EMBEDDED IN THIN SLICES

The Internet of Things (Part 8)

Security Vulnerabilities from the OWASP

Last time, Bob detailed some security vulnerabilities in web-enabled devices. This month he looks at five more vulnerabilities from the Open Web Application Security Project (OWASP).

By Bob Japenga

As the leaves begin to turn here in New England, it is time to turn the page (at least for a while) on the Internet of Things. We have only touched on this important topic, but for now, we will make this our last installment discussing the Internet of Things (IoT).

As I mentioned in the last article, the Open Web Application Security Project (OWASP) is a great resource for you to stay up to date on security issues. **Table 1** provides the current list of the top 10 means of web hacking. Last month, we looked at the first five. This month, we will look at the next five individually and make sure we know what they are, look at a real-world example where they have caused trouble, and how to prevent them.

SENSITIVE DATA EXPOSURE

My first experience with the sixth most common cause of security violations, sensitive data exposure, came in the early 1970s. We were designing computer controls for glass container manufacturing. Our company built both the machines and the controls. Initially our designs used proprietary hardware and proprietary software. By the end of the 1970s, my boss wanted us to use off-the-shelf hardware and proprietary software. We chose a system built around a DEC PDP-11/04 minicomputer and their Industrial Control Subsystem (see **Photo 1**). One day, I walked into the plant of one of our customers and found a number of control systems using our proprietary software on their off-the-shelf hardware. Since this was a really good customer who bought a lot of machines from us, our marketing department decided not to protest this theft. The controls costs were about 10% of the cost of the machine. The problem was that we exposed our sensitive data (our software) with inadequate protection and with a wink from our marketing department, our customer began using it on their hardware at no cost. Basically, in this case, our code was not protected from reuse.

Several years later, MicroTools designed a postage scale system for a customer. The system weighed the item to be shipped, the user selected the carrier (UPS, FedEx, USPS, etc.), and the system provided the shipping costs. Every year or so, each carrier would update the shipping rates. Each unit in the field was updated with new rates several times a year. This was a lucrative business that created a recurring revenue stream from their existing products. These units were

PHOTO 1
A DEC PDP-11/04 minicomputer (Source: www.corestore.org/1104.htm)

sold through dealers who found out how to update the rate data without having to pay our customer. Again there was a symbiotic relationship between these good dealers (who could sell our competitors' products instead of ours) and our customer. Our customer depended upon the dealer for sales. Our customer chose not to go after these "good" dealers. We were asked to create a new design that included a way to protect the data so that this could not happen. Basically, we included a unique electronic serial number in the hardware and created a unique option code for each device to unlock the new rate. The dealers did not like it, but we didn't tick them off enough to stop the thievery.

Both of these instances can happen today with your IoT designs if we don't secure our data. Most IoT products follow the 80/20 rule of thumb for software and hardware development costs. Software development costs are much higher than your hardware development costs. Foreign competition can copy your hardware cheaply at will, but can you protect your software from being copied?

PHYSICAL ACCESS

The bottom line is that you don't want anyone to be able to read your software and sensitive data either over the air (OTA) or in the hand. For example, someone could buy your product, reverse engineer it, obtain sensitive data out of the chips, and use that to launch a significant attack on all of the products you have deployed. We saw that with the attack on the Jeep Cherokee that we discussed earlier in this article series. Once inside, they will have access to your algorithms and data if it is not protected. So what can we do to prevent physical access?

Your first level of protection is to prevent someone from being able to read the software inside the chip (e.g., FPGA, microcontroller, flash, or SD card). Let's look at some options that are available.

Microcontrollers and FPGAs: Most microcontrollers and FPGAs include methods of locking down the software contained internally from being read externally. But if your data is really sensitive, pirates have techniques to read the ones and zeros optically! It is chilling what they can do. Lattice Semiconductor employs a technique that supposedly prevents that, but I am not familiar with this process. Of course, since we are never perfect, we want to have a device that we can program. So you cannot lock it down internally. With the advent of IoT, OTA firmware updates are huge. So that defeats the built-in lock down. What we need are more sophisticated lock downs that include

Rank	Title
1	Code Injection
2	Broken Authentication and Session Management
3	Cross Site Scripting (XSS)
4	Insecure Direct Object Reference
5	Security Misconfiguration
6	Sensitive Data exposure
7	Missing Function Level Access Control
8	Cross-site Request Forgery
9	Using Components with Known Vulnerabilities
10	Unvalidated Redirects and Forwards

TABLE 1
The top 10 means of web hacking

a cryptographic key or some other means of authentication. If you are aware of any such devices, please drop me a line.

SD Cards: Many designs now use SD cards as the base flash memory. Both the software and any sensitive data can be easily copied and reverse engineered. All data that you want to protect should be encrypted. Operating systems like Android and Linux build this into their infrastructure. I would strongly recommend not using SD cards in designs that contain sensitive data unless the data is encrypted.

Flash Memory Chips: I am not familiar with any means to prevent flash memory chips from being read once they are extracted from your PCB. If you are aware of some techniques, please drop me a line.

Remember to design for the worst. As you design, assume that your worst enemy can read all of the data. Keep in mind that every piece of data that goes over the Internet can be read. Keep in mind that every piece of data inside your device can be read. No critical keys should be stored. No passwords should be stored. What this approach will do is to identify the data that you really care about. And then you can come up with a scheme to protect this—your most sensitive data.

OVER-THE-AIR ACCESS

Most of the techniques for preventing OTA access to your sensitive software and data have been covered in previous articles in this series. Simple things like not running your applications as root or with root privileges can be a major step. I have two Android devices at home that will not play certain video content that I get from a pay-for-view streaming provider. The problem (as I am told by their tech support) is that the applications are running with root privileges and the licensed material won't run in that environment

The bottom line: minimize physical access, assume that everyone can see your data, and be hyper vigilant about any means of OTA access.

MISSING FUNCTION-LEVEL ACCESS CONTROL

And that leads nicely into the seventh most common cause of security violations on the web: devices that don't require any level of access control. We have covered most of these in previous articles when we discussed: open ports; anonymous logins to FTP, SFTP, or browser access; and console ports. Protecting console ports is something we have not been doing but will begin doing that going forward. The bottom line: do not allow any remote or local access without a strong username and password.

CROSS-SITE REQUEST FORGERY

Most of us have experienced this as a user. This is when you are asked to provide your user name and password to some financial institution. This is sometimes called "phishing." I am not sure how this could be applied to IoT devices. Perhaps you have experienced this. Drop me a line and let me know. Bottom line: don't worry about this one.

USING COMPONENTS WITH KNOWN VULNERABILITIES

The ninth most prevalent security vulnerability for web services is using components of libraries that have vulnerabilities. The OWASP/WASC limits them to "known" vulnerabilities. But there is nothing to stop an unknown vulnerability from wreaking havoc with your IoT. This can be the most difficult of all of the security vulnerabilities to mitigate. For example, one of the most common libraries used in our IoT devices is OpenSSL. They maintain a list of known vulnerabilities that provides some interesting bedtime reading. Seriously, take a look this list. It demonstrates how difficult it is to create secure software. But it also demonstrates that we are going to incorporate components that have known vulnerabilities. It is a way of life. As we discussed in an earlier article in this series, your best approach is to keep abreast of all of the libraries or components that you are incorporating into your IoT device.

The bottom line: only incorporate perfect libraries or components into your IoT device. But seriously, to cover this, we must do two things. Keep abreast of the known vulnerabilities in the libraries we use. And always provide a means for updating all software-driven devices.

UNVALIDATED REDIRECTS & FORWARDS

This is normally a problem for web designers who open their site to spammers and phishers with their insecure designs.

circuitcellar.com/ccmaterials

REFERNCES

Lattice Semiconductor, "Security Aspects of Lattice Semiconductor iCE40 Mobile FPGA Devices," v1.0.0, 2011.

Open Web Application Security Project, www.owasp.org.

OpenSSL Software Foundation, "Vulnerabilities," www.openssl.org/news/vulnerabilities.html.

Some unsuspecting client is phished and then fooled into thinking that the content they are seeing is from the legitimate website. For example, this happened to the CNN website and it allowed spammers to display their malicious content while the URL the client saw had "cnn.com" as an address.

But this can be a problem for IoT devices in several ways. For example, one way might be if your local embedded web server has a feature that allows your server to be redirected or forward a request to another of your IoT devices based on the parameter in the URL. This might be useful if the device was down and it wanted the server to access someone else's data. This would allow your IoT device to become a relay for all sorts of malicious content. And the unsuspecting bloke would think it was coming from you.

The core problem here is a design that allows your web server to redirect or forward to another site based on a URL parameter or some other mechanism without validating the redirection or forward. For example, if your web server in your IoT device had this PHP code in it:

```
$redirect_url = $_GET['url'];
 header("Location: " . $redirect_url);
```

The bottom line: don't forward or redirect requests to sites that are not validated.

SECURITY RECAP

In these past 18 months, we have covered a lot of information about the Internet of Things. We started in June 2015 by looking at some options you have for connecting to the Internet. Then in August of that year, we looked at how to select a wireless carrier for your IoT device. In October of 2015, we looked at some options for using a very small microprocessor to connect wirelessly to the Internet. In December of 2015, we looked at the costs of certifying your IoT device. Then, in February of 2016, we delivered the first of five articles dealing with IoT security. Sometimes when you take many things in thin slices, you can have yourself quite a feast.

ABOUT THE AUTHOR

Bob Japenga has been designing embedded systems since 1973. In 1988, along with his best friend, he started MicroTools, which specializes in creating a variety of real-time embedded systems. With a combined embedded systems experience base of more than 200 years, they love to tackle impossible problems together. Bob has been awarded 11 patents in many areas of embedded systems and motion control. You can reach him at rjapenga@microtoolsinc.com.