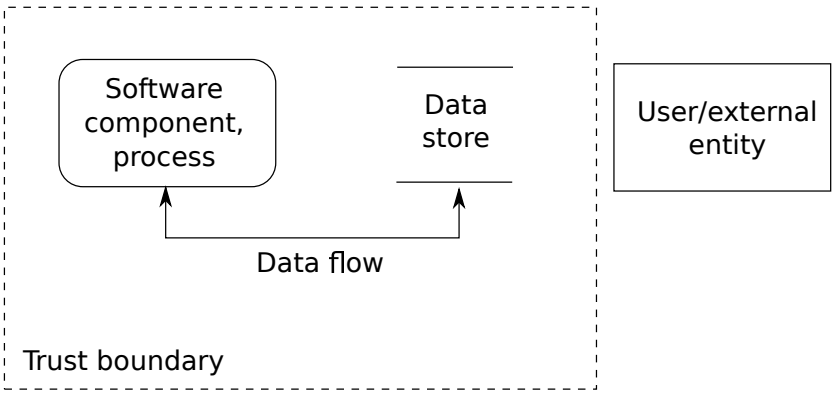
Extra Notes on Drawing Data-Flow Diagrams

v1.0, February 14th 2023

Data-flow diagrams (or DFDs for short) are a common step in threat modeling, since they are an aid to thinking about an overview of the structure of a system that is relevant to how the system might be attacked. Our approach to data-flow diagrams in this course is based on Adam Shostack's "Threat Modeling" book, but we don't have time to go into as much detail, and the book would be a lot to ask you to read, so in lecture I've tried to distill what I think are the most important points. This set of notes are intended to supplement the lecture discussion with more details you can refer back to when drawing data-flow diagrams on your own for assignments.

Data-flow diagrams are a tool for technical communication, but compared to writing code or even some other kinds of diagrams you might draw for databases or software engineering, they don't have mathematical precision in their meaning. Drawing a good data-flow diagram is more like writing a good prose description of a system than it is like implementing the system in a programming language. Not coincidentally, you'll often be drawing them together with writing assignments. We have conventions about how to draw data-flow diagrams to make it easier to share them smoothly. But beyond those conventions, you should also be guided by this intent in communication, just like when you are writing prose: which way of expressing the structure of the system is going to give the reader the most correct understanding?

1 Things you draw in data-flow diagrams:



- External entities are represented by rectangles with sharp corners, and they are often found near the edges of a diagram. They represent things that you don't get to design as part of your system, but that your system interacts with. For instance people using a system will usually show up as external entities.
- Software components are represented by rectangles with rounded corners. Shostack calls them "processes", and while they are not the same as operating system processes, that terminology does emphasize that they are best thought of as units of functionality. The boxes may or may not line up with particular units of code at the implementation level, like functions, methods, classes, packages, or executable programs. It's best to label them based on functionality.
- Data stores are represented with lines above and below a text label, with no lines on the sides. They represent parts of your system that store information but that we don't think of as initiating actions: for instance a database, or a directory in a filesystem. For this reason, we usually don't draw data flow edges directly from one data store to another: it should be a software component driving the access.

- Potential data flows are represented with edges connecting between the above three kinds of nodes. The edges should have arrows representing the potential directions of data flow, but you will find that a large majority of your data flow edges usually have arrows in both directions, because communication is commonly bi-directional. We usually don't write labels on data flow edges, since labels could clutter a diagram and be a little too restrictive in thinking about the role that a data flow plays. For instance, an attack will often use a data flow in a way that is technically possible, but is different from what the system designer intended. The best choice to keep larger diagrams readable is to draw data flow edges with horizontal and vertical lines and 90 degree corners (like wires in a circuit diagram).

- Trust boundaries are represented by a dashed outline (commonly rectangular) that groups together nodes. The trust boundary represents a situation where the nodes inside the boundary are more trusting of each other than they are of nodes outside the boundary. These boundaries can exist for either technical or organizational reasons. When thinking about an implementation, you can infer the existence of a trust boundary based on where security checks are or are not performed.

2 Other points to watch out for in drawing data-flow diagrams:

- Be sure to distinguish control flow from data flow. Control flow is one piece of software triggering another part to be executed, while data flow is about information that is the output of one piece of software being the input to another. They sometimes happen together, but sometimes not, and as the name suggests, a data-flow diagram should be about data flow.

- Check that all your nodes are connected by data flows. Software components and data stores will usually have both incoming and outgoing edges, unless you want to show that they are read-only or write-only. It should only be possible to divide your system into disconnected subgraphs if they are completely independent from each other.

- Both the presence of a data flow between two nodes and the absence of data flow represent design decisions and implementation requirements. If two nodes aren't connected, what about the system will ensure that? If an access control mechanism is complex in and of itself, it might deserve to be represented by its own node.

- A trust boundary that surrounds a single node doesn't provide much useful information, since we already assume that each node trusts itself. (If you are imagining a node with mutually-distrusting parts, it probably needs to be multiple nodes.) If you are tempted to draw such a trust boundary, make sure that its label provides additional information beyond the node's label. Once case where this might make sense is if it could make sense to move more things inside the trust boundary as a design choice or a future change. For instance, showing that one software component is hosted on Amazon AWS says something about its trust, and even if your system currently has only one such component you could imagine moving others inside that boundary.