Week 4 exercise: Data flow threat modelling

This week we are going to practice data flow based threat modelling. As we saw during Session 4, once you have an idea of the technical realisation of a system, an effective way of determining potential security issues is to draw a Data Flow Diagram and go through the steps of data flow based architectural security analysis.

Note: This week’s results are going to be used for the week 5 exercise. We will provide a reference answer for week 4 later, in the case you will elect not to do this week’s exercise; however, it will greatly help the week 5 exercise if you have done the week 4 one. Also, we can only provide the reference answer once the deadline for week 4 exercise has passed. Therefore, if you decide to skip this exercise, your week 5 exercise timebox will be shortened into one week.

Note: We are not expecting that you are familiar with all the technologies used in this system. For example, you are not expected to know the details of Node.js security. However, when you are drawing the picture, indicate which aspects you assumed or guessed. Those are the aspects you would (in reality) discuss with the development team.

Note: The explanation this week is rather lengthy. If you want to simulate a case where you are in dialogue with the development team, the week 4 exercise “lab hour” can be (at your option) used to ask clarifying questions of the technical solution. I’ll just whip out appropriate answers from my hat :-)

If you have time, I would well advise you to read through the primary material on the Week 4 Reading List before you start. In addition to that, if you have access to this book, Ross Anderson’s Security Engineering, 2nd Edition has Chapter 12 on “Monitoring and Metering” that could well prove to be useful in setting your mindset for this particular exercise. If you can’t get the book from the library, don’t worry though - the chapter is also online: https://www.cl.cam.ac.uk/~rja14/Papers/SEv2-c12.pdf

System description

In this exercise, you are consulting a small start-up company that is planning to implement an electricity meter that communicates with the utility company (electric company) over a GSM text message return channel. You have met with the development team, and they have an initial high level description of the system, and would like your feedback on what security risks might still be present.

The system description will follow in prose. This is typical of software development; they might not have a pretty picture drawn for you. Your task is to be able to “decipher” this explanation. Of course, because this exercise is on paper and not a face-to-face meeting with the developers, it
is not easy for you to ask clarifying questions. If you believe some information is missing, you can provide an educated guess, and you will not be punished for that.

The electric meter

Each customer household will be provided with an electricity meter. This meter will measure the energy that has passed through it, measured in kilowatt-hours. Physically, it will be a sealed and tamper-evident metal box that has an internal nickel-metal hydride battery for backup power, and it runs a Debian GNU/Linux system on hardware comparable to a Raspberry Pi (http://www.raspberrypi.org/) that can handle the simple operations of storing the data. It has no Internet connection. The operating system and data are stored on filesystem on an SD card. Each box also has a GSM modem (a commercial off-the-shelf 3G module, only configured for 2G) that is capable of sending text messages, and a SIM card that is provisioned by the electric company. There is a small stub antenna on top of the box.

The box also has a small LCD display that shows the current meter reading (and can show short error messages) and a LED that is blinks every kilowatt-hour. The meter also has the meter number on a sticker that is used by the customer for creating an account in the web system (described later). The meter number consists of 25 alphanumeric characters, and is random.

The actual measurement is done by a watt-hour meter with embedded software, also within the box, that connects to the Linux board over an I²C bus. Power is obtained from the electric network (unmetered side) through a small switching power supply, also within the box. The box is physically secured to the mains distribution board.

The watt-hour meters (i.e., the subsystem that actually measures the energy through sampling the voltage and current) and cellular modems are sourced as off-the-shelf components from third parties, and their internal security is not a subject of the analysis.

For service, the box can be opened using a key, but it breaks the seal. The hardware board has a USB socket that can be used to attach a serial console and to mount extra filesystems over USB. The service personnel can configure the device and update the software by giving appropriate commands on the system console. The SIM card, the USB socket and the SD card interface are all sealed with a tamper-evident seals within the box.

Communication with the electric company

Every hour, the current meter reading and the timestamp are stored on the SD card by the metering application that runs on the Debian GNU/Linux. Another process reads the accrued newest measurements and sends it to the electric company over text message (SMS) two times per day. The message contains the meter number, the total cumulative energy spend (i.e., the meter reading) for each hour from the previous message until now, and a timestamp. This all is packed into a 8-bit text message to save space. As a response, the electric company server will send back a text message that contains the latest meter reading and the current real time. The
box updates its internal clock based on this server-provided time, and now knows what was the latest reading that the company successfully received.

If the electric meter’s message sending process does not get a response that acknowledges the meter reading, it will just send a new message when the next message would be due, sending all the readings that haven’t yet been sent. Concatenated text messages are used if necessary, and if the length overflows the maximum concatenation amount, several messages are sent.

To ensure that the messages are not tampered with and are genuine, the messages in both directions are encrypted using AES-CBC using the first 128 bits of the electric meter number (which, as we recall, is random). The same key is used in both directions, and the electric company has a mapping from phone numbers of the electric meters to meter numbers (i.e., keys).

The text messages will be delivered to the electric company through the operator’s SMSC (Short Messaging Service Center) and then over to the electric company using the operator’s Content Gateway platform. This service is described in http://www.sonera.fi/media/13caa3ee5521dfba08d4687796d2e8c032ff2f1b.pdf.

(In reality, I’ve never touched that service – I just found that document on the Internet and it seems to fit this exercise well. I make no claims about the security level of that service.)

The document is fairly long, so the applicable sections are referred to below. The electric company says that the actual settings of the Provider Server are secured by an operator consultant, so the Provider Server configuration is outside the scope of this analysis.

In short, a text message from the electric meter travels through:
- Operator air interface (i.e., from the Mobile Terminal to the Base Station)
- Operator’s cellular network infrastructure
- Operator SMSC
- Operator Server (see figure 1 and section 4.1.3 of the linked document above)
- The Internet
- Provider Server (see section 2.2.2), running on the same host as the application server
- HTTP request to the electric company application server (see section 4.2.3.2)
  - The HTTP request is described in detail in section 1.5.2.2.1 of another document, http://www.sonera.fi/media/13caa3ee5521dfba08d4687796d2e8c032ff2f1b.pdf

A text message from the application server to the electric meter travels through:
- Application server calls the ‘send’ command line tool (see section 5.4)
- Provider Server
- The Internet
- Operator Server
The electric company database & billing
The electric company application is a web application implemented in server-side JavaScript using Node.js and using nginx as the web server. The web application receives the text message from the Provider Server over HTTP (see above), and stores the data into a PostgreSQL database running on Debian GNU/Linux Squeeze. The application runs on the same host as the Provider server and the PostgreSQL database server.

The PostgreSQL database contains the following tables that are relevant to this system:

- Mapping of customer electric meters’ SIM cards’ phone numbers to the meter numbers for message encryption/decryption
- All the received meter readings from every customer (each line has meter number, meter reading and timestamp)
- Customer usernames and hashed passwords for the web user interface (see below)

The PostgreSQL database has one user (and associated password) that is used by all the back-end systems.

For billing purposes, the electric company has a server application that connects to the PostgreSQL database over TCP over the electric company’s internal network. It retrieves the meter readings for each customer (i.e., the customer number, the timestamps and meter readings), calculates the amount of electricity spent since the last invoice, and handles the billing. You are told that this billing application itself is legacy code and is not in the scope of this analysis.

For fraud prevention purposes, the billing system also detects large changes in the use of electricity. If there is a significant drop in the metered usage (that might mean that the meter has been bypassed), a service engineer is dispatched to the customer to check the system and obtain the locally stored meter readings from the SD card of the device. This fraud detection system is also legacy code - it was used back in the day with old meters, too - and not in the scope of the analysis.

Customer web service
The customers can log in on the electric company web system using a username and password, and view the current electricity usage status for their meter. The plan is to have much more functionality for this web system, but for now, there is only the option of viewing the current meter reading.

The web system is implemented using JavaScript and the Node.js platform (http://nodejs.org/) with Nginx (http://nginx.org/en/) as the web server, running on Debian GNU/Linux. The web
The web system consists of pages to:

- Create an account, where the user provides an email address and the meter number (printed on a sticker on the meter, which, as we recall, contains 25 alphanumeric characters). The system sends a link to the email address, which, after followed, lets the user to provide a password.
- Retrieve a reset link for a forgotten password, that prompts for the meter number and email address. It uses the same link system as is used with account creation and sends the user an email with a password reset link.
- Log in with the email address and a password.
- Display the meter reading.

The web system uses HTTPS (TLS) towards the user's web browser.

Your task:

- Draw a Data Flow Diagram, using the suggestions and guidance from the lecture notes, that shows the system on a level that is appropriate for architectural threat analysis. I recommend you draw the data flow diagram on paper with a pen instead of wasting time with a computer-based drawing program, although you are free to use a diagramming tool if you really want to do so. The diagram should be returned as a scanned image¹, or a (sufficiently clear) digital picture. Electronic submissions must be PDF, JPEG or PNG files. To reiterate, it should show:
  - The data flows that are related to information collection at the electric meter, and sending that information to the electric company database;
  - The data flows associated with retrieving billing data and fraud prevention (but you do not have to go into the details of billing/fraud prevention systems as such, and you can treat those as black boxes);
  - The data flows associated with the web service;
  - The users, service personnel, and administrators;
  - Data stores that store the metering data and credentials required for any of the above users;
  - Information processing components on a process level;
  - Annotation of what data is being transferred at which data flow, and over which protocol stack.

- Note that the explanation may not have all the information you need. This is typical, and in reality, you would be in dialogue with the developers to figure out what they actually have. For the purposes of this exercise, you can pretend you asked them, and just implement something that is plausible and not in conflict with the rest of the architecture.

¹ Photocopiers at Exactum can send scanned images via email to any email address. There is a better scanner on the 1st floor corridor (i.e., the floor labeled “2” on the elevators). If you return a digital photo, make sure it has good contrast, no glare, and no compression artifacts that make it hard to read.
• List the interfaces which you think should be security or robustness tested. List them in the order of importance: Those interfaces that you feel must be tested first, and those interfaces that are not that critical afterwards. For each interface, describe in a few words what type of testing you would expect to see done to it, and why you determined it is a high/low priority target.

• Perform an open-ended threat modelling exercise for the data flows you have drawn in the diagram. Use STRIDE - you might want to try the cards and/or the questions from lecture notes.

• **Note** that the description above may not have enough information to say whether something really is a risk or not. You can assume it might be, and you need more information to determine whether it really is. (This is how the analysis typically goes in real life.) List as many risks you can find, but there is **no need to list more than 20 risks - if you find more, just list the top ("worst") 20.** For every security risk that you find, list:
  ○ What type of security risk is it (using the STRIDE or CIA classification);
  ○ A very short description of what the (technical and/or business) **impact** is, and what the attacker could perhaps do;
  ○ What extra information you would require to determine whether this really is a risk or not, if it is not clear-cut. Remember that risk has two parts: impact and likelihood, and you would have to have some sort of gut feeling about both aspects.

**To return:**
• The data flow diagram.
• The list of interfaces that you think would require security testing.
• The list of security risks you found, with information as described above, maximum of 20 risks.