:32:38 GMT
 Server: Apache/1.3.27 Ben-SSL/1.48 (Unix) PHP/4.2.3
 Content-Length: 0
 Allow: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, PATCH, PROPFIND,
PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK, TRACE
 Connection: close
```

This gives you all of the HTTP commands (or "methods") to which the server will respond in a specific directory (in this case "/", the root directory of the web server or "document root").

Notice that HTTP is completely "in the clear"; everyone can see what you're browsing for. Consider how this might be changed; look up and learn about "secure searching".

Doing all of this by hand is rather tedious, and matching it manually against a database of known signatures and vulnerabilities is more than anyone would want to do. Fortunately for us, some very enterprising people have come up with automated solutions like **nikto.**

Nikto is a **Perl** script that carries out various tests automatically. It runs a scan and provides a detailed report:

```
    ./nikto.pl -host www.hackerHigh School.org
- Nikto v2.1.5
```

```
---------------------------------------------------------------------------
+ Target IP:          216.92.116.13
+ Target Hostname:    www.hackerHigh School.org
+ Target Port:        80
+ Start Time:         2012-03-20 13:33:49 (GMT-6)
---------------------------------------------------------------------------
+ Server: Apache/2.2.22
+ ETag header found on server, fields: 0x2f42 0x4b8485316c580
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS
+   /cgi-sys/formmail.pl:   Many   versions   of   FormMail   have   remote
vulnerabilities, including file access, information disclosure and email
abuse. FormMail access should be restricted as much as possible or a more
secure solution found.
+ /cgi-sys/cgiwrap/~bin: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+ /cgi-sys/cgiwrap/~daemon: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+ /cgi-sys/cgiwrap/~ftp: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+ /cgi-sys/cgiwrap/~mysql: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+  /cgi-sys/cgiwrap/~operator:  cgiwrap  can  be  used  to  enumerate  user
accounts. Recompile cgiwrap with the '--with-quiet-errors' option to stop
user enumeration.
+ /cgi-sys/cgiwrap/~root: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+ /cgi-sys/cgiwrap/~sshd: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+ /cgi-sys/cgiwrap/~uucp: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+ /cgi-sys/cgiwrap/~www: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+ /cgi-sys/cgiwrap/~OYG3G: Based on error message, cgiwrap can likely be
used to find valid user accounts. Recompile cgiwrap with the '--with-quiet-
errors' option to stop user enumeration.
+ /cgi-sys/cgiwrap/~root: cgiwrap can be used to enumerate user accounts.
Recompile  cgiwrap  with  the  '--with-quiet-errors'  option  to  stop  user
enumeration.
+  /cgi-sys/cgiwrap:  Some  versions  of  cgiwrap  allow  anyone  to  execute
commands remotely.
+  /cgi-sys/Count.cgi:  This  may  allow  attackers  to  execute  arbitrary
commands on the server
+ OSVDB-3092: /readme.txt: This might be interesting...
+ OSVDB-3093: /cgi-sys/counter-ord: This might be interesting... has been
seen in web logs from an unknown scanner.
+ OSVDB-3093: /cgi-sys/counterbanner: This might be interesting... has been
seen in web logs from an unknown scanner.
+ OSVDB-3093: /cgi-sys/counterbanner-ord: This might be interesting... has
been seen in web logs from an unknown scanner.
+ OSVDB-3093: /cgi-sys/counterfiglet-ord: This might be interesting... has
been seen in web logs from an unknown scanner.
+ OSVDB-3093: /cgi-sys/counterfiglet/nc/: This might be interesting... has
been seen in web logs from an unknown scanner.
+ 6474 items checked: 1 error(s) and 22 item(s) reported on remote host
+ End Time:           2012-03-20 13:50:22 (GMT-6) (993 seconds)
---------------------------------------------------------------------------
1 host(s) tested
```

Note: Almost every one of these lines represents a possible vulnerability or exploitable code. Using various options you can fine tune nikto to do exactly what you need, including stealth scans, mutation and cookie detection.

Mightier and Mitre-er

Finding a vulnerability is all well and good but what you do with that information is a whole different story. Security professionals will take the scan results of their own web servers and patch, update, remove, repair or do whatever they need to in order to close each vulnerability. The nice folks over at **Mitre.org** operate several databases (http://mitre.org/work/cybersecurity.html) that collect and catalog every known vulnerability you could imagine.

These databases are given scary names like "**Common Weakness Enumeration (CWE)**" and "**Common Vulnerabilities and Exposures" (CVE)**" yet they are fairly simple to operate. These systems are a collection of other tools and data with a search engine built into each database. Each data repository is focused on different aspects of hardware, software, services, system configurations, and compliance requirements.

Looking at the results nikto gave us earlier, you can see near the bottom of the log are the letters **OSVDB** followed by a bunch of numbers. OSVDB stands for the **Open Source Vulnerability Database** located at OSVDB.org. In the log results from nikto the numbers after OSVDB identify a specific type of vulnerability.

Exercises

10.6 Install **netcat** on a Linux virtual machine or use a live Linux CD/DVD to boot up your computer:

`sudo apt-get install netcat` [for Debian/Ubuntu family]

or

`sudo yum install netcat` [for Red Hat/Fedora family]

10.7 Install nikto on your Linux virtual machine:

`sudo apt-get install nikto` [for Debian/Ubuntu family]

or

`sudo yum install nikto` [for Red Hat/Fedora family]

10.8 Repeat the experiments above, targeting www.hackerhighschool.org.

Note: Do not try this on any other public or private web server, including your school's. Bad, bad idea.

Browsing Behind Bars: SSL

When the web started to take off, it wasn't long before everyone realized that HTTP in plain text wasn't good for security. The next variation was to apply encryption to it. This came in the form of **Secure Sockets Layer/Transport Layer Security** (**SSL/TLS**, simply called **SSL**), a cryptographic suite encompassing secure ciphers implementing 40 to 128 bit (or more) symmetric key encryption methods. A 40 bit key is not as secure than a 128 bit key, and with specialized hardware, 40 bit is breakable within a reasonable period of time, like during a lunch break (okay, maybe a bit more than that). The 128 bit key will take much longer: cracking it with only brute force will require somewhere between a trillion years and the total age of the universe. A streaming cipher suite like RC4 only offers protection for about the square root of the key space, or half the length of the key; so a 128 bit key offers protection of 64 bits, which can be cracked on a modern PC in relative short time – think days. One thing to remember is that the stronger the key algorithm you use, the longer it will take to encrypt and decrypt code. Use encryption sparingly or only apply as much strength as you need for web surfing.

Along with SSL, an open source version is available called "**OpenSSL**" and can be found at openssl.org. OpenSSL works alongside Transport Layer Security to provide an entire library of cryptographic recipes. OpenSSL is a command line tool with many options to work with. Turn to http://www.openssl.org/docs/apps/openssl.html for all the latest information on the command interface and library updates.

For known HTTPS attacks there are more complex approaches using something called a **known cyphertext attack**. This involves calculating the encryption key by analyzing a large number of messages (over a million) to deduce the key. Along with **cyphertext**, you will find multiple attack methods that are discovered every day.

Applied SSL

You shouldn't rush to try and crack 128 bit encryption. Since SSL just encrypts standard HTTP traffic, if we set up an SSL tunnel, we can query the server just as we did earlier. Creating an SSL tunnel is a snap, especially since there are utilities like **openssl** and **stunnel** made just for the job.

Exercises

10.9 Macs and Linux machines have openssl already installed. Try this command:

```
openssl s_client -connect www.hackthissite.org:443
```

This connection will only stay open a few seconds, but you can run any HTTP command, like:

```
GET www.hackthissite.org/pages/index/index.php
```

10.10 Check to see if stunnel is installed on your Linux machine or on the live CD. If it's not installed, install it. Go ahead and give it a test drive.

10.11 Once it's installed or at least running on your machine, run the command:

```
stunnel
```

This will tell you the location of the configuration file you need to create.

10.12 At that location, create stunnel.conf and enter this text (replace *ssl.enabled.host* with the name of a real SSL server that you want to connect to):

```
client=yes
 verify=0 [psuedo-https]
 accept = 80
 connect = ssl.enabled.host:443
 TIMEOUTclose = 0
```

10.13 Once that's done you start stunnel with this command:

```
stunnel &
```

Stunnel will map your local computer port 80 (default HTTP port) to SSL/TLS port 443 (default SSL port) and uses plain text, so you can open another shell and connect to it using any of the methods listed above, for instance:

```
netcat 127.0.0.1 80      [hit <Enter>]
HEAD / HTTP/1.0          [hit <Enter> twice]
```

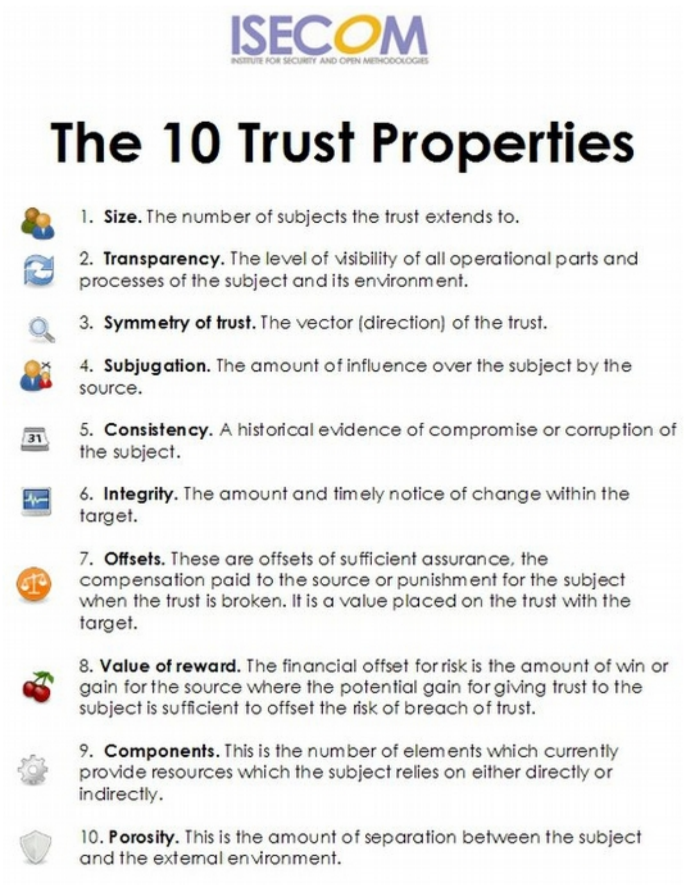
```
HTTP/1.1 200 OK
 Server: Netscape-Enterprise/4.1
 Date: Fri, 07 Jan 2005 10:32:38 GMT
 Content-type: text/html
 Last-modified: Fri, 07 Jan 2005 05:32:38 GMT
```

```
Content-length: 5437
Accept-ranges: bytes
Connection: close
```

The Properties of Trust

The whole SSL-thing is designed around trust. The browsers contain certificates which are really just long numbers that serve as keys. When you use SSL, the S in HTTPS, the certificate of the browser is used to check the certificate of the server and see if it's a trustworthy web server. The theory is that if the domain name matches the SSL certificate of the server as the domain that we visited then it's trustworthy. Of course who says it's trustworthy is a whole other problem. It wouldn't be the first time that a criminal infiltrated the certificate authorities and got the keys to fake their own server certificates that your browser then tells you is trustworthy. In another case, a known skeevy organization that was responsible for making spyware paid its way into being a certificate authority and added to all the browsers. So trust is a big deal on the web. It's another way to attack and it's another way that humans badly understand how to protect against it going wrong.

ISECOM spent time researching trust and discovered 10 main properties. You can read more about these properties in the OSSTMM. But basically, these are 10 things that need to be evaluated to have logical (not emotional, gut-feeling) trust:



ISECOM
INSTITUTE FOR SECURITY AND OPEN METHODOLOGIES

The 10 Trust Properties

1. **Size.** The number of subjects the trust extends to.
2. **Transparency.** The level of visibility of all operational parts and processes of the subject and its environment.
3. **Symmetry of trust.** The vector (direction) of the trust.
4. **Subjugation.** The amount of influence over the subject by the source.
5. **Consistency.** A historical evidence of compromise or corruption of the subject.
6. **Integrity.** The amount and timely notice of change within the target.
7. **Offsets.** These are offsets of sufficient assurance, the compensation paid to the source or punishment for the subject when the trust is broken. It is a value placed on the trust with the target.
8. **Value of reward.** The financial offset for risk is the amount of win or gain for the source where the potential gain for giving trust to the subject is sufficient to offset the risk of breach of trust.
9. **Components.** This is the number of elements which currently provide resources which the subject relies on either directly or indirectly.
10. **Porosity.** This is the amount of separation between the subject and the external environment.

Our parent organization ISECOM pioneered the field of trust analysis: the study of reasons we trust – and which reasons are actually good ones.

Figure 10.1: Trust Properties

Exercises

10.14 Look at Property 8. Do you think **reward** is a valid reason to trust? Do you think it is a GOOD reason to trust? Would you trust someone because they gave you money? And now: Do you use Gmail? Do you trust the company that provides it because it's free?

Employing a Middleman: Proxies

Using Someone Else's Server

A **proxy server** (or just **proxy**) is a middleman in the HTTP transaction process:

- The client (your browser) sends its request to the proxy;
- The proxy stops and holds your request, then sends its own request to the web server;
- The web server (which doesn't even know who you really are) responds to the proxy; and
- The proxy relays the response back to the client, completing the transaction.

A proxy can be a server on your own network that lets you pass your connection through it. This is handy because it gives you some protection, since the proxy hides your identity and acts as a firewall between you and the rest of the web. But you can also use a proxy server that's out on the Internet, which hides you even better (on-line you can find many lists of publicly available proxies, such as http://tools.rosinstrument.com/proxy/). These **external proxy servers** provide critical access to the outside world for people in countries that censor or cut off their ISP's connections to the Internet.

But, as you might have figured by now, proxy servers are vulnerable to attacks themselves, and can become jumping-off points for launching attacks on other servers. Not to mention that you may be going through one and not even be aware of it – which means **IT'S RECORDING EVERYTHING YOU DO**. This is something you especially want to consider if you use a free public proxy server. There is absolutely no guarantee the owner of that proxy is honest and will not use your username/password or credit card details. Schools and businesses usually have a proxy to enforce their “appropriate use” policies.

Exercises

10.15 Are you behind a proxy? How can you find out?

10.16 If you lived in a country that blocked access to some web sites, could you find external proxies that would let you access them? If so, wouldn't the government also look for them and block them? How might you overcome this?

Using a Local Proxy

You can run a proxy server right on your local computer. It won't change your source IP address (because its address is the same as yours), but it can prevent caching and filter out undesirable content.

Exercise

10.17 Find a piece of software called Privoxy. If you install it, what will it give you? Are you a candidate to use it?

The Onion Router

The Onion Router, or **TOR**, was created to hide your IP address – many times over. When you use the TOR network, your traffic gets encrypted and passed along through a tangle of routers, and eventually emerges … somewhere. But in theory your traffic can't be traced back to you. In theory. In reality, some things – like using Flash on TOR – has given unsuspecting users bad surprises.

Technically, you could set up TOR yourself, which involves some interesting configuration. We recommend it for the learning experience. However, most of us mere mortals will appreciate the **TOR Browser**, which has all the defaults set for safety, and lets you do all that interesting research in a separate browser.

Exercises

10.18 Who created TOR? Why?

10.19 Find out where you get it. Get it. Open it up and make it work.

HTML Programming: A Brief Introduction

HTML is a set of instructions that explains how information sent from a web server (Apache, IIS) is to be displayed in a browser (Firefox, Opera). It is the heart of the web. The **World Wide Web Consortium (W3C)** is the standards organization governing the workings of HTML.

HTML can do much more than just display data on a web page. It can also provide data entry forms, for server-side processing by a higher level language (Perl, PHP, etc). In a business setting this is where HTML is most useful, but in a hacker setting this is where HTML is most vulnerable.

Note: HTML is supposed to be a standard format across all platforms (browsers and operating systems), yet this isn't always the case. Some browsers read the HTML slightly different than other browsers just as other OS's won't be compatible with other operating systems. Be sure to cross check your HTML code against other types of browsers to ensure it is interpreted correctly and doesn't show up as a pile of gibberish.

Reading HTML

HTML works by using **tags** or **markup**.–Most opening tags, **<h1>**, for instance, must have a closing tag, **</h1>**. Just a few tags have no closing tag, like **<br>** and **<img>**.–The markup **<h1>** tells the browser where to start and stop displaying a large, bold heading, for example. **Well-formed HTML** has proper opening and closing tags (and follows other rules as well). One factor to consider in coding (writing) HTML is the need to be read by several different platforms.

Take, for example, the code:

```
<html>
<head>
 <title>My Hello World Page</title>
</head>
 <body>
 <h1>Hello World!</h1>
 <p>I'm a new page.</p>
 </body>
 </html>
```

We are telling the browser this is an HTML document with the tag **<html>**, and inside the **<head>** we give it the title "My Hello World Page" with the **<title>** tag. The My Hello World Page tag tells our browser "here's the information to show to the viewer." Finally, the **<h1>** tag tells the browser to display the information in "Heading 1" style, and the **<p>** tag tells the browser "here's a regular paragraph." The tags that have a "/" are the closing tags, which tell the browser to stop displaying the contents as described by the opening tag.

Exercise

10.20 Search the W3C.org website for the HTML standard. Are the HTML5 standards in the same place? Why wouldn't they be?

10.21 Copy the code above and paste it into a text file called **hello.htm**. Open that file in your browser of choice and you should see something similar to this:

Hello World!

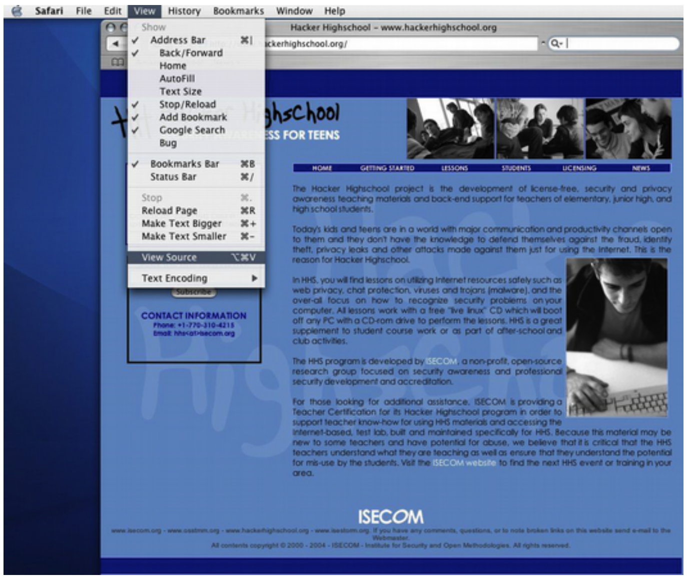
I'm a new page.

Viewing HTML Source Code

All browsers contain a way to view the underlying HTML code that generated the web page you are looking at. In most cases, this is the **View Source** option under the **View** menu in your browser. Many browsers will also show source code if you press Control-U or Command-U.

Exercise

10.22 Go to a web page you often visit. View the source code.



The Hacker Highschool project is the development of license-free, security and privacy awareness teaching materials and back-end support for teachers of elementary, junior high, and high school students.

Today's kids and teens are in a world with major communication and productivity channels open to them and they don't have the knowledge to defend themselves against the fraud, identity theft, privacy leaks and other attacks made against them just for using the Internet. This is the reason for Hacker Highschool.

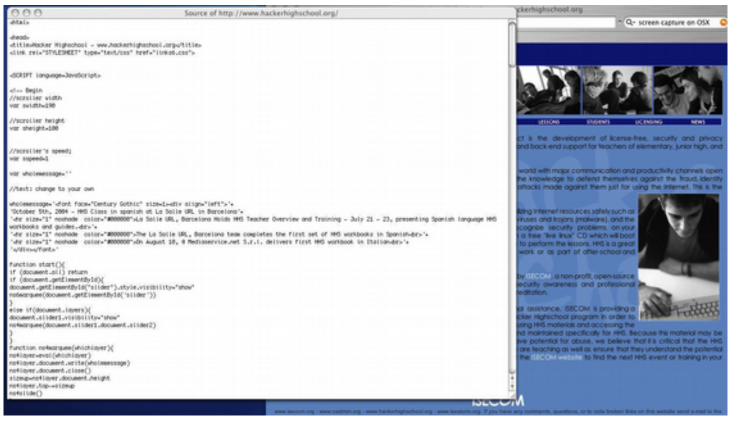
In HHS, you will find lessons on utilizing Internet resources safely such as web privacy, chat protection, viruses and trojans (malware), and the over-all focus on how to recognize security problems on your computer. All lessons work with a free "live linux" CD which will boot off any PC with a CD-rom drive to perform the lessons. HHS is a great supplement to student course work or as part of after-school and club activities.

The HHS program is developed by ISECOM, a non-profit, open-source research group focused on security awareness and professional security development and accreditation.

For those looking for additional assistance, ISECOM is providing a Teacher Certification for its Hacker Highschool program in order to support teacher know-how for using HHS materials and accessing the Internet-based, test lab, built and maintained specifically for HHS. Because this material may be new to some teachers and have potential for abuse, we believe that it is critical that the HHS teachers understand what they are teaching as well as ensure that they understand the potential for mis-use by the students. Visit the ISECOM website to find the next HHS event or training in your area.

Figure 10.2: View menu

The results should be something similar to this:



```
<html>

<head>
<title>Hacker Highschool - www.hackerhighschool.org</title>
<link rel="STYLESHEET" type="text/css" href="links6.css">


<SCRIPT language=JavaScript>

<!-- Begin
//scroller width
var swidth=190

//scroller height
var sheight=100


//scroller's speed;
var sspeed=1

var wholemessage=''

//text: change to your own

wholemessage='<font face="Century Gothic" size=1><div align="left">'+
'October 5th, 2004 - HHS Class in spanish at La Salle URL in Barcelona'+
'<hr size="1" noshade  color="#000000">La Salle URL, Barcelona Holds HHS Teacher Overview and Training - July 21 - 23, presenting Spanish language HHS
workbooks and guides.<br>'+
'<hr size="1" noshade  color="#000000">The La Salle URL, Barcelona team completes the first set of HHS workbooks in Spanish<br>'+
'<hr size="1" noshade  color="#000000">On August 18, @ Mediaservice.net S.r.l. delivers first HHS workbook in Italian<br>'+
'</div></font>'

function start(){
if (document.all) return
if (document.getElementById){
document.getElementById("slider").style.visibility="show"
ns6marquee(document.getElementById('slider'))
}
else if(document.layers){
document.slider1.visibility="show"
ns4marquee(document.slider1.document.slider2)
}
}
function ns4marquee(whichlayer){
ns4layer=eval(whichlayer)
ns4layer.document.write(wholemessage)
ns4layer.document.close()
sizeup=ns4layer.document.height
ns4layer.top-=sizeup
ns4slide()
```

Figure 10.3: Source code

HTML code is visible to anyone with a web browser. This is why it is very important, when you create web pages, not to hide passwords or sensitive information in the HTML source code (including comments). As you can see, it's not very secret.

In most browsers you can also view the source by doing a right mouse click and select "view source". The web page can instruct your browser to behave different on a right mouse click. On some sites this is used as a (poor) security measure to prevent visitors from viewing the source. When you select view, view source, it always works on older websites. With some more modern sites, the page source is dynamically and only partially updated. You need additional add ons in Firefox to view such a dynamic source (install the Add-on "Web Developer" in Firefox and the select: View Source, View Generated Source).

Linking Up With Hyperlinks

Links, or **hyperlinks,** are really the heart of HTML. The ability to link from document to document allows you to create **hypertext**, textual knowledge connected to related knowledge. A link, in HTML code, looks like this:

```
<a href="www.yahoo.com">www.yahoo.com</a>
```

This creates a link that looks like www.yahoo.com on your website, and of course will send your visitor away to Yahoo.

Links can be followed and checked by **link checker** programs. These programs search HTML source code for <a href="..."></a> tags and then create a file or index of the links they find. Spammers use this technique to harvest email addresses or identify contact forms they can use to spread their mass emails. Link checkers can also be used to check your website for broken links that don't go anywhere. Due to the dynamic nature of the web, these are depressingly common even in relatively small sites.

Exercises

10.23 Create a link to www.hackerhighschool.org that displays as "Hacker Highschool" on your web page.

10.24 Find and download a link checking program. Run that program against www.hackerhighschool.org and document how many broken links you find.

HTML5

The new flavor of HTML, HTML5, brings a lot of security improvements. It doesn't require Flash to stream videos, which is a huge leap in terms of preventing unwanted tracking and the endless vulnerabilities of Flash.
On the other hand, is carries a whole stack of new problems with it.

Cross Domain Messaging or **Web Messaging** lets HTML5 escape the ugly hacks we've used in HTML4 when documents come from more than one source. The problem is, this kind of messaging requires a lot of trust – and some very secure coding – to ensure nasty things don't get into the middle of this process.

Cross Origin Resource Sharing is when a web server allows its resources to be used by a separate domain. Again, this is a cool way to **mash up** content from multiple sources – but this level of trust just begs to be cracked.

WebSockets provides asynchronous full duplex communication. That's a mouthful, but essentially it means your browser can bypass the usual security measures in return for pure speed.

Local Storage APIs let web pages store data on your computer. Did you know all contemporary browsers include a mini-database called SQLite? Now just ponder: what's stored on your computer in that database? All kinds of interesting things about you, right?

Exercise

10.25 Find an application that lets you look into that SQLite database. Hint: you're looking for a "database browser." Install it and see what's in there!

Scripting Languages

Old-school coding in languages like C++ meant hours of deep coding, then compiling **binaries**, machine-language instructions that run very quickly. If you want raw horsepower, and have a whole lot of time on your hands, this is the world for you.

The rest of us will use **scripting languages** to write **scripts**, programs written in plain text that are interpreted at runtime by binaries (that you don't have to write) underneath. They don't run as fast as freestanding binaries, but with the very fast processors we have today, you may never notice the difference.

Scripting languages are great for dynamic web pages, but they also create a new avenue of attack for hackers. Most web application vulnerabilities aren't caused by bugs in any particular language, but by bad coding and poor web server configuration. For example, if a form requests a zip code but the user enters "abcde", the application should return them to the form and point out the error. This is called **input validation**. Here are some of the most common scripting platforms today:

Common Gateway Interface (CGI): This is the granddaddy of scripting interfaces, and it's not really a language itself; it's a way to run scripts. **Perl** was one of the most popular scripting languages to write CGI programs in the early days, though it's not used much for web pages anymore. Perl is, however, very useful for hackers, and many handy tools are written in this language.

PHP: PHP is a very popular open-source scripting language that runs on the server before the page is sent to the user. The web server uses PHP to get data from databases, respond to user choices and build a dynamic page with the information the visitor wants. HTML displays static content; PHP lets you create pages that give the user dynamic, customized content based on their input. Web pages that contain PHP scripting usually have a file name ending in ".php".

Python: Another popular language, Python is a competitor to PHP, and does many of the same things. Many web sites use both PHP and Python (as well as other languages), including Google.com, Yahoo.com and Amazon.com. Python scripts usually have the file extension .py. In the world of security, you ought to know as much as you can about at least one language. The flavor used today for security pros is Python.

Active Server Pages (ASP): Web pages that have a .asp or .aspx extension (ASPs) are database-driven and dynamically generated just like PHP or Python pages. ASP was Microsoft's first server-side scripting engine for the web. Its popular successor, ASP.NET, is built on the Common Language Runtime (CLR), allowing programmers to write code using any supported .NET language, such as: C#, VB.NET, Jscript.NET, etc. If you love Microsoft you'll love writing code specifically for IIS.

Java Server Pages (JSP): It is a technology that helps software developers create dynamically generated web pages. JSP is similar to PHP, but it uses the Java programming language. To deploy and run, a compatible web server with a servlet container (such as Apache Tomcat) is required.

Coldfusion and **Ruby** have their own cult followings, and there are dozens of less-well-known languages that can do very interesting things.

Javascript: Javascript (which is very much *not* the same thing as Java) probably runs on more web pages than any language besides HTML. It's different from the scripting languages above, because it doesn't run on the server to generate a page. Instead, it runs in your browser after the page arrives. This gives you visual effects like fly-out menus, expandable and collapsible sections of pages and "live" interaction with the page. Practically every dynamic page uses Javascript somewhere, and it's the front line of defense for validating the information people submit via forms. However, note that client-side input validation only is not enough to guarantee protection against attack targeting dynamic web application parameters. One example of ways to abuse client-side scripting is to pull up a web page, fill in the form, and capture it with a specialized proxy. The proxy lets you rewrite the code in the returned page and send it back using bogus values. You could also do this by "saving" the web page, editing the code, then sending it back using netcat. Or you could simply disable Javascript in your browser, using either built-in controls or add-ons.

Web Vulnerabilities

Giving someone what they ask for is simple; selling them something is a lot less simple. Online stores, companies selling products, bloggers selling ideas and personality, or newspapers selling news – all require more than just HTML-encoded text and pictures. Dynamic web sites that market products based on your preferences, show you alternatives, recommend other options, up-sell add-ons and make sure you pay for what you get require complex software. They're no longer static web sites, they're web applications. When we say goodbye to web sites and hello to web applications, we enter a whole new world of security issues.

Lions, Tigers and Bears Oh My!

We mentioned earlier "complexity breeds insecurity". Now we are going to look at the true meaning behind that mantra. When the automobile was first invented, it was nothing more than a wheel, one simple wheel. Some folks thought that the wheel was just fine but other people wanted to improve on that wheel. These early hackers began to add more wheels together attached with an axle and a frame. Others added seats to the frame and still more added a fancy horn or installed a horse as the engine. As time went on, this simple wheel automobile slowly took shape into a fast multi-horse drawn carriage. Luckily, someone added a brake before too many other got hurt. Today, we have airbags, seat belts, V-8 engines, and really great sound systems in our cars without having to smell horse dung all the time.

The Internet's evolution has progressed in a similar fashion. People weren't happy staring at a green display screen so they added a color monitor. The old dreary ASCII text became flashing lights and splash colors with midi sounds giving way to 3-D surround sound. Each new upgrade to the Internet added a new level of complexity, another layer of vulnerabilities to consider.

Let's take a ride on the Vulnerability Train to see some of the threats in their natural habitat.

SQL Injection - The Lion

A **SQL injection** isn't so much as an attack against a web site but rather an attack to gain access to databases behind the web site. The primary purpose of a SQL injection is to bypass the web pages. An attacker will want to gain either super user privileges to the databases associated with the web site or get the web page to dump database information into the attacker's hands.

SQL is the basic building block for databases. SQL commands will perform whatever task it is asked, including giving up passwords, credit card numbers, and so forth. All the attacker is doing is adding rogue SQL code (injecting) to any open form field on a web page.

For example, a web page asks a user if they would like to sign up for a monthly newsletter. The page will have open fields for the user to enter their name, email address, and whatever else the web builder wants. Each open field allows a user to input text, which is then stored in a database. A SQL injection simply requires an attacker to input SQL commands into the open fields. If the open fields are not protected (parsed) against allowing such commands, the attacker can easily type a request into an open field for a password list.

Here is an example of such a SQL request that has been slightly sanitized for your protection.

```
<?php
$query  = "SELECT '1', concat(uname||'-'||passwd) as name, '1971-01-
01', '0' from usertable;
           WHERE size = '$size'";
$result = odbc_exec($conn, $query);

?>
```

SQL injections are the most popular form of attack vectors. This attack can be rendered useless by correctly filtering SQL escape characters in user input fields or building your web site without using SQL. Don't forget about the URL field, that's a user input field too! Not to mention HTTP headers, cookies and more.

Buffer Overflows - The Tiger

Think of a buffer as a small cup. This buffer cup holds a certain amount of data and will spill if too much data is added. When the buffer cup overflows, because someone or something tried to add too much data, the system behaves strangely. When a system behaves strangely, other weird things can happen. A **buffer overflow** is an attack (or accident) where too much data is forced into a buffer that was only built for a certain amount or type of data.

When we look to the same open fields used in SQL injections, we can insert massive amounts of gibberish into fields that were designed to handle 25 characters. What happens when we add 1 million characters to that same field? The web page goes a little crazy and can provide an entry point to an attacker.

Buffer overflows can be avoided by limiting the amount of acceptable data to each open field and ensuring rogue code/commands can't be run inside these user input fields.

Cross Site Scripting (XSS) - The Bear

Our first two vulnerabilities were direct attacks against a server. **Cross Site Scripting (XSS)** is a client side vulnerability that exploits a user's trust to gain access into the web servers the client is looking at. A good example of this type of attack is when an attacker hitch hikes on a users browser while that user is logged in to a security site, such as a bank. The attacker preys on the users established trust between the user and the bank to gain access.

The attacker has various methods to piggy-back on users browsers. Some attackers will establish a **Phishing** web site or create a web site that looks identical to the one the user would visit. Since few people pay any attention to the URL field in their browser, it is fairly common for an attacker to lure a victim into allowing malware to be installed in their browser. Email is the preferred method to install malware since humans are, well, human. Java is another application that can be used to install malware from a malicious web site.

The attacker uses the trust of that user when they log onto an HTTPS site or any useful site the attacker would like to gain entrance into. One method you can use to protect yourself against this type of attack is to keep your browser updated and disable scripting in the browser "options". Based on your type of browser, there are add-ons available that can eliminate URL redirecting and stop any script that may appear dangerous.

Exercise

10.26 Go to http://scriptalert1.com/. Set up at least one tool from that page. Be prepared to explain what it does, how it's installed or set up and what security issue it addresses.

Bear Traps and Tiger Rugs

Each web application event and every web site you visit can bring potential harm to you or your loved ones. This doesn't mean that you should unplug from the Internet and go back to crayons in some remote cave somewhere. Each minute of each day new threats are presented and some are more successful than others. However, as a security professional, it is (or will be) your job to learn as much as you can about emerging threats and how to protect against them.

Honey pots can be used to lure bad guys into exposing themselves or track them as they probe your network. It is also common practice to apply maximum security and privacy settings to your firewall. You are using a firewall, aren't you? If you are not using a firewall, stop reading immediately and install a firewall this very instant. Once installed, max out the security and privacy settings. Your network router can be used as a firewall, as you should use both hardware and software firewalls. Don't forget the malware protection software (anti-virus).

Next on the list of self preservation is using anomaly detection software, otherwise know as **Intrusion Detection Systems (IDS).** An IDS can be as simple as a program running in the computer's background that looks for changes to system files. More sophisticated IDS's will perform more sophisticated actions, like sounding alarms or rerouting an ongoing attacker into a honey pot. We'll talk more about these later on.

Some often overlooked aspects of security is cyber law knowledge, compliance requirements, auditing, and knowing what government regulations you need to adhere to. Yes, those do sound lame but having a basic understanding of these areas can prevent you from making the international news headlines. Jurisdiction is a constant problem when an attacker is based in another country. Knowing what you can do and what you can't do to pursue an attacker is valuable information.

Exercises

10.27 What can happen if a web application doesn't perform validation on the data that the user inputs?

10.28 What are Cross-Site Scripting (XSS) and SQL injection? How does an attacker use them to steal data or gain access?

10.29 What security and privacy settings do you have on your own system? What about the rest of your family's computers?

10.30 Look up at least two different methods to mitigate SQL injections and describe them as a feasible addition to a web host. Tell us what they mean and how they work.

10.31 Do a Google search on the terms "inurl:search.asp" or "inurl:search.php". How many results do you get?

10.32 Find a search form at www.hackthissite.org, and type in:

```
        <script>alert ("hello")</script>
```

What happens?

Now, think: what happens if you try this on other sites? What kind of peril are you in if you do this?

10.33 Do a Google search on the terms "inurl:login.asp" and "inurl:login.php". Once again, consider how web sites invite attack simply by using these names.

10.34 Back on www.hackthissite.org, find a login form and attempt to type in special characters (@#$^&) for both the username and password. What happens? Do you think this kind of probing is logged?

Using a Local Proxy for Web Application Testing

An external proxy isn't the only kind of proxy server. You can also install a proxy server right onto your own computer, where it can perform some of the same firewall and filtering functions as an external proxy. Proxies designed specifically for testing web applications can manipulate data requests to test how the web server will respond. That means that with a proxy testing utility (such as Burp Suite, SpikeProxy etc.), you can run various cross-site scripting attacks, SQL injection attacks and almost any other direct request attack. While some of these proxy server utilities have an automation feature, the best tester is a real person behind the keyboard.

Remember, these tools are designed for you to test your own web applications! Be sure to test only websites for which you have permission, or you will most certainly be logged, and possibly jailed.

Exercises

10.35 Find and download **Burp Suite**. How do you install it? How do you run it?

10.36 Change your browser settings to point to the new proxy. This is usually localhost port 8080, but since these tools may be updated read the instructions to be sure. (Tip: the IP address 127.0.0.1 is the localhost address.)

10.37 Pull down the Help menu and select User Guide. Take a look at the documentation.

10.38 Now spend some time browsing the web site you're testing, in this case www.hackerhighschool.org. Test out forms and visit every page. You're creating a recording, so try to be thorough!

10.39 Once you have surfed around with your browser, go back to the ZAP interface and save your session.

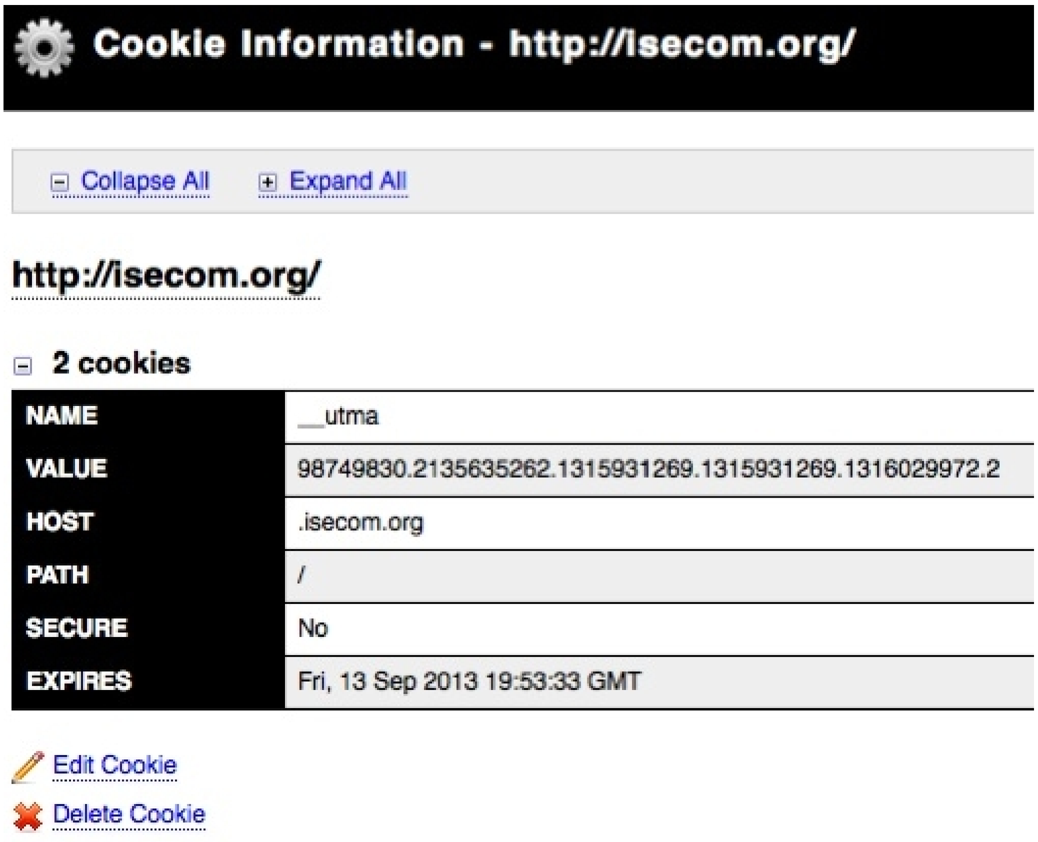
10.40 Right-click on the www.hackerhighschool.org listing on the left, and note the types of attacks you can run. Try them.

A proxy server can be a powerful tool in helping you determine how solid a web application is. For penetration tests or vulnerability assessments, a proxy utility is a good tool in your toolbox. But it's not the only one.

Looking Behind the Curtains

There is a lot going on that's not visible when you request a web page. Long before a page loads, **headers** and **cookies** are flying back and forth, and if you know how, you can see them.

Cookies are usually harmless, and only hold things like a unique user ID, so that when you visit that famous online retailer, the site already knows who you are and what you like.


Cookie Information - http://isecom.org/

http://isecom.org/

2 cookies

NAME	__utma
VALUE	98749830.2135635262.1315931269.1315931269.1316029972.2
HOST	.isecom.org
PATH	/
SECURE	No
EXPIRES	Fri, 13 Sep 2013 19:53:33 GMT

Figure 10.4 Cookies

HTTP headers can be confusing. HTTP is Hyper Text Transfer Protocol: notice the **P**! It's a transmission protocol, a way to send signals down the wire.

Headers get transmitted invisibly with every web request, and usually you never see them. Here's what they look like:

```
http://en-us.fxfeeds.mozilla.com/en-US/firefox/headlines.xml

GET /en-US/firefox/headlines.xml HTTP/1.1
Host: en-us.fxfeeds.mozilla.com
User-Agent:  Mozilla/5.0  (Macintosh;  Intel  Mac  OS  X  10.7;  rv:11.0)
Gecko/20100101 Firefox/11.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
X-Moz: livebookmarks

HTTP/1.1 302 Found
Content-Encoding: gzip
Cache-Control: max-age=604800
Content-Type: text/html; charset=iso-8859-1
```

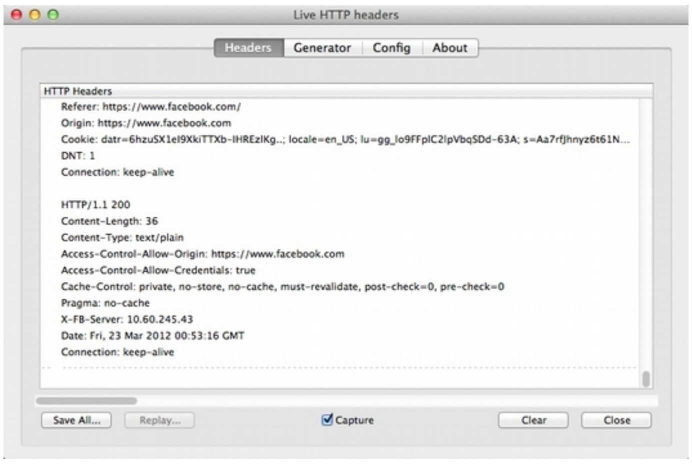
```
Date: Fri, 23 Mar 2012 00:50:26 GMT
Expires: Fri, 30 Mar 2012 00:50:26 GMT
Last-Modified: Sun, 18 Mar 2012 10:39:59 GMT
Location: http://fxfeeds.mozilla.com/firefox/headlines.xml
Server: ECS (lax/2872)
Vary: Accept-Encoding
X-Backend-Server: pm-web01
X-Cache: 302-HIT
X-Cache-Info: cached, cached
Content-Length: 200
----------------------------------------------------------
http://fxfeeds.mozilla.com/firefox/headlines.xml

GET /firefox/headlines.xml HTTP/1.1
Host: fxfeeds.mozilla.com
User-Agent:  Mozilla/5.0  (Macintosh;  Intel  Mac  OS  X  10.7;  rv:11.0)
Gecko/20100101 Firefox/11.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
[and many more...]
```

The HTML programming language (notice the **L**!) has a part of the document called the **<head>**, but that is *not* the same thing; it's where hidden things like scripts or meta tags go.

Even worse, HTML has **headings**: <h1>, <h2> etc. These are yet a third thing, bold headings for documents! We apologize for the confusion, but you do have to understand the difference.

So when we're talking about HTTP headers, we're talking about the hidden, behind-the-scenes chatter that happens before your page is even sent.

You can use tools like the Firefox add-ons **Live HTTP Headers** and **TamperData** to capture headers, which gets you a display like this:



```
Referer: https://www.facebook.com/
Origin: https://www.facebook.com
Cookie: datr=6hzuSX1eI9XkiTTXb-IHREzIKg..; locale=en_US; lu=gg_lo9FFpIC2IpVbqSDd-63A; s=Aa7rfJhnyz6t61N...
DNT: 1
Connection: keep-alive

HTTP/1.1 200
Content-Length: 36
Content-Type: text/plain
Access-Control-Allow-Origin: https://www.facebook.com
Access-Control-Allow-Credentials: true
Cache-Control: private, no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-FB-Server: 10.60.245.43
Date: Fri, 23 Mar 2012 00:53:16 GMT
Connection: keep-alive
```

Figure 10.5: LiveHTTPHeaders

You can use other tools, like **Add N Edit Cookies** or the **Web Developer** add-on, to capture the cookie information your browser is exchanging with the web site you're visiting. Notice that you can also edit that information. Finally, local proxies can also be used to intercept, analyze and modify cookies.

Exercises

10.41 Find and install the LiveHTTPHeaders add-on for Firefox. How do you use it?

10.42 Find and install the Web Developer add-on as well. How do you use it to view and edit cookies?

10.43 If you think a web application uses the User-Agent for some purpose (why do they care what browser you're using?), check what happens if you remove it. Most applications don't anticipate a missing User-Agent, so when they inspect the content, they're referring to a missing variable. In most cases the application throws an unhandled "object not set" exception, which can include stack dumps or even source code snippets (for instance, with .Net apps with the debugging option on). Sometimes this gives very interesting results.

Building Secure Web Applications

Have you ever considered building a web site? Selling something online? Building a web community? In some ways, it's getting simpler all the time. But in terms of security, it just keeps getting trickier.

Secure programming starts with the programmers. Each has his or her own logic and skills with particular languages. But there are accepted guidelines that a programmer determined to write secure code can follow, like the ones derived from the OSSTMM (**http://osstmm.org**):

1. Security should not require user decisions.
2. Every input and output in the application should have a business justification.
3. Quarantine and validate all inputs including the application's own content. Most especially, re-validate ***all*** the data server-side. Someone, somewhere, will try the trick of sending you bad data.
4. Tightly limit the systems and users the application trusts.
5. Encrypt data stored and in transit.
6. Create hashes of all the application's files, so you can check for unauthorized changes.
7. Make sure that all interactions occur on the server side.
8. Layer the security.
9. Invisibility is the best defense. Show only the service itself.
10. Build in triggers for alarms.
11. Promote security awareness in users and technical support people.

Exercises

10.44 What kind of error messages have you seen online? Do you remember where you got them, or how? Could you do it again?

10.45 Name three ways you could build an alarm into a web application.

Protecting Your Server

Consider what it's like to run your own server. You can set it up to do anything you like. Interested in Java or PHP? Joomla or OpenWiki? Photography, animals or both? That's the upside. The downside is that unethical people will try to do unethical things to your server.

You'll have to take some steps to protect your server. These include ensuring that your OS software is always up-to-date with patches, updates and service packs. And do the same thing with your web server software, your database software, your firewall, your antivirus, your intrusion detecting system, all of your installed software (which makes a good point: the less you install, the better!). There are many sources for correct configuration of servers (also called **hardening**), scattered around the Web. If you are running a server, search for those sources and read through some of them.

Firewalls

Firewalls originally were fireproof walls in a building or used between your car engine and the occupants (old school stuff). We use the same words for systems (either hardware and software) that are designed to prevent unauthorized access to a network. Just as a firewall in a building can be a matter of life and death, network firewalls are critical to the health and survival of computer systems.

The key to understanding firewalls is that they're based on sets of **rules**, commonly called **ACLs (Access Control Lists)**. Imagine the rules for a security guard at a school: anyone with a school ID can enter, but no one else can; anyone can exit, except that small children have to be with a parent. The rules for a network firewall are very similar, for instance:

1. Nobody from the Internet gets to come in, except by invitation.
2. Anybody from inside the network can go out to the web, except when this poses a risk.

Simple as that! (If only it were.) In the digital world, we have to work with three things: MACs, IP addresses, and port numbers. You're familiar with ports and protocols from Lesson 3. Here's where you can use this information: if you're running a web server, you need to open port 80 to allow HTTP traffic. If you've got a secure website, you'll need to open port 443. *But only these ports.* Anything else is an open invitation to attackers.

Also, unlike a security guard, a pure network firewall basically isn't very smart. It doesn't look inside of packets, so it can't tell what's being transmitted. That means you could do clever things like paste the contents of corporate documents into forms on a convenient website, and steal intellectual property.

At least, you used to be able to do things like that. The newest generation of "**smart firewalls**" (which are actually very, very smart proxy servers) can indeed tell exactly what's in every IP packet. Woe unto you if you try to sneak a stolen credit card number past this kind of system, because you will be not just detected but identified.

Which brings us to the subject of intrusion detection.

Intrusion Detection Systems/Intrusion Prevention Systems

Let's return to our school. It has good security guards at the doors and gates, but they could be fooled by a fake ID (and yes, you can fake or **spoof** your IP address). Also, the guards go home at night. We need another line of defense, in case somebody who shouldn't have access slips in: we need burglar alarms.

Intrusion Detection Systems (**IDSs**) are our alarm systems. They can detect the burglar, as well as call the police and ring a loud bell. In the Internet world this means noticing abnormal activity, realizing that something is wrong and sending email and text alerts.

Newer systems include **Intrusion Prevention Systems (IPSs)**, which are even more active: if an attack is detected, they can cut off traffic, block IP addresses and even launch countermeasures.

Exercises

10.46 What was the original popular IDS? What is its nickname?

10.47 If you had to pay a lot of money for either one, would you want your server to have a firewall, an IDS/IPS or both?

Secure Communications

What do we mean by "secure communications" on the web? The phrase sounds like something you'd hear in a spy movie, but actually it's critical to a lot of things we do online. Communicating securely requires a lot of **PAIN**: **Privacy** (and **confidentiality**), **authentication**, **integrity** and **non-repudiation**.

Consider your account at that really famous social network. Obviously, you have to have a user name and password so you can log in to the account. And there are all kinds of privacy settings you can tweak, especially about who can see your most private stuff, so you really don't want anybody else getting into it. Which means you're counting on **authentication** to give you some security. So, does authenticating yourself to the web site makes you safe?

No.

You might be completely deceived about the web site. Someone could send you a **phishing** email that looks exactly like a notice from that social network. Click that link, and it looks like you've gone to the site, but you've actually gone to a malicious site and gotten yourself infected with malware. Or you might make a simple mistake, like typing **whitehouse.com** instead of **whitehouse.gov** – and ending up at a porn site. (No, it's not there any more! But do check out **http://www.antiphishing.org**.)

This means authentication is a two-way street. You should have to identify yourself, certainly. But the site should provide authentication credentials too, and on the web, they come in the form of a **certificate.** When you go to a famous web retailer, and buy products online, your browser probably accepts the web site's certificate automatically, because it comes from one of the big-name, big-dollar certificate authorities like **Verisign.**

When you go to a less famous site, you may be prompted by your browser to accept or refuse its certificate. There is no more critical time on the web to read the details carefully! If you know you want to shop there, then perhaps you should accept the certificate. But if you have no idea why you're being asked to accept a certificate, then you probably shouldn't take it. The idea to understand is that a site's certificate provides its identification and authentication. You're trusting that ID.

So you've ordered that book (or whatever). You don't want the price changed after you order, naturally, or the number of books. And the retail website doesn't want anyone altering critical details like your credit card number. What you both want is **integrity**, which means that what you communicated hasn't been changed.

This is really critical when it comes to big-money transactions. No one should be able to change the dollar amount in the document you send to make an offer for a house – at least not without detection. And the way you do that is by having your computer calculate a **hash** from that offer document. A hash is the result of some complex math, but basically it's a big number that will be unique for every document.

On any Unix computer (which includes Linux and Macintosh), you can calculate an **MD5 hash** (that's the most common kind, although it's not considered that secure anymore and is therefore slowly being replaced by SHA-1, SHA-256 and similar algorithms) with a simple command. Let's say your letter is named Offer.txt. In a shell you can use the following command:

```
md5 Offer.txt
```

and you'll get back something like:

```
MD5 (Offer.txt) = d41d8cd98f00b204e9800998ecf8427e
```

The long string of numbers and letters (actually, it's a **hexadecimal** or **hex** number) is the hash. If someone changed so much as one period in that document, the hash would be totally different. Oh, so the hash you made before you sent the document no longer matches the document's hash now? Watch out! The document has been changed! And you might be buying a much more expensive house if you're not careful.

It's not just you that has to do the trusting in our real estate transaction. The people you send your letter to need to be certain you can't say, "I never sent that offer!" What they want is **non-repudiation**, which means that you can't deny making the offer. And if they accept your offer, you don't want them to repudiate their acceptance later. So how can we make sure of this?

Hashing does the trick here as well, though the process is more complex. You learned in another lesson about getting yourself a PGP/GPG key pair, which you can then use to create a **signature.** A signature works here because nobody but you can create that signature – and it will be different for every document you sign. At its root, that signature is just another hash.

Privacy and Confidentiality

Privacy means keeping your information yours, or at least maintaining control over it. This may be as simple as protecting your credit card number, or as complicated as keeping your new love affair secret.

What you do on the web stays private, right? You know by now that's not true. Web sites gather information about you constantly, particularly the search engine giants. You sign away your privacy when you accept that free email account, or when you join a social network. Web sites that collect information are supposed to have a **privacy policy** that clearly spells out how they will use your information. If you visit a site that wants information but doesn't have a privacy policy, *get out!*

When you visit, sites set **cookies** on your computer. Cookies aren't dangerous, in the sense that they don't install malware on your machine. In fact they're very handy for remembering your book preferences on Amazon.com. But cookies can be dangerous because they track your every move. Be particularly aware of this:

> When you log on to a website, you've given it permission to watch everything you do, everywhere you go, until you log back out! Even if you've left that site, if you didn't log out, it's still watching everything you do. Some, for instance that famous social network, track you even after you log out.

When you are done with a web site, to protect your privacy, *log out*. Ensure your browser is set to clear cookies, cache, temp files and history, then close the browser (not just the tab). Depending on the website and why you are there, you may wish to provide false information to avoid tracking. And you should be aware: some cookies (like Flash cookies) can survive "clearing" and can still be used to track you.

Confidentiality sounds very similar to privacy, but it's a different thing. If I don't know you have a crush on Betty or Bret, you've kept your privacy. If I know you're meeting behind the gym, your meeting is no longer private, but if I'm not watching it may still be confidential.

On the web, this means I may know you're visiting a site, but I can't read your communications because they're **encrypted** – scrambled so thoroughly you'd need a supercomputer and millions of years to decrypt them. When you visit a secure (HTTPS) web site, that's exactly what's it's doing. And it's using its certificate yet again to provide the **key** for that encryption. Is HTTPS totally safe? No, but it's the best we've got for now.

Exercises

10.48 Take a look at Google's **privacy policy**. Can you find it? Can you understand it? Compare it to the privacy policy of a retailer like Amazon.com. What about the privacy policy at social networking sites (SNS), such as Facebook?

10.49 Under these privacy policies, who can access your information? What can you do if the site violates its own policy? Can you take your information away?

Knowing If You Are Communicating Securely

If you have a private conversation, somebody could still find out about it. And even if it's confidential, somebody could be looking over your shoulder. (Search for “spy camera” to see why this is important.) So how can you be sure your communication is secure?

On the web, there are two major signals almost all browsers will give you that you are on a secure connection (using **HTTPS**, in other words). The first is the web protocol you see (or type) up in your URL address bar. If it's HTTP (which will look like **http://**), it is not secure, repeat, not secure.

You only have a secure connection when the protocol is HTTPS (so that the address starts with **https://**). While it's not visible, HTTPS is doing encryption, and unlike HTTP's port 80, HTTPS uses port 443.

The other visible cue in some browsers is a small **padlock icon** in the URL bar (or sometimes in the lower right corner of the window). If the lock is open or missing the connection is insecure, and if it's closed the connection is secure. Hover your mouse pointer over the lock, and in most cases a small message will appear telling you how long an encryption key is being used. Forty bits is very flimsy, while 128 bits is fairly stout, and the current standard.

Some newer browsers are using a feature like a “site button” - an area you can click near the URL address bar – which gives you security information about the site, though you may have to dig down to find the key length.

This is a good place to point out that while we use the term “secure connection” in this section, there is ultimately no such thing. The SSL/TLS encryption used by HTTPS is flawed and definitely can be cracked, if someone wants to crack it badly enough. You can probably trust it with Amazon.com (for now), but you should not trust it with your life. Why do we put it that way? Because in some countries, people literally do risk their lives accessing the web, and they should do it through a more secure technology called a **Virtual Private Network (VPN)**.

Methods of Verification

By now you've learned the basics of web security and privacy, and we hope you've thought about the things you could do as a hacker, or as the operator of a web site or server. Every decision you make will affect the security and privacy of real people – including you.

So how do you know if a server is secure, or a network, or an application? Should you trust the developers of the app or the OS? Will you be secure if you just keep your system updated?

The answers are, in order: Testing, No and No. We'll come back to testing, but when it comes to either applications or operating systems, manufacturers have proven over and over that you shouldn't necessarily trust them to get security right. (Read an **End User License Agreement** the next time you install software; you'll see that the manufacturer isn't responsible for anything at all!) And there are famous stories of updates that broke more things than they fixed, or introduced new vulnerabilities. Not to mention the **backdoors** that were discovered in some applications and operating systems!

When it comes to testing, the very best thing you can do is to think like a hacker. That's what this course is about. In other words, take advantage of other people's work. There are several security testing methodologies available, the results of contributions from hundreds of professionals. Just remember that any computer, network or application will change frequently, so test early and test often.

Methods arise from different philosophies. The **IT Infrastructure Library (ITIL)** is all about system life cycles; the **ISO 9000** standard deals with quality management systems. **ISECOM**, the organization that brings you Hacker Highschool, also provides a security testing methodology built on a open-source contributor model using open-source tools: the OSSTMM.

OSSTMM

The **Open Source Security Testing Methodology Manual (OSSTMM)** documents a widely-used and straightforward format for conducting security verifications. The individual tests in the OSSTMM aren't revolutionary, but the methodology itself allows anyone to conduct ordered tests with consistent professional quality.

The OSSTMM tests are divided into five sections:

- **Human Security**
- **Physical Security**
- **Wireless Security**
- **Telecommunications Security**
- **Data Networks Security**

You don't have to do every type of test; part of the methodology is doing only the relevant ones. This is where you can learn to be an expert: knowing which test to do, why to do it and when to do it.

To put it another way, the OSSTMM details the technical scope, but doesn't prescribe specific testing software. Instead, it describes:

- what should be tested
- the form in which the test results must be presented
- the rules for testers to follow to assure best results
- the **Attack Surface Metrics**, which put hard numbers on how much security you have

The OSSTMM is a document for professionals, but it's never too early to take a look at it and learn how it works. The concepts are thoroughly explained and are written in an easy-to-comprehend style.

Exercises

10.50 Patching is a necessary evil, and web administrators need to constantly patch code as new vulnerabilities are discovered. Search for a case in which a new problem arose from installing a security patch. Imaging you're a system administrator: would you argue for, or against, patching your systems immediately when the patches are released?

10.51 You've discovered that a patch will introduce a new vulnerability. Should the patch still be installed? Would it matter whether you have the source code or not?

10.52 Go to **http://cve.mitre.org** and find the button or link to search CVEs. When you find the right page, do a keyword search on "Apache". How many vulnerabilities are listed? (And consider: how many patches do you want to install? Is patching a realistic solution, or is it a cat and mouse game?)

10.53 Download a copy of the OSSTMM and review the methodology concepts. What principles can you learn from this methodology? Could you do a security audit based on the OSSTMM?

Feed Your Head: Understanding HTTP

Now that you have an understanding of how information flows, did you ever wonder how your web browser really asks for whatever it is you want to see online, and presents it to you?

What do you think the browsers ask for? What kind of information? We know that, depending on the website, we might have a few images, some animations, perhaps video. We also have text, and polls of different types. That is quite a lot of different things a web server has to send us, for us to properly enjoy a well formatted page, don't you think?

So, now you'll see EXACTLY what it is that your browser does, behind your back, so you can have a pleasant experience.

To help us in that task, we are going to install a Firefox extension called **httpfox**.

Httpfox monitors and analyzes all incoming and outgoing HTTP traffic between the browser and the webservers, so you get to see each individual request that is made.

You can download httpfox at https://addons.mozilla.org/en-US/firefox/addon/httpfox/

Restart your Firefox browser, and now on the bottom right side you get an extra icon that you should click in order to open httpfox.

Press the icon, and then open a new tab and visit a couple of different websites. You now have a list of all the requests that your browser made.

Clear all of it, close all open Firefox tabs, start httpfox again and open www.isecom.org. Hit stop, and let's go through the requests.

The first thing that gets sent out of your browser, happens once you hit the enter key after typing www.isecom.org, and that's exactly what we see in httpfox.

Your browser does a GET request (as you can see from the Request Header), in HTTP 1.1. We are also saying that we want a specific host in that address, host www.isecom.org.

Consider the situation of a shared service provider. On one IP address, you could have hundreds of websites. This is possible exactly because of thie "host www.isecom.org" part of the request. If this functionality wasn't present, we could only have one domain per IP address, because the browser would have no way of telling what it really wants.

Now, because not all browsers are created equal, our browser is going to introduce himself, so the server can be sure to reply with information that suits us.

Next, we are telling the web server the kind of content that we are willing to accept. In this case, our browser says it accepts code written in HTML, XHTML and XML and would like to receive the information in American English (en-us) and what kind of encoding. If you have been to the ISECOM website before doing this capture, you will also see that we sent information regarding a cookie. A cookie is a small piece of data kept in your browser that stores information about a particular website. Almost every website you visit uses cookies. This information is used to serve you better (when it's not being used by the dark side of the force) because once you visit a website, you can be authenticated and so the website owner can greet you by your name, and present you with his content the way YOU want to see it. Think about the personalization that you can have on your online email application, for instance. All of this is for the GET request you instructed your browser to perform.

Now we will run through just three more lines. If you go to the "Content" tab, you will see the exact HTML code that was downloaded from the web server as a response to our previous GET request.

After the META tags, we got the following lines of code (at the time of writing):

```
    <link rel="stylesheet" href="style.css" type="text/css"
media="screen" />
    <!--[if IE 6]><link rel="stylesheet" href="style.ie6.css"
type="text/css" media="screen" /><![endif]-->
    <!--[if IE 7]><link rel="stylesheet" href="style.ie7.css"
type="text/css" media="screen" /><![endif]-->


    <script type="text/Javascript" src="jquery.js"></script>
    <script type="text/Javascript" src="script.js"></script>
```

Look at the highlighted pieces of code. You will find that the next three requests our browser made were exactly to those files. Notice that if we had Internet Explorer 6 for instance, we would have downloaded style.i6.css instead of downloading style.css. Different browsers, different code.

The rest of the files downloaded are more of the same. The browser interprets the HTML code it receives and makes decisions based on what it receives on the contents it should request next, depending on your behavior (clicking, mousing over, etc.).

Think about what all this means from a hacker's perspective. Everything your browser shows you are interpretations of what it receives from the servers. So could we somehow intercept this traffic and change how a client interacts with a server and vice versa? Of course it does... in another lesson.

Exercise

10.54 Aren't you curious what those .css, .js, and .swf files are? Try searching for those file extensions to learn a bit more about them.

Going Deeper

While web hacking can go so much further because it's a huge, immense, monstrous field, you can't. You've reached the end of the line of what we can show you in this lesson. However, if web hacking is your thing, jump deeply into the OSSTMM and check out the hacking methodology for Web Applications.

Today's teens are in a world with major communication and productivity channels open to them and they don't have the knowledge to defend themselves against the fraud, identity theft, privacy leaks and other attacks made against them just for using the Internet. This is the reason for Hacker Highschool.

The Hacker Highschool project is the development of security and privacy awareness learning materials for junior high and high school students.

Hacker Highschool is a set of lessons and a practical means of making hackers. Beyond just providing cybersecurity awareness and critical Internet skills, we need to teach the young people of today how to be resourceful, creative, and logical, traits synonymous with hackers. The program contains free security and privacy awareness teaching materials and back-end support for teachers of accredited junior high, high schools, and home schooling. There are multiple workbooks available in multiple languages. These are lessons that challenge teens to be as resourceful as hackers, including safe Internet use, web privacy, researching on the internet, avoiding viruses and Trojans, legalities and ethics, and more.

The HHS program is developed by ISECOM, a non-profit, open-source research group focused on security awareness and professional security development and accreditation.

Hacker Highschool
SECURITY AWARENESS FOR TEENS

ISECOM