CS50 Cybersecurity

Overview

The Internet is full of security threats. One significant threat is the threat of **cyberattacks**, where hackers attempt to target computer systems and networks for malicious purposes. **Cybersecurity** refers to systems and practices that web sites and users can employ in order to better protect themselves against cyber threats. Users can help to protect themselves against cyber threats through a variety of means, including choosing more secure passwords and being mindful of spam email.

Key Terms

- cyberattacks
- cybersecurity
- phishing
- two-factor authentication
- SSL
- TLS

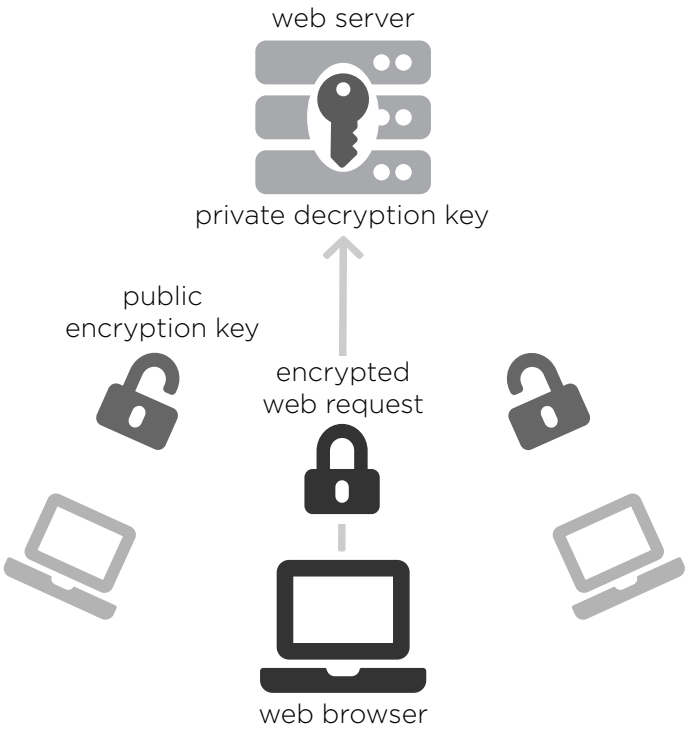
Passwords

Hackers can attempt to obtain passwords in various ways. One way is to try submitting millions of possible username and password combinations until one is successful. This is why choosing a longer and harder to predict password can improve security. Hackers may also attempt **phishing** attacks, where they send emails to users pretending to be a legitimate company and ask users to click on a link that asks for a password or other sensitive information.

Some services (including Google and Facebook) offer **two-factor authentication** as a means of combating possible password theft. Two-factor authentication requires two types of authentication that are inherently different, one factor would be a username and password, while the other factor can take the form a verification code sent via text to your phone or a security question or SecurID, which was a physical device that would generate a random 6-digit integer. Thus a user needs both their password and a secondary device in order to be able to login successfully. However, there is a tradeoff in convenience: if your phone is lost, or do not have access to your phone then you may be unable to access your account.

Website Security

HTTPS (with the "S" standing for "Secure") is a protocol for communication across the internet that combines HTTP and a technology called Secure Sockets Layer (**SSL**). Websites that use SSL each have a certificate, which is distributed to users who are trying to access the website. The certificate secures the connection between the server and the individual and also contains a public encryption key, which tells web browsers how to encrypt requests that are sent to the web server. The web server has another key, the private key, which can decrypt the encrypted requests. As a result, when a user sends an encrypted request to a web server using SSL, the information is more secure. You can generally tell which websites are using this technology by noting whether or not their URLs begin with `https://`



Today, a technology called Transport Layer Security (**TLS**) is more commonly used, but it is just an updated and improved version of SSL.

Other Cyberattacks

Hackers use several other techniques to perform cyberattacks. In a man-in-the-middle attack, a malicious piece of equipment (like a router or DNS server) in between a user and a web server can replace any occurence of `https://` with `http://` in links and redirects. The result is that an adversary can return pages to a user that look like the correct website, but actually are not.

Session hijacking is another cyberattack technique, wherein an adversary monitors network traffic for cookies, and uses the cookie in the adversary's own HTTP headers, tricking a web server into thinking that the adversary is someone else.

CS50 Trust Models

Overview

Downloading a piece of software from the Internet requires a substantial amount of trust on part of the user. The user must trust that the piece of software that is being downloaded doesn't contain malicious code. In theory, any software downloaded onto a computer could delete all of the files on that computer. Yet, we still trust that the software we download is safe and secure. This is the basis of **trust models**.

Key Terms

- trust model
- backdoor

Backdoors

To the right is an excerpt of a hypothetical login program written in C, which checks a username and password to determine whether a user's account credentials are valid. In reality, login programs would probably compare the user's inputs against username and password values stored in a database. Furthermore, these would most likely be encrypted in some way and not just stored as plain text. Still, we'll use this simplified version for the sake of example.

```
if ((strcmp(username, "rob") == 0 &&
       strcmp(password, "thisiscs50") == 0) ||
    (strcmp(username, "tommy") == 0 &&
       strcmp(password, "i<3javascript") == 0))
{
    printf("Success!! You now have access.\n");
}
else if (strcmp(username, "hacker") == 0 &&
       strcmp(password, "LOLihackedyou") == 0)
{
    printf("Hacked!! You now have access.\n");
}
else
{
    printf("Invalid login.\n");
}
```

Notice that after performing the initial check for username and password combinations, the code offers an additional way to gain access to the system (by using the username "hacker" and the password "LOLihackedyou"). This method of accessing a system through an alternate means, one that differs from the way that users are supposed to access a system, is known as a **backdoor**.

In this case, any users who were to read the code of the login program would be able to identify the fact that there's a backdoor into the system. However, users who download software usually won't have the opportunity to see the code of a program before it's compiled.

Exploits in a Compiler

Even if a user sees a program's code before they download it and determines that there doesn't seem to be any malicious code or backdoors in the code, that doesn't necessarily mean that the program itself is secure. Compilers, the program that translates source code into object code, can also be the source of exploit.

There are a couple ways that compilers can be used to exploit users. A compiler could, for instance, be programmed to take a perfectly benign login program and inject code into it that creates a backdoor. Anyone who looked at the source of the login program code itself wouldn't detect any signs of a backdoor. But if the source code were compiled with the malicious compiler, then the resulting program would have the backdoor in it. Of course, in this case, anyone who were to look at the source code of the compiler would see that there was code in the compiler that injects malicious code into the login program.

Let's take this one step further. Imagine that we wrote a compiler that injected malicious code into the compiler itself (remember that compilers themselves need to be compiled). Then a hacker could theoretically take benign source code for a compiler and turn it into a malicious compiler. In this case, even if the compiler source files and the login program source files didn't contain any malicious code or backdoors, compiling the source files could still result in the injection of malicious code.