THE INFLUENCE OF INTERNET-ENABLED DEVICES ON MANUFACTURING: COLLABORATION AND CYBERSECURITY

BY

KATARINA M. STRUCKMANN

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Advisor:

Professor William P. King
ABSTRACT

The manufacturing industry must adapt to the advances in technology that the wealth of Internet-enabled devices permeating society in recent years has both heralded and effected.

As the Internet allows people to work together from remote locations, manufacturing has become increasingly globalized and distributed. A product’s design may take place on a different continent than its assembly, and its components and subassemblies can be sourced from an eclectic set of suppliers. However, the manufacturing industry has yet to adopt a solution for communication across these distances. The Digital Manufacturing Commons (DMC), which aims to enhance collaboration throughout supply chains, hosts tools including a marketplace of analytical models. The first project explores a hybrid cloud implementation of the DMC that allows users to design custom user interfaces for models that they publish to the DMC. The analytical models can live in the DMC, but the hybrid implementation allows them to be linked to custom interfaces by hosting them on a separate Front End machine. This project explores custom interface elements including specialized input fields, animations that provide users with real-time feedback, and plots that visualize iterations on data.

The second project takes advantage of the prevalence of smartphones and the manufacturing industry’s slow adoption of security measures, exploring a side-channel attack on manufacturing systems. This attack captures data using the sensors in a smartphone and reconstructs the object being fabricated as well as some parameters of the process. The project investigates the efficacy of two separate methods of reconstruction as well as the effects of certain variables, such as the model of smartphone, on the data quality. A potential defense against this attack is suggested and tested.
I have so many people to thank. First, my advisor, Dr. King, for giving me the opportunity and resources to work on these projects.

I am grateful to Jim Barkley, Alex Maties, and the rest of the Digital Manufacturing Commons team, without whom there would be no DMC. They implemented much of the DMC’s hybrid cloud architecture and provided useful feedback at every stage.

My thanks also go to Avesta Hojjati, Anku Adhikari, and my other collaborators & coauthors on the side-channel attack, particularly for designing, implementing, and testing the reconstruction methods. I also must thank Patty Weisensee, who called me for the phone call attack experiment.

Finally, I am grateful to Nancy Struckman of Editing for Style, LLC, for proofreading this thesis, and to her and the rest of my family for providing me with the resources and support that got me to where I am.
# TABLE OF CONTENTS

CHAPTER 1  INTRODUCTION  ........................................... 1  
  1.1  Motivation ................................................. 1  
  1.2  Research Scope ............................................ 3  
  1.3  Outline .................................................. 4  

CHAPTER 2  USER INTERFACES FOR THE DIGITAL MANUFACTURING COMMONS  ............................................ 5  
  2.1  Background ................................................ 5  
  2.2  Motivation ................................................ 9  
  2.3  A Hybrid System Architecture  ......................... 10  
  2.4  DMC Applications with Custom User Interfaces ........ 12  
  2.5  Summary ................................................. 22  
  2.6  Figures ............................................... 23  

CHAPTER 3  A SIDE-CHANNEL ATTACK ON MANUFACTURING EQUIPMENT  ............................................ 36  
  3.1  Introduction .............................................. 36  
  3.2  Motivation & Background .................................. 39  
  3.3  Attacks .................................................. 43  
  3.4  Experiments .............................................. 52  
  3.5  Defense ................................................ 57  
  3.6  Summary ................................................. 59  
  3.7  Figures and Tables ....................................... 60  

CHAPTER 4  CONCLUSIONS AND RECOMMENDATIONS  ............. 76  

REFERENCES ..................................................... 78  

APPENDIX A  CREATING AND RUNNING DOME MODELS  .......... 85  
  A.1  Using DOME .............................................. 85  
  A.2  Figures ................................................ 87  

APPENDIX B  DOME MODELS AND INTERFACE SCRIPTS ............ 94  
  B.1  Model Rocket Design ..................................... 94  
  B.2  The Beer Distribution Game .............................. 115
CHAPTER 1

INTRODUCTION

1.1 Motivation

Manufacturing is a dated industry. The word itself evokes a gritty feeling: the cacophonous roar and clack of the machines in a belt-driven textile mill, the pervasive fine dust of raw material processing, the acrid fumes of curing adhesives, the tangible wall of heat emanating from a furnace. Manufacturing hardly has the bright, futuristic associations of the technological products it sometimes creates, which is an accurate, if rough, depiction of the industry’s state.

While manufactured products have been increasing in complexity, manufacturing itself has been lagging behind. In certain industries, the development of new products—high-performance electronics and pharmaceuticals, for example—is tightly coupled with the development of new manufacturing technologies, but most products or their components, such as office chairs, food packaging, automotive parts, and watches, are manufactured by techniques that have seen little recent innovation [1]. This creates two types of problems: first, manufacturers must adopt new technologies to handle the effects of manufacturing becoming increasingly distributed and globalized; and second, manufacturers must also adopt security measures to combat the increasingly complex attacks that new technologies support. Unfortunately, the adoption of these two solutions is a slow and painful process for many manufacturers, especially smaller businesses.

The growth of the Internet and Internet-enabled devices in recent years has changed both the types of products that are manufactured and manufacturing processes. While these changes are sometimes linked—innovative products may require novel manufacturing processes—they have also experienced independent growth.
Advances in computing have enabled the manipulation of large data sets, known as big data. This, in conjunction with the resource sharing that the Internet supports, has led to the increasingly popular collection and use of manufacturing data. This capture and use of data throughout a product’s lifecycle is referred to as digital manufacturing. There are two categories of digital manufacturing proposed by Xu et al.: intelligent manufacturing and cloud manufacturing [2]. Intelligent manufacturing includes real-time data collection from manufacturing equipment. While installing monitoring software can cost a few thousand dollars per machine, the efficiency gained from interpreting and acting on the data rapidly offsets the cost. Some versions of monitoring software benefit from the data standards provided by protocols such as MTConnect, increasing their efficiency and interoperability [3].

Cloud computing refers to connecting designers to production services through a cloud-based collaboration platform. It allows designers and fabricators to connect and collaborate in a way that is beneficial for both parties, as designers are able to request small production runs that would otherwise be economically prohibitive and manufacturers are able to add small jobs to their schedules to reduce their machines’ downtime [4, 5].

Manufacturers who have adopted digital manufacturing have been successfully capturing and utilizing data from their production floors, but collecting data and providing connectivity to the sensors and machines introduces new security risks. Internet-enabled devices, especially smartphones, have recently become ubiquitous. This is beneficial in that more designers, manufacturers, and other members of the supply chain have the means to communicate easily. However, these devices also present a security risk.

While digital manufacturing and modern computing can increase the efficiency of design, production, distribution, and other aspects of the manufacturing pipeline, they also leave manufacturing susceptible to new avenues of attack that are just now being explored.
1.2 Research Scope

This research explores two topics related to digital manufacturing: collaboration and cybersecurity.

The first project is related to the Digital Manufacturing Commons, a communication and collaboration platform for the manufacturing industry that allows users to publish and run analytical models describing design, manufacturing, supply chains, and other elements of a manufacturing lifecycle. The project focuses on the development of custom user interfaces for these models and the hybrid cloud architecture that allows interfaces to communicate with models remotely.

The second project investigates the security of manufacturing floors by exploring a side-channel attack on manufacturing equipment. The attack utilizes the sensors in a smartphone to capture data about a manufacturing process and, with this data, reconstruct the design of the object being manufactured as well as some of the manufacturing process parameters. This project also considers a defense against the attack.
1.3 Outline

This first chapter provides motivation common to both projects and outlines the remainder of the thesis.

The second chapter discusses the first project, user interface design and remote model execution architecture for a manufacturing collaboration platform. The chapter introduces tools used to develop the models and interfaces before discussing the system architecture that allows the models and interfaces to communicate. Finally, the project is illustrated with two examples.

The third chapter discusses the side-channel attack on manufacturing equipment, introducing motivation for the attack and describing why it might be executed. The chapter then discusses how the attack is carried out, presenting two different methods. The results of experiments testing variants of the attack are presented. The chapter concludes with an illustration of a defense against the attack.

The fourth chapter presents conclusions drawn about these projects and about manufacturing as well as recommendations for future research.
CHAPTER 2

USER INTERFACES FOR THE DIGITAL MANUFACTURING COMMONS

This chapter introduces custom user interfaces designed to enhance the utility of the Digital Manufacturing Commons, a collaboration platform that hosts manufacturing applications. The remote execution of these interfaces is supported by a hybrid cloud architecture.

2.1 Background

**Context for the Digital Manufacturing Commons.** The Digital Manufacturing Commons (DMC) is an open-source software tool in development by the Digital Manufacturing and Design Innovation Institute (DMDII). The DMDII, a program founded in February 2014 by the Chicago-based innovation accelerator UI LABS, is the second institute in the National Network for Manufacturing Innovation Program (NNMI) [6]. The NNMI was founded on a 2012 recommendation from the Advanced Manufacturing Partnership (AMP) Steering Committee that was adopted by the President’s Council of Advisors on Science and Technology (PCAST) [7]. AMP noted that, while the United States has a strong tradition of basic, early-stage research and development, there is a technical and financial gap between research and its application in a successful product. Agencies such as the Department of Defense (DoD) and the National Aeronautics and Space Administration (NASA) refer to Technology Readiness Levels (TRL) to describe the maturity of a technology. The nine-level scale ranges from early observation (TRL 1) to early prototyping (TRL 3), prototype demonstration in the operating environment (TRL 7), and successful mission operation (TRL 9) [8]. Research and development institutes typically work at TRL 1-3, and manufacturers fall around TRL 8-9. The NNMI’s institutes occupy the gap the United States has between approximately TRL 4 and TRL 7. The DMC is
one of many projects supported by the DMDII to bridge this gap between research institutions and industry partners.

**The Digital Manufacturing Commons.** The DMC is a web-based collaboration platform designed to coordinate the digital thread throughout the product lifecycle. This web interface is shown in Figure 2.1. The DMC has two primary capabilities that support this: project management tools and a marketplace of services. The project management tools allow users to collaborate on projects, share files, assign tasks, and more. Users can also ask for submissions to their projects; they may do this to find partners, suppliers, or providers of other types of services with whom they can collaborate.

The DMC supports powerful analytical modeling. Models, also called “apps” or “services”, are published to the service marketplace for other users to run. These models accept input from the user, perform analysis of the input, and return results, analyzing anything from airfoil design to supply chain dynamics to data streams from factory floors. Models are defined in a tool called the Distributed Object-based Modeling Environment (DOME).

**Architecture of the Digital Manufacturing Commons.** The DMC is an open-source project. The DMDII hosts one instance of it that users can join. Users can also deploy their own copy of the DMC, should they choose [9].

At present, the DMC can be deployed as a stack of virtual machines in either Amazon Web Services or Microsoft Azure, commercial cloud computing platforms. The basic stack, shown in Figure 2.2, consists of six machines: the Front End, REST Services, Solr, DOME, ActiveMQ, and Database machines. The Front End machine serves content to the user’s browser; the REST Services orchestrate the communication between the other machines; Solr provides search functionality; DOME hosts and runs models; ActiveMQ displays status messages from DOME; and the Database stores information such as the results of model execution.

Three different types of cloud computing implementations exist: public clouds, private clouds, and hybrid clouds. In public clouds, services and resources are provided by a third party and accessed over the Internet; while users do not have to maintain these resources, public clouds are inherently less secure than other options. Private clouds are accessed using a private network; while users are responsible for maintaining their resources, private
clouds are more secure. Hybrid clouds combine the two, hosting some services and resources locally and sourcing others from third party providers.

The DMC can be implemented as any of the three types of clouds, as shown in Figure 2.3. In a public implementation, all of the machines in the stack and all of the data, models, and interfaces are hosted by and stored in the DMDII’s instance of the DMC. Users may also host a private implementation of the DMC; companies with sensitive data may prefer to host a private instance within the company network for security reasons. A hybrid implementation would make use of the model execution services provided in the DMDII’s instance, but the models, data, and interfaces could be generated, hosted, and stored in any combination of machines either in the DMDII stack or in a user’s personal computing environment, whether local or cloud-based. For example, a user could publish a model in the Service Marketplace in the DMC, allowing the DMC to store the model, run the model, and store data, but host a custom interface for the model on her own website.

**DOME.** DOME is the tool inside the DMC that supports the creation and execution of models. DOME was developed in the CADLab at MIT in the late 1990s and early 2000s in three iterations that highlight its major capabilities. The first version explored computation, modeling, and optimization; the second, a distributed marketplace; and the third combines these features into a marketplace that provides simulation services [10, 11, 12]. DOME supports the analytical modeling capabilities of the DMC as well as the distributed, hybridizable nature of the marketplace, as illustrated in Section 2.3.

DOME standardizes communication among analytical modeling programs, allowing users to design complex chains of models that make use of the strengths of different programs. DOME has plugins for a number of different programs, including Abaqus, Adams, Excel, MATLAB, SolidWorks, and Vensim; it also supports Name-Value models that allow users to run an arbitrary script with an arbitrary compiler, such as a Python or Java script and its interpreter. A user can link together models in different programs: for example, she could design a MATLAB script that optimizes a parameter such as a dimension, use Abaqus to run a finite element analysis (FEA) on the part with this dimension, and then confirm in SolidWorks that the part mates correctly with its assembly without causing interference. DOME would link these models together so that the user only had to provide one set
of inputs at the beginning, instead of running three separate models. This example chains the models together serially; DOME can also network models in parallel or in complex combinations thereof.

DOME is composed of two services, a server and a client. The server, which runs in the background, hosts models as well as the simulation engine that executes models. The client provides a graphical user interface (GUI) where users can build models, deploy models to a server, run models on a server, and administrate the server. An example of the process of creating, deploying, and running a model is illustrated in Appendix A.
2.2 Motivation

The austere user interface the DMC provides for interacting with models belies the power and complexity of these models. This interface, shown in Figure 2.4, consists of a list of text fields that represent the inputs and outputs. A user enters inputs (the units are preset in the definition of the DOME model), presses a button to run the model, and receives the output in another series of text fields. Note that the interface shown in Figure 2.4 is a mockup; this interface is currently unavailable while it is under development by the DMDII, but the figure illustrates approximately what it looked like when it was last available. This interface is an accurate reproduction of the simple text field interface that the DOME client provides for running models (Figure 2.4, bottom), but it adds no functionality. For more information about DOME, see Appendix A for an illustration of the process of creating and running a model.

The deceptively simple list of text fields not only masks the DMC’s capability, it also forces the data into a specific visualization that may not suit it. There are two primary ways in which the DMC’s user interface is lacking: it is time-consuming to fill in text fields by hand, and the text view is a poor representation of many kinds of data. The first can be addressed by the file upload feature currently in development. The number of inputs can quickly grow to hundreds for even a simple model; instead of entering them individually, the user can upload them in a format such as a CSV and design the model to extract the values and assign them to variables.

Data visualization, however, is a more complex problem to address. Data visualizations can extract and present insights that are difficult to see, especially in large data sets, but the correct type of analysis is specific to each data set. There is no standard visualization that would work well for every possible data set, so any interface provided by the DMC must be capable of adapting to specific models and data. In addition, users may wish to have multiple interfaces for a certain model: different types of users (e.g., project managers, designers, engineers, suppliers, and students) may want to see a specific subset of the inputs and outputs in a model or a specific data visualization that is tailored to their needs. The text field interface lacks this flexibility.
2.3 A Hybrid System Architecture

A hybrid cloud implementation of the DMC presents a solution to the problem of interface improvement: it allows users to design and host custom interfaces for models. Each interface and corresponding data visualization can be precisely tailored to the model it represents. The user links the interface, hosted on her own website, to a model hosted in either a public or private instance of the DMC.

The communication that supports hybrid clouds by allowing components of the DMC to be hosted on different machines is implemented with a REST API. The REST services live on a dedicated virtual machine in the DMC stack, as shown in Figure 2.5. Remote model execution requires five of the machines in the DMC stack: the Front End, REST Services, DOME, ActiveMQ, and Database machines. While the REST Services, ActiveMQ, and Database machines must all belong to the same instance of the DMC, the system can use any Front End and DOME machine, as illustrated in Figure 2.6. The user can use a browser on any machine to access the DMC.

A request to run a model is initiated by the user in the browser, which sends the input data as well as parameters including the model ID to the Front End (Figure 2.5). The REST Services machine passes these parameters to DOME. The Simulation Engine in DOME will select the appropriate model and run it with the user’s input data, returning the results. DOME sends the results to the ActiveMQ message queue. The REST Services machine reads the message and stores the results in the Database. The browser will send a request through the Front End and REST Services machines to retrieve the results from the Database if they are there. If the model has a long run time, the browser will not return a result, and it will try again at a later time.

There are a number of different ways this architecture could be hybridized. At present, the REST Services, ActiveMQ, and Database machines must all belong to the same instance of the DMC. The other pieces can be freely linked from other instances (Figure 2.6). A user who designs a custom interface for her project may find it most efficient to link her own Front End machine to the rest of the DMC stack. A user who wishes to publish models to the DMC can use a DOME Client on one machine to publish models to the DOME Server in the DMC, or she could replace the DOME machine altogether with her own instance. Users could also choose to replace both the Front End and
DOME machines. The hybrid cloud architecture, combined with the REST API, provides users with a powerful flexibility.
2.4 DMC Applications with Custom User Interfaces

This section illustrates a hybrid cloud implementation of the DMC with two DMC applications, each consisting of a customized user interface and an associated analytical model run by DOME. These applications, or apps, are linked to a custom Front End machine and a custom DOME machine that communicate with machines in the DMC stack.

**Tools.** In both apps, all of the analytical models are native DOME Models. These models do not make use of any external applications such as MATLAB; instead, they allow the user to run a Jython script. Jython, an implementation of Python that can run on the Java platform and import Java modules, is used because DOME is written in Java.

The interfaces are web interfaces. They have an HTML skeleton and are styled with CSS. The simpler input fields, such as sliders, use jQuery, a JavaScript library. The more complex parts of the interface, such as animations and updating charts, are designed and implemented with d3.js (also known as D3, or Data-Driven Documents), a JavaScript library designed to power complex data visualizations. Unless otherwise specified, the visualizations are SVGs (vector images in the Scalable Vector Graphics format) that are published and updated with D3. Using vector graphics not only allows the interfaces to be updated rapidly and smoothly (D3 can handle animations with smooth transitions), it also allows the interfaces to be scaled; they render cleanly at any zoom magnification.

2.4.1 Model Rocket Design

The Model Rocket Design app simulates the design, construction, and launch of a model rocket, calculating the height attained from launching a rocket of the user’s design. This app was designed as a tutorial model to teach new users of the DMC how to run apps published in the DMC marketplace. As this model is intended for all audiences, not only technical users who may build their own apps, the model has a simplified qualitative interface that is friendly to users of all backgrounds. Technically-inclined users can choose to view the model’s engineering details if they wish.

The app has three interface pages and three DOME models. The primary
interface page collects inputs; the others have details about the model but are not interactive. The primary page asks the user to choose which components to use in the rocket, and, as the user selects components, an animation showing the rocket design updates in real time. The calculations are carried out by three models: an integration model that links two other DOME models together, a model that calculates the rocket’s drag coefficient, and a model that uses the drag coefficient to calculate the maximum height the model rocket could reach.

The Jython scripts for the DOME Models and the JavaScript scripts for the interfaces discussed in this section are provided in Appendix Section B.1.

Models. A model rocket, though simplified, is a complex system of interdependent variables. There are a number of tradeoffs involved in design: for example, a larger motor, while it provides more thrust, also weighs more, and the larger diameter body needed to accommodate a larger motor will experience more drag. This system is captured in a set of analytical models that execute the calculations.

The REST Services machine that orchestrates the communication between the other machines is still under development by the DMC team, but we have designed the way that custom interfaces will be integrated as a hybrid cloud implementation. When a user submits inputs in the custom model rocket interface, they will be sent from the browser to a custom Front End machine, through the DMC’s REST Services machine, and to a custom DOME machine. In this case, the Front End and DOME machines can be the same machine; one can host all of the necessary components. When the inputs reach DOME, it calls on an integration model—a model that links other models together—that coordinates two analytical models. The interaction between these models is shown in Figure 2.7.

The first of the two models calculates the drag coefficient of the rocket as a function of parameters such as the body diameter and the fin shape. These parameters are constants built into the interface; when the user submits qualitative parameters to be run, the interface identifies the quantitative constant associated with the parameter and sends it to the Front End machine. After the first model calculates the drag coefficient, it is used as an input to the second model, alongside the remaining parameters, such as the motor burn
time, that the interface collected and provided. The second model calculates the maximum height the rocket would reach when launched.

The integration model has an interface that exposes only the overall input and output parameters; intermediate variables do not need to be shown. As a simple example, for the equations $a = b \times c$ and $c = d + e$, the interface will request the inputs $b$, $d$, and $e$ and show the output $a$, leaving the intermediate variable $c$ hidden. For the rocket design model, the interface shows every parameter except for the drag coefficient, the sole intermediate. The inputs to the integration model are the constants required to run the calculation, some of which are parameters determined by the user’s choices. The output that is displayed to the user after the model is run is the height the rocket would achieve if launched.

The two models were published to the DOME Server before the integration model could be created, as the integration model subscribes to the two other models as resources. When the integration model is run, it is linked to the two other models and passes intermediate values between them. Here, the DOME Server will call the integration model, which runs the first model, retrieves its output, and then runs the second model.

**User Interface.** The Model Rocket Design app interface is split into multiple pages, showing all users a basic interface and relegating the details to separate pages to reduce clutter. The primary page, shown in Figure 2.8, lists seven parameters the user can choose from in the design of her rocket, including the type of motor and the fin shape. To the right, an animation updates in real time as the user selects new parameters. Figure 2.8 illustrates the animation’s state with different sets of parameters chosen. When the user has finished selecting parameters and components, she presses the button at the bottom of the page. This button collects the values of the inputs and sends them to the Front End. The second page of the interface describes the calculations that the model runs, as shown in Figure 2.9. The third page, shown in Figure 2.10 lists the commercial off-the-shelf (COTS) components on which the model is based. The interface has been simplified to a series of qualitative choices, but each choice is linked to the values for COTS components that a user could buy and assemble if she wished. For example, the “wider” and “narrower” choices for the body tube are linked to the parameters, such as diameter and weight, of the 24 mm and 18 mm
body tubes sold by Apogee Components [13, 14]. The full list of datasheets from which the model draws its constants is given on this page of the interface [15, 16, 17, 18, 19, 20, 21, 22, 23].

While the features this interface provides immediately benefit the user, some additional features would improve the interface further. The interface could benefit from a feature that allows the user to toggle between qualitative and quantitative descriptions of the components. Similarly, a chart tracking the history of the users’ submissions would help them see how their choices interact and optimize their design. While this interface does not feature history tracking, it has been included in the Beer Distribution Game interface discussed in Section 2.4.2.

**Calculations.** This section reproduces the calculations used in the two analytical models. The models are adapted primarily from equations published by Randy Culp [24], with details supplied by additional sources [25, 26, 27].

The drag coefficient is equal to the sum of the drag coefficients for each element of the rocket:

\[
C_d = C_{dN} + C_{dBT} + C_{dBS} + C_{dF} + C_{dInt} + C_{dLL}
\]

where the terms on the right side of the equation are the drag coefficients for the nose cone, the body tube, the base, the fins, the fin interference, and the launch lug, respectively. Determine necessary constants from the rocket’s geometry:

- \(d\), the diameter of the body tube
- \(d_b\), the diameter of the base, typically equal to \(d\)
- \(L\), the length from the tip of the nose cone to the end of the body tube
- \(S_W\), the wetted surface area of the nose cone and body
- \(S_{BT}\), the cross-sectional area of the body tube
- \(C_R\), the fins’ root chord (the length along the line where the fin attaches to the body)
- $S_F$, the fins’ surface area
- $t/C_R$, the fin thickness ratio, where $t$ is the fin thickness
- $S_{LL}$, the cross sectional area of the launch lug (a very thin ring)
- $S_{LLw}$, the wetted surface area of the launch lug
- Last, assume a 3:1 ogive nose cone for this model. For subsonic model rockets, a rounded ogive shape has the smallest drag coefficient; a pointed nose cone is not necessary because there is no shock wave to deflect.

Calculate $C_{dN} + C_{dBT}$ together:

$$C_{dN} + C_{dBT} = 1.02C_f \left(1 + \frac{1.5}{\left(L_{dL}\right)^{1.8}}\right) \frac{S_W}{S_{BT}}$$

where $C_f$ is the skin friction coefficient, which is a function of the rocket’s length, velocity, and type of boundary layer. The skin friction coefficient requires a complex calculation of its own. This model approximates it by selecting a velocity of 100 ft/sec with a turbulent boundary layer. For these conditions, $C_f$ ranges between 0.004 and 0.005 for various lengths. This model approximates the empirical values of $C_{dN} + C_{dBT}$.

Calculate $C_{dBS}$ from the previous result:

$$C_{dBS} = \frac{0.029}{\sqrt{C_{dN} + C_{dBT}}} \left(\frac{d_b}{d}\right)^3$$

Next, calculate an intermediate value for the fin drag that will be corrected for area in the next step. For fins that are unfinished or have slightly rounded edges, use the following equation:

$$C_{dF}^* = \frac{D_F}{0.5dV^2S_F}$$

where $V$ is the velocity, previously approximated as 100 ft/sec, and $D_F$ is the drag force.
For fins that have been sanded into a streamlined shape, with a rounded leading edge and a sharp trailing edge, use the following equation, where the velocity included in the unfinished fin calculation is incorporated in $C_f$:

$$C_{d_F}^* = 2C_f \left(1 + 2\frac{t}{c}\right)$$

Calculate the fin drag and interference drag from $C_{d_F}^*$:

$$C_{d_F} + C_{d_{int}} = \frac{C_{d_F}^*}{S_{BT}} 0.5An$$

where $A$ is the area of a wing planform, including the section covered by the body tube, and $n$ is the number of fins. This is equivalent to calculating the surface area of a single wing including the section between the wing and the body’s center axis and multiplying by the number of fins. This term is divided by $S_{BT}$ to normalize it by the same surface area as the other drag coefficients so they can be summed.

Find the drag due to the launch lug. This is the sum of the pressure drag on the face of the launch lug $S_{LL}$ and the skin friction drag due to the wetted surface $S_{LLW}$. The term is normalized by $S_{BT}$:

$$C_{d_{LL}} = \frac{1.2S_{LL} + 0.0045S_{LLW}}{S_{BT}}$$

Finally, sum the drag coefficients to find $C_d$:

$$C_d = C_{d_N} + C_{d_{BT}} + C_{d_{BS}} + C_{d_F} + C_{d_{int}} + C_{d_{LL}}$$

The following constants are required to calculate the rocket’s final height:

- $\rho$, the density of air
- $g$, the acceleration due to gravity
- $m$, the rocket’s mass, supplied by the components’ data sheets
- $A$, the rocket’s frontal area, determined from the body diameter
- $C_d$, the rocket’s drag coefficient
- $T$, the rocket motor thrust, supplied by the rocket motor datasheet
- $I$, the rocket motor impulse, supplied by the rocket motor datasheet

From these constants, calculate $k$, the wind resistance factor; $t$, the rocket’s burn time; $x$, an intermediate variable; and $v$, the rocket’s velocity at motor burnout:

$$k = \frac{1}{2} \rho C_d A$$

$$t = \frac{I}{T}$$

$$x = \frac{2k}{m} \sqrt{\frac{T - mg}{k}}$$

$$v = \sqrt{\frac{T - mg}{k} \left(1 - e^{-xt}\right)}$$

Finally, calculate $y_b$, the height reached during burn, and $y_c$, the height reached while coasting, which sum to $y_{total}$, the rocket’s maximum altitude:

$$y_b = \frac{-m}{2k} ln\left(\frac{T - mg - kv^2}{T - mg}\right)$$

$$y_c = \frac{m}{2k} ln\left(\frac{mg + kv^2}{mg}\right)$$

$$y_{total} = y_b + y_c$$
2.4.2 The Beer Distribution Game

The Beer Distribution Game app allows a user to play the Beer Distribution Game, a teaching tool that simulates the management of a supply chain. Like the Model Rocket Design app, this app was designed as a tutorial for users new to the DMC. It demonstrates two primary capabilities of the DMC: the storage and retrieval of data from previous runs of a model and an updating visualization of the model’s output.

The Jython scripts for the DOME Models and the JavaScript scripts for the interfaces discussed in this section are provided in Appendix Section B.2.

The Beer Distribution Game. The Beer Distribution Game was developed by the System Dynamics Group at MIT in the 1960s as part of an effort to provide students management experience and relevant lessons in a short time span [28]. The game simulates a supply chain for manufacturing, distributing, and selling cases of beer. It is played for a set number of rounds that represent weeks—usually around 40—with the goal of minimizing expenses and losses incurred during that time. There are typically four or eight players, with one or two assigned to each of the four stages in the supply chain: the Retailer, the Wholesaler, the Distribution Warehouse, and the Factory. In each round, each stage takes two actions: placing an order to the stage that precedes it in the supply chain and shipping inventory to the stage that follows it. This flow is illustrated in Figure 2.11.

This process sounds straightforward until the players are presented with a complication: they are not allowed to communicate with each other during the game. The players choosing how many orders to place may only base their decisions on information they have at hand, such as the history of their inventory, backlog, and shipments. Without any way of estimating how demand varies, the game invariably ends up demonstrating the bullwhip effect.

The bullwhip effect, or the whiplash effect, is the behavior of minor variations in a supply chain to accumulate, becoming magnified as they are passed through the supply chain [29]. Small fluctuations in order values grow at an astonishing rate; if the Retailer sees a demand that varies between 5 and 10 orders per week, the Factory may see orders placed with values close to 100 after just a few weeks. This trend is demonstrated in Figure 2.12. The
decisions that drive order placement are influenced by both historical data and the tacit expertise of the players. The players, hindered by their lack of information and the time delay that each round creates, often place orders for amounts larger than they need, creating a buffer that lends them a sense of security. Studies of traditional supply chains suggest that lack of communication contributes to many of the pitfalls supply chains experience, as is demonstrated by The Beer Distribution Game [30].

**Model and Calculations.** The analytical model that runs this application is a DOME model that controls a single round of the game. The model reads the state of the game from the last round and simulates a round by moving inventory, updating order amounts, and calculating costs. Ideally, for each round, the user’s desired number of orders would be passed to the model along with the values from the last round, read out of the database; as this functionality is still under development for the DMC, the current implementation stores the values from the previous round in the browser using JavaScript instead. The model would run with these input values and return the outputs to be stored in the database for the next round. The interface would collect the outputs and update the charts, and the user would be able to begin a new round by entering a new order value. The REST Services and Front End machines are still under development by the DMDII; while, at the time of writing, the required communication channels are a work in progress, the foundation has been laid for this to function in the near future.

The process of playing the game is described with simple calculations: addition and subtraction for placing orders and moving inventory and multiplication to track costs. The model accepts incoming goods by incrementing the appropriate variables, pays for these goods by tracking their cost, ships outgoing goods by decrementing inventories, and finally tracks additional costs such as backorder fees. The full set of calculations is provided in Appendix Section B.2. The model is based on the original Beer Distribution Game as well as a cost metric suggested by Ammon et al. [31, 32].

While the calculations are simple, the system they describe is highly complex. Decisions affect numerous variables in ways that are difficult to predict with the limited knowledge a player has access to, and it is difficult to accurately model the intuition and expertise that influences players’ decisions.
This model is a strong example of how DOME can represent powerful complex systems easily.

**User Interface.** Like the interface for the Model Rocket Design app, this one has been split into multiple pages. The first page describes how the game is played, providing an introduction for new users. The second page is the interactive interface shown in Figure 2.13. After the user chooses a role, the interface updates automatically to provide an input field for the orders placed by that role. The interface includes a diagram to remind users how orders and product flow between the stages. The interface also displays three charts that show the player’s Inventory, Orders, and Costs, updating at the end of every round to reflect progress through the game. While these charts by default show only the series for the user’s role in the game, to reflect that the players cannot communicate, the user can choose to display the series for all four of the roles.

This user interface presents a new feature that would prove invaluable to many types of models: charts that reflect the history of a model’s use. Updating charts could be used with iterative models like this one, where each run is closely related to the previous one. They would also be beneficial to models where the user is manually optimizing certain parameters, such as the Model Rocket Design app. For this type of model, updating charts or tables could plot the selected variables and the corresponding outputs, allowing a user to run experiments that indicate the influence of certain variables on the outcome. The nature of the data visualization will vary, depending on the type of model the visualization represents and what aspects of the data are interesting to the user. The ability to customize the interface of a DMC model, tailoring it to the user’s needs, will be invaluable.
2.5 Summary

This chapter introduces a hybrid cloud implementation of the Digital Manufacturing Commons. The hybrid architecture allows users to link custom Front End and DOME machines to the DMDII’s instance of the DMC. This enables users to design and host custom user interfaces and new analytical models.

Two applications, each consisting of an interface and a model or a group of models, are illustrated. The first showcases custom input fields and an animation that provides real-time feedback on design parameters as well as a collection of linked analytical models on the back end. The second application explores iteration, with a model that draws past results from the database as inputs and an interface that updates to reflect the history of each iteration.
2.6 Figures

Figure 2.1: Two pages of the Digital Manufacturing Commons web platform. The Dashboard (top) shows a user Services she has published, Tasks assigned to her, Projects she is a part of, and Community members she is following. Services, or analytical models, are published to the Marketplace (bottom) for other users to discover and run.
Figure 2.2: The six virtual machines that constitute the basic stack of the DMC. The REST Services machine manages communication between the other machines. The machines are typically hosted as a stack on either Amazon Web Services or Microsoft Azure.
Figure 2.3: Possible public, private, and hybrid implementations of the DMC. If all of the machines, and their corresponding data, interfaces, and models, are hosted in the DMDII’s instance of the DMC, it is considered a public cloud. They can also all be hosted on a private network. If the resources are split up, with some provided in a public instance and some residing in a private instance, it is referred to as a hybrid cloud.
Figure 2.4: A mockup of the web interface in for running a model in the DMC (top) and the interface in the DOME client (bottom). Shown is an interface for the model \( V = I \times R \). The user enters values in the input text fields (\( I \) and \( R \)); the units were set in the model definition. Pressing “Run” or “submit” will run the model and update the output field (\( V \)). The DMC’s interface is closely based on the interface in DOME.
Figure 2.5: The communication, initiated when a user runs a model from a web interface, between the interface and the five machines in the DMC stack that are required to execute models. The REST Services machine manages the communication that allows the components of the DMC to be hosted on separate machines.
Figure 2.6: The system architecture of the DMDII’s DMC instance as a hybrid cloud. The user can link private instances of the Front End and DOME machines (dark gray) to the machines in the DMDII’s stack (light gray). A private Front End machine allows a user to render a model’s custom user interface in a browser. A private DOME machine allows a user to host and run a model locally instead of in the DMDII’s instance but still publish it to the Marketplace and store data in the DMDII’s database. The hybrid architecture allows users fine-grained control over security.
Figure 2.7: The interaction between the three models in the Model Rocket Design app. The integration model accepts input from and returns results to the user. It subscribes to two other models as resources. First, the integration model runs a model that calculates the rocket’s drag coefficient; second, it runs a model that uses this drag coefficient to calculate the rocket’s maximum altitude.
Figure 2.8: The primary interface for the Model Rocket Design app. The animated sketch at the right updates as users select different parameters, providing feedback during the design process.
Rocket Analysis

This page describes the calculations that the model performs with your choice of parameters in the following sections:

I. Rocket Altitude Calculations
II. Rocket Drag Coefficient
III. Example

Rocket Altitude Calculations

The following constants are required to calculate the rocket’s final height:

- $\rho$, the density of air
- $g$, the acceleration due to gravity
- $m$, the rocket’s mass, supplied by the components’ data sheets
- $A$, the rocket’s frontal area, determined from the body diameter
- $C_d$, the rocket’s drag coefficient (see below)
- $T$, the rocket motor thrust, supplied by the rocket motor datasheet
- $I$, the rocket motor impulse, supplied by the rocket motor datasheet

From these constants, calculate $k$, the wind resistance factor; $t$, the rocket’s burn time; $x$, an intermediate variable; and $v$, the rocket’s velocity at motor burnout:

$$k = \frac{1}{2} \rho C_d A$$
$$t = \frac{I}{T}$$
$$x = \frac{2k}{m} \sqrt{\frac{T - mg}{k}}$$
$$v = \sqrt{\frac{T - mg}{k} \left(1 - e^{-xt} \right)}$$

Finally, calculate $y_b$, the height reached during burn, and $y_c$, the height reached while coasting, which sum to $y_{total}$, the rocket’s maximum altitude:

$$y_b = \frac{-m}{2k} \ln\left(\frac{T - mg - ku^2}{T - mg}\right)$$
$$y_c = \frac{m}{2} \ln\left(\frac{mg + ku^2}{mg}\right)$$

Figure 2.9: The beginning of the second page of the Model Rocket Design interface. This page illustrates the calculations that the model performs after a user submits her inputs.
Rocket Datasheets

The following data sheets and specifications are the sources of the constants used in the model, such as rocket motor impulse and body tube mass.

Rocket Motors

Motor codes are given as [letter][number]-[number], where these values specify the motor’s [total impulse]-[average thrust]-[time delay]. See the National Association of Rocketry’s explanation for more information.

This model references datasheets supplied by the National Association of Rocketry after testing and certification.

- Estes A8 [PDF]
- Estes B4 [PDF]
- Estes B6 [PDF]
- Estes C6 [PDF]

Rocket Body Components

Apogee 18 mm and 24 mm body tubes
Apogee 1/8” launch lug
Apogee nose cones for 18 mm and 24 mm body tubes
The average density of balsa wood (second source) used to calculate fin weight

Figure 2.10: The third page of the Model Rocket Design interface, which lists datasheets used as reference. The model stores constants, such as the components’ dimensions and masses, to use in the calculations.
Figure 2.11: The supply chain modeled in The Beer Distribution Game. Orders flow from the Retailer to the Wholesaler, the Regional Warehouse, and then the Factory. The Factory processes raw materials into new product. Inventory flows in the opposite direction.
Figure 2.12: The bullwhip effect, often seen in the results of The Beer Distribution Game. Small fluctuations in order volume are amplified as they progress through the supply chain.
Figure 2.13: The primary interface for the Beer Distribution Game application. Users can choose to play as any of the four stages in the supply chain. The graphs of Inventory, Orders, and Costs update after every round of the game. By default, the user can see only the series for the role she is playing (top), but she can choose to show all four (bottom).
This chapter introduces a side-channel attack that can be performed on manufacturing equipment using only a mobile phone. The attack captures data from the phone’s sensors and reconstructs the form of the object being made as well as some of the details of the manufacturing process. The chapter explores two different reconstruction methods and a number of variations on the process of gathering data, concluding with an experiment that tests a potential defense against this attack.\footnote{This chapter is based on a paper that is in review at the time of thesis deposit [33].}

3.1 Introduction

Hackers have taken notice of the increasing amounts of valuable information available in the cyber-physical systems on manufacturing factory floors. In addition to straightforward data theft, adversaries can potentially take advantage of simple yet effective side-channel attacks based on electromagnetic leaks, acoustic emissions, timing information, light emission, and power consumption [34, 35, 36, 37, 38, 39]. The leaked information can be used to compromise systems and to obtain or infer sensitive data. For example, researchers have partially compromised Diffie-Hellman exponents, factored RSA keys, and broken other cryptosystems by measuring the amount of time required to perform private key operations [40, 41, 42]. Defending against side-channel attacks requires a level of security more advanced and more comprehensive than updating an operating system or installing security patches. Despite the efficacy of firewalls and anti-virus software, there is currently no effective way for manufacturers to protect against information leakage from their factory floor equipment.

Side channels also exist in modern phones. Smartphones are programmable
and come with a growing number of cheap yet powerful embedded sensors, including a microphone, accelerometer, magnetometer, gyroscope, GPS, and camera. While most intended uses of smartphone sensors, such as fitness tracking, are benign, they open up avenues for attack. A smartphone’s accelerometer, for example, can be used to infer a password typed on its screen [43].

This novel attack uses the sensors in a mobile phone to capture sensitive information from manufacturing equipment on a factory floor, as shown in Figure 3.1. In a typical factory, nearly everyone on the factory floor has a phone or other electronic device that can be deliberately or inadvertently used to execute a version of this attack. As such, the effectiveness of the attack is independent of the level of information technology or security sophistication on the factory floor and inside the manufacturing equipment.

The attack captures the relevant sensor data by deliberately or accidentally placing an attack-enabled phone close to, on top of, or inside a piece of manufacturing equipment while the machinery is fabricating a target object. An example of this setup is illustrated in Figure 3.2. Alternatively, the relevant audio can be recorded by deliberately or accidentally making or receiving a phone call while standing next to the machinery, or it can be captured on any other device nearby that has a microphone and appropriate malware. The captured data can reconstruct a model of the object being manufactured, along with its manufacturing process parameters, using either of two methods. The attack is viable for both additive and subtractive manufacturing processes, demonstrated with a 3D printer and a CNC mill.

**Contributions.** This work contributes:

- *New techniques.* The data captured by acoustic and magnetic sensors embedded in the phone can be used to identify specific manufacturing equipment and manufacturing processes, including reconstructing manufactured objects and reproducing the processes used to make them.

- *New understanding.* This side-channel attack is applied to both additive and subtractive manufacturing equipment, specifically 3D printers and CNC mills. The fundamentally different operating modes of these two types of manufacturing equipment indicate that the attack may be broadly applicable across many types of manufacturing equipment.
• Implementation and evaluation of reconstruction methods. Two methods are capable of accurately reconstructing manufactured objects and the processes used to make them, based on machine learning and on signal processing plus crowdsourcing, respectively.
3.2 Motivation & Background

Cyberattacks on the manufacturing sector typically fall into one of three categories: theft of intellectual property or processes, disruption of manufacturing operations, or sabotage of products or reputation [44]. These cyberattacks are already widespread: in 2014, 21% of manufacturers reported a loss of intellectual property (IP) [45]. These observed losses may be the tip of the iceberg, as 69% of all 2012 data breaches were carried out within a few hours, but 64% of breaches took months or years to detect [46]. In addition, the number of manufacturing cyberattacks is growing quickly: the Industrial Control Systems Cyber Emergency Response Team (ICS-CERT), operated by the US Department of Homeland Security, responded to 50% more incidents in the manufacturing sector in 2015 than in 2014 [47, 48].

IP theft can target product design information, manufacturing process information, or both. An attacker could theoretically reproduce a product given its design. However, many manufacturers’ competitive advantage lies in their ability to manufacture a given design better, faster, or cheaper than their competitors. Process information may include the details of what materials are used, what machines are used and in what order, and the settings of those machines, e.g., which tool was used, the tool rotation rate, and the material feed rate.

One contributing factor to the prevalence of IP theft is the rudimentary IT security measures in place in many factories. Expensive manufacturing equipment can last for decades, and, like those of a probe sent to outer space, its embedded computer systems will forever reflect the state of the art at the time the equipment was fabricated. The largest barrier to the adoption of additional cybersecurity defenses is that many manufacturers, especially smaller businesses, are unwilling to compromise their productivity [49]: every minute of downtime is costly, such as the average $1.3M per hour downtime loss reported by the auto industry [50]. Any software or hardware change can produce an expensive domino effect that increases downtime. For example, upgrading the operating system may require new hardware, which requires new software and drivers; any equipment that isn’t compatible with the upgrade must be replaced; the new systems must be tested extensively, as some applications may behave differently; and the users must be trained on the new systems [51]. Though methods to improve IT security while respecting
the need for continuous operation are an interesting direction for future research, this attack is not affected by the quality of the security measures in a factory’s IT systems.

The manufacturing sector has a rich history of research on obtaining information about a manufacturing process from its acoustic emissions. Recordings have been used to judge parameters including tool wear, tool breakage, chatter, chip formation mechanism, material removal regime, sheet metal material hardness, sheet metal thickness, and the identity of the metal or alloy being machined [52, 53, 54, 55, 56, 57, 58]. These reconstruction methods use acoustic information for less benign purposes.

The many sensors in Android phones and iPhones include an audio sensor (microphone), image sensor (camera), touch sensor (screen), acceleration sensor (tri-axial accelerometer), light sensor, proximity sensor, and several sensors (including the Global Positioning System [GPS]) for establishing location [59]. Currently, most phone sensors are used to collect data from the surroundings of the user to offer external observations to a specific application. For example, **Metal Sniffer: Metal Detector** is a portable metal detector app available on the Google Play store that takes advantage of the magnetometer sensor in an Android device to identify electromagnetic fields that ultimately assist the user in locating nearby metal and magnets [60]. Another example is the **Accelerometer Monitor** app, which records and saves accelerometer readings [61]. Such data has already been used for malicious purposes [62, 63, 64, 65]. For example, Cai et al. [66] highlight the ability of modern mobile devices to snoop on users by sniffing their smartphone’s sensors, such as the microphone, camera, and GPS sensor.

Al Faruque et al. used thermal side channels to infer the activity of a 3D printer [67] and investigated the possibility of attacking manufacturing machinery via audio recordings [68]. The study placed a microphone close to additive manufacturing equipment to record fabrication runs, then used machine learning to reconstruct the G-code used by that specific 3D printer to manufacture the object, with an accuracy of 89.72% in reproducing the as-designed object’s perimeter. While Al Faruque et al. used a high-quality microphone located in a specific location in a controlled environment, this work uses ordinary mobile phones that may be located anywhere near the machine or in the user’s hand and could readily be applied to machinery
located in public fabrication labs. A final difference is that because G-code is quite low level, reproducing the same object on a different model or type of machine requires nontrivial extra work to rewrite the G-code. For that reason, this attack uses a high-level reconstruction that can be translated into G-code for a variety of machines.

3.2.1 Attacker Motivations & Access

When Eve’s phone illicitly records data about the factory floor, she could be intentionally carrying out corporate espionage. Alternatively, she could be an unwitting dupe with a compromised application or the innocent maker or receiver of a phone call at an ill-advised moment. In these latter cases, Eve may have been targeted by a third party such as a rival manufacturer or swept up in a large net cast by a well-financed backer of economic espionage such as a nation-state.

A third-party backer’s decision to finance the attack or purchase the information thus obtained may be driven by the cost or impossibility of developing an equivalent design or process for local use or by a desire for greater situational awareness. For example, a nation-state hacker might be seeking to increase the competitiveness of its manufacturing sector or to gain the ability to manufacture objects viewed as important for national interest. Such motivations may have been behind the theft of the design for Lockheed Martin’s US F-35 Lightning II fighter jet, stolen by hackers allegedly supported by the Chinese government [69]. Likewise, the US and Israeli governments have been attributed as potential sources of the Flame malware, apparently designed to increase situational awareness of Iran’s technical capabilities and activities [70]. In addition to gathering files likely to contain technical information, Flame collected data from the sensors of the devices it infected.

While Flame targeted Windows PCs, similar malware can be constructed for phone applications. Once a malicious app has been installed, recording could be activated by a geofence around the factory in the background of another app with appropriate permissions, such as a game.

These considerations affect the practicality of phone-based attacks. If national interests are at stake, state sponsored hackers may have access to the resources to carry out the kinds of technical work and social engineering nec-
necessary to get a compromised application onto a factory employee’s phone. The compromised application needs only to record the data from key sensors and offload it to the sponsoring organization at a convenient future time. Similarly, state sponsorship can make it easy to arrange a call to an unsuspecting employee on the factory floor and record the audio, e.g., while the employee is on hold for a supposed pizza delivery or sweepstakes prize. These attacks are effective even if the factory floor’s own cyber-physical systems are invulnerable.
3.3 Attacks

Once a phone has recorded a manufacturing process, by either capturing the audio from a phone call or recording readings from multiple sensors, the attack reconstructs from this data the object being manufactured and the process used to make it. We explore two reconstruction methods. The crowdsourcing reconstruction method uses basic signal processing techniques and can be guided by a non-expert user if reconstruction is difficult. Its current implementation uses audio data only, making it compatible with both the phone call and malicious application attacks. The machine learning reconstruction method uses machine learning techniques to reconstruct a fabricated object from magnetometer and microphone sensor data.

Different fabrication machines have different process parameters: a mill’s parameters include tool shape and size, spindle speed, feed rate, coolant flow, stepover value, and cut depth, while a 3D printer’s parameters include material diameter, extruder feed rate, material extrusion rate, extrusion temperature, platform temperature, and layer height. The primary contribution of this work is on tool location: deciphering the path that the tool head—a cutting tool or an extruder—traces out, which, when combined with process parameters, allows us to reconstruct designs. A secondary parameter we explore is feed rate. The location parameter is critical to both additive and subtractive manufacturing, especially in the fabrication of parts with tight tolerances. While previous works have extracted a wealth of information from the acoustic emissions of a manufacturing process, tool head location is a novel parameter to explore.

Different machines have different constraints in traversing 3D space; reconstruction can take advantage of these constraints to simplify the task. For example, a 3D printer builds an object as a series of horizontal layers. For each layer, the print head traces out a path in the XY plane before pausing, incrementing its vertical position, and beginning a new layer. Additionally, the shape of each layer is constrained by previous layers, since the layers must overlap either part of the object or support material. Likewise, subtractive processes often work to remove material in layers, and they typically remove material adjacent to material that has already been removed. We take advantage of this layer-focused machine behavior by restricting our attention to the XY plane for a fixed value of Z, i.e., a given layer. Our training data
and validation experiments use almost-planar objects (a two-layer 3D print or a planar contour cut).

Tool head movements are described as vectors: the head’s angle of movement with respect to the X axis and the distance moved along that straight line. Any planar figure that can be manufactured by machines like a CNC mill and a 3D printer can be specified as a list of these vectors, with splines, or curves, decomposed into short tangential segments. We reconstruct both the angles and the distances.

3.3.1 Crowdsourcing Reconstruction Method

**Pre-processing.** The crowdsourced reconstruction method relies on a reference library constructed in advance, such as the one shown in Figure 3.3, using one example audio clip for each angle that the machine tool head might traverse. These audio clips were obtained from recordings of the machine moving in a fan shape, shown in Figure 3.4, at increments of one degree from the X axis. Before saving an audio clip in the library or applying the reconstruction technique, we use several standard audio signal analysis techniques to improve the quality of its signal. First, the audio clip is divided into frames using a Hanning window method with an overlap of 25% between successive audio frames; this is the default sliding window technique used in audio analysis to give cleaner frequency estimates for signals and to reduce spectral leakage. Second, we convert the audio clip to a spectrogram that shows the magnitude of each frequency over time, using a short time Fourier transform. Third, we define the background noise of the machine as the data in an audio clip from when the machine was running but not fabricating and apply standard audio background noise normalization techniques to clarify the signal during fabrication. Fourth, we remove frequency information outside the important band for that particular machine, as determined by a cursory visual inspection of its spectrograms. Fifth, for each frequency in a library audio clip, we average its magnitude across all frames in the audio clip, retaining only the average value for each of its frequencies. After this processing has been done for an example angle audio clip, we save the results in the library. We also record any domain constraints specific to that machine, such as inherent limits on the path that its tool head can follow.
during fabrication.

With these signal processing techniques, the audio of each angle $\alpha$ appears very similar to that of three other angles created by mirroring the given angle in each quadrant of the plane (angles $\pm \alpha$ and $180 \pm \alpha$); this introduces ambiguity into reconstruction. We suspect that more sophisticated signal processing techniques that can pick out the secondary tones visible in our spectrograms (and audible to a keen ear) can be used to tell these angles apart, but that remains for future work; we rely on the simpler methods described below.

**Reconstruction.** The reference library is, at present, constructed by an individual with knowledge of processing techniques. The library must be generated only once per machine. Once this pre-processing step is complete, reconstructions can be accomplished by a number of users in parallel. These users need minimal domain expertise; a short training session would suffice. This is the section of the attack that can be molded into a crowdsourcing task.

With the reference library and a clip of object fabrication audio ready, we begin by computing the normalized cross correlation between each frame of the target audio and each library angle. This calculation produces one value for each combination of a target audio frame and a library angle. For each target audio frame, we select the library angle that maximizes the cross correlation value. Then we compute a match filter that shows the chosen library angle for each frame in the audio, from the first frame to the last. We also produce a spectrogram corresponding to the target audio. The spectrogram and the match filter are presented to the user, as shown in Figure 3.5, a screenshot of our interactive reconstruction framework. Here the crowdsourcing phase begins, with a human evaluating and guiding the reconstruction work. To prepare for this role, a user requires brief training in how to recognize changes of angles and the start and stop of tool work in audio frequency spectrograms.

In our current framework implementation, the user has two tasks. The first is to identify the points in the audio at which the tool head changed its angle; these points can be seen quite easily in a spectrogram. The identified points divide the tool head’s path into a series of straight segments. We expect that segmentation can be automated in the future using signal processing
techniques, so that the user’s task is confined to catching irregularities caused by background sounds. The user’s second task is to look at the match filter for each segment and click on the most common match head height(s) among the frames for that segment. The framework can automatically select the most common match head height in a segment, but interaction allows the user to guide the search when she is not satisfied with any of the framework’s automatically-generated objects.

Because of ambiguity, each match head clicked on by the user may correspond to several different potential angles in the reference library. In our experiments, a $k$-sided manufactured object has a reconstruction search space of size $4^k$, and the framework automatically explores this space. Fortunately, manufacturing domain constraints allow us to prune away nearly all of the search space. The constraints we used for the 3D printer are that each object layer should form one or more two-layer closed figures in the plane, surrounded by a one-layer bounding box; each segment in a closed figure should intersect exactly two other segments, one at each of its endpoints; and at the end of a segment, the printer head should change its angle of travel, rather than continuing in a straight line or (unless it is the end of a layer) doubling back on itself. Reconstructions that pass these tests are shown to the user, who can accept or reject them. If unhappy with all the reconstructions shown, the user can click on additional match heads for a segment so that additional angles will be considered.

**Calibration.** To construct a reference library, one needs access to audio recordings of a machine model similar to the machine under attack. This could be a machine owned by the attacker to be used for fabricating stolen designs or processes. Alternatively, the needed calibration pattern could be hidden in the design of an object fabricated on a machine belonging to the victim or a third party, and recorded in an attack launched specifically to gather that information for the library. This calibration attack is discussed in detail in Section 3.4.3.

**Process Parameters.** The sound associated with travel at a particular angle does not depend on where the tool head was located relative to the origin, so both training and test audio could be recorded with the tool head anywhere on the machine bed. In addition, angles that are just a few degrees apart have very different audio signatures, so the recordings need to include
sufficiently many different angles to reconstruct the target figures accurately. Ideally, the recordings should be made with the same machine settings as used when making the target object; as noted earlier, different settings, or parameters, tend to have different characteristic audio signatures.

For example, we found that increasing the feed rate of our 3D printer shifts the key frequencies of the resulting audio in a systematic manner, as shown in Figure 3.6. While to the human eye the pattern in the figure’s three spectrograms is essentially the same (though compressed in time), the shifted frequency magnitudes are quantitatively distinct. We expect that there is a systematic relationship between different settings that can be expressed mathematically, but that remains for future work. Our current implementation assumes that preprocessing has already identified the signature of the machine model being used and the feed rate, so that an angle reference library specific to those parameters can be consulted during reconstruction.

3.3.2 Machine Learning Reconstruction Method

The machine learning method of reconstruction presents an alternative to the crowdsourcing method. The steps are summarized in Figure 3.7. In the first phase of the method, we build a model based on audio and magnetometer training data from fabrication of the same fan shape used to construct the crowdsourcing reference library. In the second phase, we use the model to reconstruct an object from the audio and magnetometer data recorded during its fabrication. After reconstruction, we use domain constraints to fine-tune the reconstructed object. Both the training and reconstruction phases begin with manual segmentation. Unlike the crowdsourcing method, we do not apply noise reduction or other signal processing techniques to the input data for training or reconstruction.

We experimented with a variety of machine learning methods for training: linear and logistic regression, support vector machines, decision trees, and neural networks. Logistic regression performed at least as well as the other techniques; we adopt it here for simplicity. We train a model for the magnetometer data that indicates, for a given travel segment, in which quadrant the angle corresponding to the machine tool head’s direction of travel lies. We train a separate regression model for the audio data that indicates the
exact angle of travel within that quadrant. We found that the addition of a third model based on the accelerometer data did not improve the accuracy of reconstruction, so we use only audio and magnetometer data.

**Segmentation.** As in the crowdsourcing method, we manually label the sensor input data to indicate the times at which the machine tool head changes direction, delineating each segment. If the data is to be used for training, we also manually label each segment to indicate the angle of travel of the machine tool head during that segment. We split the data files into pieces corresponding to individual segments, each to be considered separately during training or reconstruction. The magnetometer data is not recorded at even time intervals, but rather is polled when the values change, so we interpolate the data to be spread evenly at a sampling rate of 100 Hz. Finally, we perform a real Fast Fourier Transform on each individual segment and pass the results to the next step.

**Training.** After the segmentation step, we use the magnetometer training data to build a logistic regression model. Given the magnetometer data corresponding to any possible segment, the model predicts which quadrant the head is moving in with respect to the beginning point of that segment. This approach is different from the crowdsourcing method’s implementation, which does not use magnetometer data, relying instead on a search process and domain constraints to determine the correct quadrant for a segment’s angle.

Similarly, we fit a logistic regression model to the audio training data. Given the audio for any possible segment, the model predicts at which angle within a quadrant the head is moving for that segment. This differs from the crowdsourcing approach, which uses signal processing techniques (normalized cross correlation) to compute the most likely angle corresponding to each individual audio frame and then suggests that the angle chosen for the majority of frames in a segment is the most likely angle for that segment.

The models are configured slightly differently for the magnetometer and audio data. Both models have the inverse of the regularization strength set at small values to increase regularization and thus reduce overfitting. Our magnetometer model only trains with four different labels for each quadrant, so a “one vs all” approach is appropriate, leading us to choose between a stochastic average gradient descent solver and a coordinate descent algo-
rithm. There is a great deal of magnetometer data, so we use a stochastic average gradient descent solver to optimize the speed of training.

In contrast, we want the angle chosen from the audio data to be close as possible to the actual angle; for example, for an angle of 90 degrees, an identification of 89 degrees is far superior to one of 23 degrees. A “one vs all” approach is inappropriate, as we need to capture the entire probability spectrum. We selected the Broyden Fletcher Goldfarb Shanno (BFGS) algorithm for constructing the audio logistic regression model, with the loss minimized set as the multinomial loss fit across the probability spectrum.

Reconstruction. Given segmented sensor data from a fabrication run, the reconstruction step reconstructs the path traced out, including segment lengths and angles. For each segment, the logistic regression models produced in the training step predict the quadrant and angle within that quadrant from the magnetometer and audio data, respectively. The duration of each segment multiplied by the machine’s feed rate gives the physical length of each linear segment in the reconstructed shape.

Domain constraints. We use domain constraints to judge the quality of a reconstructed object; if an object does not pass these tests, the object is modified to satisfy the constraints. The first constraint is that the reconstructed object be a closed figure, within a tolerance threshold (set to 0 in our experiments). If this constraint is met, we output the reconstructed object; otherwise we move on to the next constraint. The second constraint is that the object be symmetrical: the shape produced during the first half of the fabrication run should be a mirror image of the shape produced in the second half, within a certain threshold (set to 0 in our experiments), applied to objects that were known to be symmetrical. If the symmetry constraint is not met, we refine the candidate shape by replacing the values of each pair of corresponding angles in the two halves with their average. Then we test the closed-figure constraint again; if it is not yet satisfied, we replace the lengths of each pair of corresponding segments with their average. We iterate with these constraints until the closed-figure constraint is satisfied.
3.3.3 Attacking With a Phone Call

Sections 3.3.1 and 3.3.2 illustrate how data collected from smartphone sensors can be used to reconstruct the design of an object with high accuracy. We explored a separate approach in parallel that works with only audio data. This approach can be used on any device that has a microphone, including “dumbphones,” tablets, and computers.

While applying side-channel attacks on additive manufacturing equipment is a new concept, recent work has explored one approach: using the sound generated by a 3D printer, researchers reconstructed the G-code used to create an object with an accuracy of 89.72% by applying machine learning algorithms [68]. However, this work takes place in a controlled environment where a high quality microphone is used to collect the equipment’s audio emissions. We also note that the reconstructed G-code is machine specific; reproducing the same object on a different model of 3D printer or on another type of machine, such as a CNC mill, requires nontrivial extra work.

In contrast, we generalize our audio-only approach. We record using the microphone from a nearby smartphone instead of a dedicated microphone and extrapolate this approach to other recording devices. In addition, instead of recording in a controlled environment with carefully placed sensors and reduced background noise, we make a phone call to a smartphone near manufacturing equipment and record the machine’s audio emissions from the other end of the call. The results from an experiments testing this attack are discussed in Section 3.4.4.

3.3.4 Summary of Attack Methods

This section has presented two attack methods and two reconstruction methods. The attack can be performed in two ways:

- Capturing data locally by recording the sensors in a smartphone that has been placed near the manufacturing equipment.

- Recording a phone call placed to a phone that is located near the manufacturing equipment.

Once the attacker has captured data, the object and process can be reconstructed in two ways:
- The crowdsourcing method: once a reference library has been constructed for a machine, users guide a signal processing method that reconstructs segment angles using the audio signal.

- The machine learning method: a machine learning model predicts the segment angles using the audio and magnetometer data.

Section 3.4.5 compares the accuracy of these methods.

### 3.3.5 Equipment Identification

While the data generated by different machines is similar enough that we are able to apply our reconstruction methods, it is also distinct. Each machine has a unique signature, and different types of machines will have distinct sounds corresponding to their manufacturing processes. For example, a mill gives off audio from the spindle spinning and the tool cutting that would not be present in the audio from a 3D printer. The spindle noise alone is sufficient to distinguish between the printer and the mill used in our experiments, but it is not the only differentiating factor. In addition, the motors of each machine will have an acoustic signature. Though they are nominally identical, depending on the configuration of the machine, each motor moves a different amount of weight, including components such as the extruder or spindle, the axes, and the platform. This distinction is apparent even in a single machine: the X and Y motors are nominally identical but display distinct signatures. The frequencies at which the motors emit noise, as a function of the work they are doing, allows us to differentiate between different machine models. While this technique would also work to distinguish between different models of the same machine type—say, two different 3D printers instead of a 3D printer and a mill—a more complex technique would be needed to identify two copies of the same model.
3.4 Experiments

**Setup.** The experiments recorded data from the Lulzbot Taz 5 3D printer and Other Machine Co. Othermill CNC mill shown in Figure 3.2, hereafter referred to as the “printer” and the “mill.” The X axis of each machine is controlled by a stationary stepper motor that drives a carriage on which the tooling (the printer’s extruder and the mill’s spindle and tool) rides. The Y axis of each machine is controlled by a second stationary stepper motor that moves the platform. The printer’s Z axis is controlled by two stepper motors, one on each end, that raise and lower the full X axis. The mill’s Z axis is controlled by a single stepper motor that controls the height of the spindle relative to the X carriage, which remains fixed in height.

The phone records sensor data with a custom Android app. Audio is collected at a 44100 Hz sampling rate. Figure 3.8 illustrates the raw data collected from the accelerometer and magnetometer of a modern phone while recording the 3D printer fabricating a square.

### 3.4.1 Data Quality with Different Devices

To evaluate the quality of the sensor data produced by different devices, we installed the recording app on seven smartphones and one tablet, listed in Table 3.1. To ensure a fair comparison across devices, we placed each device with its lower right corner 1 inch from the rear left corner of the printer. We enabled the app’s recording function while the printer fabricated a simple geometric shape resembling a trapezoid. The printed object and its process parameters were identical in each trial.

The experiment compared the signal to noise ratio (SNR) of the accelerometer and magnetometer in each device. The Samsung Galaxy S6 performed the best overall, with the highest accelerometer SNR and second-highest magnetometer SNR. Surprisingly, the newest model, the Nexus 6P, had the second-lowest accelerometer SNR. The full results are shown in Figure 3.9.

The placement of the sensors inside each device varies. While our other experiments suggest that for the accelerometer this variation will have a negligible impact on the reconstruction quality compared to the variation inherent in the sensor’s quality, the magnetometer works at a much shorter range, and it may be affected.
In evaluating the other sensors’ data quality, we focused on the Galaxy S6.

3.4.2 Data Quality at Different Locations

To determine the impact of distance on data quality, we compared readings from the Samsung Galaxy S6 at different locations relative to the 3D printer. Beginning at the rear left corner, the phone was used to record the same fabrication activity (a simple 45-degree line) as its distance from the printer was incremented by 1 inch. We moved the phone away from the printer in a line approximately 135 degrees from horizontal, so that it remained approximately equidistant from the X and Y motors, as illustrated in Figure 3.10.

This experiment calculated the signal power of the accelerometer and magnetometer readings at each location as a measurement of the effect of distance, as shown in Figure 3.11. The accelerometer, which was measuring the movement of the table on which the printer was placed, had strong readings at all distances from 0 to 18 inches. The strength of the readings decreased slightly with distance, but the output was clear and usable at all distances. In contrast, the magnetometer was measuring a magnetic field, and the strength of a magnetic field drops with the distance cubed. The magnetometer readings were unusable at distances greater than 4 inches. This limitation affects the machine learning method, which uses the magnetometer to determine the quadrant of a segment’s angle of travel.

3.4.3 Data from Different Machines

Different fabrication machines have distinct acoustic signatures, as discussed earlier, which can be used to differentiate or identify the machine recorded in an attack. To substantiate this claim, we compared recordings of the 3D printer with recordings from the mill. The mill’s and printer’s X and Y movements are driven by similar motors in similar configurations: the X motor drives the carriage that carries the tool head, and the Y motor carries the platform. We found that the mill’s movements exhibit a clear, consistent, and uniquely identifiable audio signature, analogous to that of the 3D printer. We demonstrate this signature in Figure 3.12, comparing the spectrograms of the same turbine blade shape made on the printer and on
the mill. There are a number of features that differentiate the two machines. First, the trace is shifted in frequency on the two machines, since each motor on each machine has its own signature. We note, though, that the two traces exhibit the same pattern. The mill has a distinct signature that contains all of the information necessary to reconstruct an object using the same method that has been applied to the 3D printer. Second, there are components of the acoustic signature that are specific to the machine type: the mill’s signature includes the sound of the spindle turning and the tool cutting, while the printer’s signature includes the sound of the extruder fan and the fan that cools the controls.

This suggests that a recording of a simple calibration pattern is all that is needed to train either of our reconstruction methods on most 3D printers and desktop mills, as well as other types of subtractive manufacturing methods operated by stepper motors. This calibration pattern could be hidden in the interior of an object and designed to look like typical 3D printer infill or it could be hidden in the toolpath of a subtractive manufacturing operation. If the attacker asks the operator to manufacture this object and makes a recording, she now has all the information necessary to reconstruct objects and machining conditions from that machine.

3.4.4 Data Quality in a Phone Call Attack

The previous sections focused on the case where phone sensor data was captured by a malicious phone application. If the data was instead captured during a phone call, the audio signal may have been altered by the phone’s automatic noise reduction. Conveniently, the key audio frequencies of factory floor machinery tend to lie in the same range as the human voice, so a phone’s noise reduction does not simply remove the signal.

The phone call attack was tested on both the printer and mill. We collected data from both the printer and mill with the phone placed a few feet away from the machine, recording on the other end of the call. We also collected data where someone spoke during the recording: first, with the speaker approximately two feet from the phone near the running machine, and second, with the remote speaker talking directly into the mouthpiece. The results from each recording, while noisier, are clear and consistent with the audio
recorded directly on a phone located near the machine. The spectrograms
for these calls are shown in Figure 3.13. For example, the spectrograms show
that the same pattern is visible whether a recording is made next to the mill
or recorded through a phone call. We also tested the phone call attack while
people were speaking. The figure illustrates the difference in the audio when
a person is speaking two feet from the device next to the machine and speak-
ing directly into the microphone of the device far from the machine. Even
though the speech overlaps the information-rich regions of the spectrogram,
the trace is not obscured completely and the shape is still clearly visible.

While reconstruction following a phone call attack must rely on audio only,
the phone call technique greatly broadens the scope of the attack. Captur-
ing information from multiple sensors at once requires an appropriate app
to be present on the phone; in contrast, the phone call attack allows any
phone, regardless of its specifications, to capture factory audio with no prior
preparation beyond the attacker being prepared to record the call on the
remote end. More generally, an audio-only attack can be executed using any
device with a microphone, which expands the attack to not only phones but
also tablets, laptops, and other computers, either through malware or by
recording a voice-over-IP (VoIP) call.

3.4.5 Accuracy of Reconstruction

All training and test data for the reconstruction methods was recorded using
a Samsung Galaxy S6 placed within 4 inches of the printer, i.e., close enough
to collect high-quality magnetometer data. We did not try to place the phone
in the exact same position for each run. All of the training and test data for
reconstruction was collected with a printer feed rate of 30 mm/sec.

We implemented the machine learning approach using Python with the
scikit-learn and SciPy libraries. For training data, we recorded a single run
of the 3D printer traversing a 2-layer planar fan shape, shown in Figure 3.4,
that contains 360 different angles of machine head travel.

The crowdsourcing interactive framework consists of components in Mat-
lab, Adobe Audition, and Python. For both the printer and the mill, we
constructed a crowdsourcing reference library from the audio of one pass of
the machine head over the left-hand half of the fan shape; a spectrogram of
the resulting library is shown in Figure 3.3. As mentioned earlier, to speed up the reconstruction search process, we included several constraints in the interactive framework: the reconstructed object should be within a certain threshold of being a closed planar object with no mid-segment self-crossings, and there should be a change of angle at the end of each segment.

As using a crowdsourcing service in our experiments would have required IRB approval, a lengthy process, a single user tested the crowdsourcing task: a colleague with no prior experience using 3D printers or mills or performing audio analysis. She prepared for her tasks by watching and listening to videos of a 3D printer and mill traversing a square, a circle, a triangle, and a turbine blade outline shape. From examining the resulting spectrograms, she learned to recognize the visual signatures in the spectrogram corresponding to the start and stop of the machine’s work on an object, the points where the machine head changes its direction of travel, and tool head movement up and down. She practiced by using the framework to reconstruct the triangle and square made by the 3D printer and the mill. Due to angle ambiguities, her results included the actual objects as well as their mirror reflections. She was then asked to reconstruct unknown objects.

The machine learning and crowdsourcing reconstruction methods compared results for the 3D-printed outlines of a star and an airplane, specifically, a B2 Stealth Bomber. The crowdsourcing user had never seen either design before. The resulting reconstructions and the original designs are shown in Figures 3.14 and 3.15.

The crowdsourcing user successfully reconstructed both the star and the airplane, plus their mirror reflections. However, she described the airplane as a “fish mouth” and was dissatisfied with the result, even revisiting the match filter diagram to consider second-choice values for angles. The human worker successfully and accurately reconstructed a mystery object, even though she was not able to recognize the object and, therefore, make use of contextual clues.
3.5 Defense

As presenting an attack provides no benefit to the community, we also designed and tested a defense. The defense obfuscates the acoustic emissions from manufacturing equipment by playing recordings during production. Since noise removal has been studied extensively \[71, 72, 73, 74]\, instead of playing a random signal, we chose to play recordings of variations of the part being produced that have small dimensional deviations from the original design. The attacker would still be able to determine the general shape of the item being manufactured, which may provide situational awareness about a manufacturer’s capabilities and the current activities in their factory. However, obfuscation can make it harder for the attacker to separate the target audio stream from the others and reconstruct the object’s exact dimensions or process parameters, which is often exactly the information the attacker is attempting to gather. For example, in high-value manufacturing, there may be a hundred wrong ways to make an object, and one way to make it correctly. However, this kind of obfuscation has inherent limits: since every speaker has a unique acoustic signature, in principle, a collection of playback recordings could be identified as such and stripped away, revealing the useful audio stream. However, an effective obfuscation technique that increases the attack cost will still deter many attackers from manufacturing espionage.

To test this hypothesis, we selected eleven similar turbine blade profiles, like that shown in Figure 3.12, and scaled them so that the print time and outline length were approximately the same. The first ten were recorded as they printed individually. The audio recordings from these prints were combined and aligned with a slight stagger at the beginning, and the resultant composite audio was played while the 11th profile printed. Analysis of the composite audio shows that while the fundamental frequencies were reproduced, the harmonics were lost during the combination step. In the eleventh recording, the fundamentals from the first ten turbine blades obscure the data necessary to reconstruct the eleventh, but the harmonics from the eleventh appear clearly; this harmonic data is sufficient for an audio reconstruction. Future work may experiment with combining the audio tracks in a way that preserves the harmonics and matches other features such as amplitude, in order to obfuscate the recording in a way that will significantly raise the cost.
of reconstruction.

Limiting the electromagnetic field generated by manufacturing machinery can also raise the cost of reconstruction by making it expensive or impossible for reconstruction methods to determine which quadrant an angle of travel lies in. Since magnetometer readings drop off with the cube of the distance from the source, one option is to increase the size of a machine’s enclosure. We tested this hypothesis with a large high-end new-model mill at the DMDII, and found that when the phone was placed on the machine’s enclosure, the magnetometer was too far away from the motors to pick up useful readings.

When it is not practical to enlarge an enclosure, improving motor shielding can help. For example, recent research on interference shielding has shown that polymer-matrix composites are effective for electromagnetic interference shielding, due to their light weight, resistance to corrosion, flexibility, and modest cost [75, 76]. Additionally, researchers have shown that composites such as carbon nanofiber-polymer can provide effective shielding for the frequency range of 8.2-12.4 GHz [77]. We suggest the use of composites to cover the stepper motors in manufacturing equipment with a shield thin enough that the motor is not damaged by excessive heat retention, but thick enough to prevent it from broadcasting sensitive information to an adversary.

We expect that, over time, manufacturers will be able to adopt measures such as these to reduce leakage of audio and electromagnetic field information on their factory floors. Until then, manufacturers may consider asking their employees, vendors, and visitors to leave their phones at the door.
3.6 Summary

This chapter introduces a side-channel attack on manufacturing systems. An attacker can use a smartphone to record audio and other sensor data from a manufacturing process and use this data to accurately reconstruct the shape of the object being fabricated as well as some parameters of the process. The attack works on both additive and subtractive manufacturing machines. A defense that obfuscates audio recording by playing back recordings of similar parts is moderately successful.
3.7 Figures and Tables

Figure 3.1: The high level system and attack model. Designs and raw materials are the inputs to a manufacturing process that produces completed parts. A smartphone is placed near the manufacturing process. By analyzing the data its sensors collect, the side-channel attack reconstructs the design and the manufacturing process.
Figure 3.2: The attack setup. For the smartphone attack, a smartphone is placed on the same table as a 3D printer (left) or CNC mill (right). It records the readings of its sensors while an object is being fabricated. For the phone call attack, an attacker on the other end of a call records the call's audio during fabrication.
Figure 3.3: Example spectrograms for audio data: juxtaposed frequency magnitude spectrograms of 360 different angles of machine head travel with a 3D printer (top) and CNC mill (bottom). This data is used in a reference library for the crowdsourced reconstruction method. (Plotted by Anku Adhikari.)
Figure 3.4: The fan shape fabricated to provide example audio for constructing the crowdsourcing reference library and training the machine learning model. The machines trace each ray in both directions, providing 360 audio signatures in increments of 1 degree.
Figure 3.5: The in-progress reconstruction of a 3D-printed star outline using the crowdsourcing method. The match filter (center) is show below its spectrogram (top). Due to domain constraints, the reconstruction (bottom) will be automatically rejected because it is not a closed figure. (Plotted by Anku Adhikari.)
Figure 3.6: Frequency magnitude spectrograms of the audio from a 3D printer making the same three geometric primitive objects (a square, a circle, and a triangle) at three different feed rates (15, 22.5, and 30 mm/sec). The increase in head travel speed changes the spectrogram in a systematic way.
Figure 3.7: An overview of the machine learning approach. Training data is segmented and then used to train a model. Reconstruction data is segmented and then reconstructed using this model.
Figure 3.8: Raw sensor data from the three axes of the accelerometer and magnetometer in a smartphone placed next to the 3D printing of a square. The readings vary predictably, providing additional information that is not fully captured in audio recordings. Readings from the magnetometer contribute to determining the angle of tool head movement, while the accelerometer data delineates changes in direction.
<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Model</th>
<th>OS</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTC</td>
<td>One M8</td>
<td>Android 6.0</td>
<td>Phone</td>
</tr>
<tr>
<td>Huawei</td>
<td>Nexus 6P</td>
<td>Android 6.0</td>
<td>Phone</td>
</tr>
<tr>
<td>LG</td>
<td>G4</td>
<td>Android 5.1</td>
<td>Phone</td>
</tr>
<tr>
<td>LG</td>
<td>Nexus 5</td>
<td>Android 5.1</td>
<td>Phone</td>
</tr>
<tr>
<td>OnePlus</td>
<td>X</td>
<td>Android 5.1</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung</td>
<td>Galaxy S6</td>
<td>Android 5.1</td>
<td>Phone</td>
</tr>
<tr>
<td>Lenovo</td>
<td>K30-T</td>
<td>Android 4.4</td>
<td>Phone</td>
</tr>
<tr>
<td>Nvidia</td>
<td>Shield</td>
<td>Android 5.1</td>
<td>Tablet</td>
</tr>
</tbody>
</table>

Table 3.1: Compared devices. Sensor readings from these devices were compared to judge which has the best signal-to-noise ratio.
Figure 3.9: The signal-to-noise ratio (SNR) of the accelerometer Y axis and magnetometer Z axis readings on different devices. The Samsung Galaxy S6, with the highest SNR for its accelerometer and second-highest SNR for its magnetometer, is the best overall.
Figure 3.10: The quality of the Galaxy S6’s sensor recordings was tested at different distances from the printer. Starting from the rear left corner, approximately midway between the X and Y motors, the phone was moved away at approximately a 135° angle.
Figure 3.11: The signal power of a phone’s sensor recordings at different locations from the machine. While the magnetometer drops off sharply with distance, the accelerometer readings are strong at all locations on the table.
Figure 3.12: A comparison of the audio from the 3D printer (top) and the CNC mill (bottom) while fabricating the outline of a turbine blade (left top and left bottom). The annotations in the spectrograms illustrate how they correspond to the shape. For clarity, the spectrogram shown from the mill was recorded while the spindle was spinning but on a higher plane than the material to avoid cutting noise. The spectrograms display the trace with a consistent shape, even across different machines; each contains sufficient information to reconstruct the shape.
Figure 3.13: A comparison of recordings of the turbine blade shape recorded via a phone call instead of locally. The shape was recorded while the spindle was spinning but on a higher plane than the material to avoid cutting noise. The same shape was recorded on the remote phone with three levels of speech: silent (top), with a person speaking approximately 2 feet from the mouthpiece of the phone placed next to the mill (center), and with the remote caller speaking directly into the phone mouthpiece (bottom). While speech overlaps with the frequencies that indicate machining and movement, the traces are not fully obscured.
Figure 3.14: A comparison of the two reconstruction methods tested on the star shape. Once constraints were applied, both methods were highly accurate.
Figure 3.15: A comparison of the two reconstruction methods tested on the B2 Bomber shape. Once constraints were applied, both methods were highly accurate.
CHAPTER 4
CONCLUSIONS AND
RECOMMENDATIONS

New technologies are both improving and besieging manufacturing. While Internet-enabled devices allow for long-distance communication and remote work on manufacturing projects, one of the most prevalent types of these devices, smartphones, are able to exploit an avenue of attack on manufacturing equipment.

The first of the projects described in this work explores the remote execution of analytical models and the design of user interfaces for these models. These models are part of a larger tool, the Digital Manufacturing Commons, that allows users in different locations to collaborate. Splitting the components of the DMC in a hybrid cloud architecture allows users finer control over their interaction with the DMC, as they can develop models and interfaces locally. These models can then be hosted locally or published to the DMC, and, regardless of the location, be linked to custom user interfaces. This remote model execution, coupled with the custom interfaces it enables, is flexible and powerful.

Good user interfaces provide a seamless, smooth user experience for the user’s interaction with a product. The customized inputs, outputs, animation, and data visualizations developed for the Model Rocket Design and Beer Distribution Game examples are improvements on the simple text field interface currently supported by the DMC. The feedback that these interfaces provide to users is critical in selecting model inputs, whether the users are designing a product or optimizing based on historical context.

While the interfaces have received positive feedback from other members of the DMC team, they have not yet been formally evaluated in user testing. Future work in this area includes the development of a test such as an A/B test to assess the contributions of the individual components of the interface: for example, comparing the sliders in the Model Rocket Design interface to alternative input methods. Such testing would also be able to evaluate other
aspects of the design, such as layout.

The most critical piece of future work is to finish the development of the DMC’s communication channels through the REST Services machine so that interfaces can be remotely linked to models. Without this piece, the hybrid cloud architecture is still theoretical. Once the architecture is in place, the communication between the example interfaces and their models will need to be tested, and then they can be deployed to the DMDII’s instance of the DMC as examples.

Likewise, there is additional development and evaluation that would benefit the side-channel attack, which explores the collection of data from a factory floor with a smartphone and the use of this data to reconstruct the design and process parameters of an object being fabricated. The attack could be expanded to make use of information from other sensors in the smartphone. For example, the accelerometer accurately designates when the print head changes direction; while, at present, these demarcations are made manually, the process could be automated with the use of the accelerometer data. While we have not yet found a use for the information recorded by the gyroscope, we expect that this data could also be put to use.

The two reconstruction methods used in the attack can also be refined. The methods have been tested on 2D outlines, but the attack is meant to ultimately reconstruct complex 3D objects. The accuracy of these methods can be improved by means such as adding additional constraints and optimizing the machine learning parameters. In addition, the crowdsourcing method should be tested with a larger crowd than a single user.

The Internet is changing the landscape of manufacturing. Remote model execution is only possible because of the Internet-connected devices that run the separate machines in the DMC stack. The side-channel attack is only viable because of the prevalence of relatively inexpensive devices, such as smartphones, that have a variety of sensors built in. As manufacturers further leverage the Internet through means such as Internet-of-Things data collection devices, they must also be aware of and adopt new security measures.
REFERENCES


79


[45] “Kaspersky lab survey: One in every five manufacturing businesses has lost intellectual property to security breaches within the past year,” Kaspersky Lab press release, August 2014.


APPENDIX A

CREATING AND RUNNING DOME MODELS

A.1 Using DOME

DOME consists of two separate applications: the DOME server and the DOME client. The client is a graphical user interface used to build models, deploy models to the server, run models that are hosted on the server, and manage the organization of models deployed to the server. The server, which runs in the background, stores and executes models. The DOME client has four modes corresponding to these actions: Build, Deploy, Run, and Server.

A user begins in Build mode, where she chooses to create a new model, selecting from the list of available programs (Figure A.1). For this example, she chooses a DOME Model, which is a native model in the form of a Jython script. The script is contained within DOME and does not require an external compiler or application to run.

The user adds a Procedural Relation, which relates inputs to outputs and provides the model with a skeleton for its functionality. She adds three variables as inputs, assigning them names and units (Figure A.2) from an extensive list of units DOME recognizes (Figure A.3).

The user writes a simple script, \( V = I \times R \), in the script windows relating the variables. She then edits the model’s Causality, which tells DOME how the variables interact (Figure A.4). DOME will determine which variables are the outputs and automatically sort them (Figure A.5).

After the model is complete, the user deploys it to the server so that it can be run. She switches the client to Deploy mode and follows the prompts to deploy her model (Figure A.6). Models can be deployed to any DOME server for which the user has credentials, either on the same machine or a remote machine. If the user makes changes to a model, it can be redeployed and updated on the server.
To run the model, the user switches the client to Run mode and logs in to the server where the model is deployed. The user navigates to the model in the directory and opens one of its deployed interfaces. After changing the values in the input fields, she presses “submit.” The output fields update when the execution is complete (Figure A.7).
A.2 Figures

Figure A.1: The types of models DOME supports. Some allow users to execute scripts; the remainder are plugins for other programs.
Figure A.2: The user names and selects units for variables she adds to a model.
Figure A.3: The unit chooser. DOME recognizes an extensive list of units, though it does not always recognize different orders of magnitude. For example, DOME recognizes both meters and centimeters as units of length, but it recognizes only Amperes and not milliAmperes as units of current.
Figure A.4: DOME’s Causality windows. The user marks boxes to inform DOME which variables depend on which others (top). DOME condenses this information (bottom) and automatically determines which variables are outputs. Here, \( V \) depends on \( I \) and \( R \).
Figure A.5: The procedural relation that structures the DOME Model. Based on the causality specified by the user, DOME has assigned $I$ and $R$ as inputs and $V$ as an output. The script entered in the lower section is the analytical model DOME will execute when the model is run.
Figure A.6: DOME’s Deploy mode. The user follows prompts to deploy or redploy a model to the DOME server. Once the model is hosted on the server, if it is deployed to the public folder, anyone who logs in to the server will be able to run it.
Figure A.7: DOME’s Run mode. The user enters values in the input fields and presses submit. The colors change to indicate to the user which fields are mutable (inputs) and which are fixed (outputs) as well as whether the model needs to be run again to reflect new values.
APPENDIX B

DOME MODELS AND INTERFACE SCRIPTS

This appendix describes the DOME Models and interfaces discussed in Chapter 2. The Models’ Jython scripts are included, as are the JavaScript scripts that make the interfaces interactive.\(^1\)

B.1 Model Rocket Design

B.1.1 DOME Models

Drag Coefficient Calculation

\[
\begin{align*}
1 & \quad \text{CdNBT} = 0.0167 * \text{lenVal} / \text{diaVal} + 0.0367 \\
2 & \quad \text{CdBS} = 0.029 / \text{math.sqrt(CdNBT)} \\
3 & \quad \text{if} \quad \text{finThickness/Cr} < 0.4: \\
4 & \quad \quad \text{CdFstar} = 0.15 * \text{finThickness/Cr} + 0.011 \\
5 & \quad \text{else:} \\
6 & \quad \quad \text{CdFstar} = 0.125 * \text{finThickness/Cr} + 0.012 \\
7 & \quad \text{if} \quad \text{shapeVal} = "\text{elliptical":} \\
8 & \quad \quad \text{SF} = 0.785 * \text{Cr} * \text{widthVal} \\
9 & \quad \quad \text{elif} \quad \text{shapeVal} = "\text{tapered":} \\
10 & \quad \quad \quad \text{SF} = 0.5 * \text{widthVal} * 4/3 * \text{Cr} \\
11 & \quad \quad \text{else:} \\
12 & \quad \quad \quad \text{SF} = \text{heightVal} * \text{widthVal} \\
13 & \quad \text{CdFInt} = \text{CdFstar} / \text{SBT} * 0.5 * \text{SF} * \text{finVal} \\
14 & \quad \text{CdLL} = (1.2 * \text{SLL} + 0.0045 * \text{SLLW})/\text{SBT}
\end{align*}
\]

\(^1\)The syntax highlighting in this appendix is provided by a custom Code Style [78].
Altitude Calculation

```python
import math

mBoost = mBodyTube + mNostCone + mFins + mEmpty + 0.5*mFull + mLL
mCoast = mBodyTube + mNostCone + mFins + mEmpty + mLL

SBT = 0.25 * math.pi * diaVal * diaVal * diaVal * diaVal

k = 0.5 * densityAir * Cd * SBT

t = impulse / thrust

x = 2 * k/mBoost * math.sqrt((thrust - mBoost * gravity) / k)

v = math.sqrt((thrust - mBoost * gravity) / k) * (1 - math.e ^ (-x*t)) / (1 + math.e ^ (-x*t))

yb = -mBoost/(2*k) * math.log((thrust - mBoost*gravity - k * v * v)/(thrust - mBoost*gravity))

yc = mCoast/(2*k) * math.log((mCoast*gravity + k * v * v)/(mCoast*gravity))
```

Cd = CdNBT + CdBS + CdFInt + CdLL
B.1.2 Interface

```javascript
rocketCalc = function() {

var diaVal = $('input[name=rocketDia]:checked').val(); // mm
var lenVal = $('#slider-len').slider("option", "value"); // mm
var motorVal = $('input[name=rocketMotor]:checked').val();
var shapeVal = $('input[name=finShape]:checked').val();
var finVal = $('input[name=finNum]:checked').val();
var heightVal = $('#slider-plan-height').slider("option", "value"); // mm
var widthVal = $('#slider-plan-width').slider("option", "value"); // mm

mFull = rocketMotors[motorVal]["massProp"];
mEmpty = rocketMotors[motorVal]["massEmpty"];
thrust = rocketMotors[motorVal]["thrust"];
impulse = rocketMotors[motorVal]["impulse"];

mBodyTube = bodyMasses[diaVal]["body"];  
mNoseCone = bodyMasses[diaVal]["nose"];  

alert("DOME would calculate and return the height if this were linked.");

};

updateHeight = function() {

heightVal = $('#slider-plan-height').slider("option", "value");

taperedArr = [  // points for tapered fin
    [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
    [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
    [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal / 3)],
    [inner_width/2 + diaVal/2, lenScale(lenVal)],
```
[[inner_width/2 - diaVal/2, lenScale(lenVal)],
 [inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal / 3)],
 [inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
 [inner_width/2 - diaVal/2, lenScale(lenVal) - heightVal],
 [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
];

svgRoc.select("#finRect")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("y", lenScale(lenVal) - finScaleH(heightVal))
  .attr("height", finScaleH(heightVal));

svgRoc.select("#finEll")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("cy", lenScale(lenVal) - finScaleH(heightVal) /2)
  .attr("ry", finScaleH(heightVal)/2);

svgRoc.select("#finTap")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("d", lineTaper(taperedArr));
};

updateWidth = function() {
  widthVal = $('"#slider-plan-width"').slider("option", "value");

  taperedArr = [  // points for tapered fin
    [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
    [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
    [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal / 3)],
    [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
    [inner_width/2 - diaVal/2, lenScale(lenVal) - heightVal],
    [inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal / 3)],
    [inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
    [inner_width/2 - diaVal/2, lenScale(lenVal) - heightVal],
  ];

  var taperedPath =
      taperedArr.slice(0, taperedArr.length - 2)
        .reduce(function(p, c) {
          return p + " L " + c + " R " + c;
        },"
      " Z" " L " taperedArr[0] + " R " taperedArr[0];

  var newPath =
      taperedPath.slice(0, taperedPath.length - 2)
        .reduce(function(p, c) {
          return p + " R " + c + " L " + c;
        },"
      " Z" " L " taperedArr[0] + " R " taperedArr[0];

  svgRoc.select("#finTap")
    .attr("d", newPath);
[inner_width/2 + diaVal/2, lenScale(lenVal)],
[inner_width/2 - diaVal/2, lenScale(lenVal)],
[inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal / 3)],
[inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
[inner_width/2 - diaVal/2, lenScale(lenVal) - heightVal],
[inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
]

svgRoc.select("#finRect")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("x", inner_width/2 - finScaleW(widthVal)/2)
  .attr("width", finScaleW(widthVal));

svgRoc.select("#finEll")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("cx", inner_width/2)
  .attr("rx", finScaleW(widthVal)/2);

svgRoc.select("#finTap")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("d", lineTaper(taperedArr));

svgRoc.select("#fin_3_1")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("width", finScaleW(widthVal)/2)
  .attr("x", topViewX - finScaleW(widthVal)/2)

svgRoc.select("#fin_3_2")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("width", finScaleW(widthVal)/2)
  .attr("x", topViewX - finScaleW(widthVal)/2)
```javascript
111   svgRoc.select("#fin_3_3")
112     .transition()
113     .duration(200)
114     .ease("linear")
115     .attr("width", finScaleW(widthVal)/2)
116     .attr("x", topViewX - finScaleW(widthVal)/2)
117
118   svgRoc.select("#fin_3_4")
119     .transition()
120     .duration(200)
121     .ease("linear")
122     .attr("width", finScaleW(widthVal)/2)
123     .attr("x", topViewX - finScaleW(widthVal)/2)
124
125   svgRoc.select("#fin_3_5")
126     .transition()
127     .duration(200)
128     .ease("linear")
129     .attr("width", finScaleW(widthVal)/2)
130     .attr("x", topViewX - finScaleW(widthVal)/2)
131
132   svgRoc.select("#fin_3_6")
133     .transition()
134     .duration(200)
135     .ease("linear")
136     .attr("width", finScaleW(widthVal)/2)
137     .attr("x", topViewX - finScaleW(widthVal)/2)
138
139 }
140
141 updateLength = function() {
142
143   lenVal = $("#slider-len").slider("option", "value");
144
145   taperedArr = [ // points for tapered fin
146     [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
147     [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
148     [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal / 3)],
149     [inner_width/2 + diaVal/2, lenScale(lenVal)],
150     [inner_width/2 - diaVal/2, lenScale(lenVal)],
151 ```
[inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal / 3)],
[inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
[inner_width/2 - diaVal/2, lenScale(lenVal) - heightVal],
[inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
];

svgRoc.select("#leftBtLine")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("y2", lenScale(lenVal));

svgRoc.select("#rightBtLine")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("y2", lenScale(lenVal));

svgRoc.select("#fillRect")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("height", lenScale(lenVal) - ogiveHeight);

svgRoc.select("#finRect")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("y", lenScale(lenVal) - finScaleH(heightVal));

svgRoc.select("#baseLine") // base
  .transition()
  .duration(200)
  .ease("linear")
  .attr("y1", lenScale(lenVal))
  .attr("y2", lenScale(lenVal));

svgRoc.select("#finEll")
  .transition()
  .duration(200)
  .ease("linear")
.attr("cy", lenScale(lenVal) - finScaleH(heightVal) /2);

svgRoc.select("#finTap")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("d", lineTaper(taperedArr));

updateDia = function() {
  var diaVal = $("input[name=rocketDia]:checked").val();
  var ogiveHeight = diaVal * 3.05;
  var radius = diaVal * 9;
  var radArray = [];

  for (i = 0; i < points; i++) {
    radArray.push(diaVal*9);
  }

  var ogiveArr = d3.zip(radArray, points);

  taperedArr = [  // points for tapered fin
    [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
    [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
    [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal / 3)],
    [inner_width/2 + diaVal/2, lenScale(lenVal)],
    [inner_width/2 - diaVal/2, lenScale(lenVal)],
    [inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal / 3)],
    [inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
    [inner_width/2 - diaVal/2, lenScale(lenVal) - heightVal],
    [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
  ];

  svgRoc.select("#leftBtLine")
  .transition()
.duration(200)
.attr("x1", inner_width/2 - diaVal/2)
.attr("y1", ogiveHeight)
.attr("x2", inner_width/2 - diaVal/2);

svgRoc.select("#rightBtLine")
  .transition()
  .duration(200)
  .attr("x1", inner_width/2 + diaVal/2)
  .attr("y1", ogiveHeight)
  .attr("x2", inner_width/2 + diaVal/2);

svgRoc.select("#fillRect")
  .transition()
  .duration(200)
  .attr("width", diaVal)
  .attr("x", inner_width/2 - diaVal/2);

svgRoc.select("#baseLine")
  .transition()
  .duration(200)
  .attr("x1", inner_width/2 - diaVal/2)
  .attr("x2", inner_width/2 + diaVal/2);

svgRoc.select("#leftOgLine")
  .transition()
  .duration(200)
  .attr("d", lineOgL(ogiveArr))
  .attr("transform", "translate(" + (inner_width/2 -
      diaVal/2 + radius) + "," + ogiveHeight + ")");

svgRoc.select("#rightOgLine")
  .transition()
  .duration(200)
  .attr("d", lineOgR(ogiveArr))
  .attr("transform", "translate(" + (inner_width/2 +
      diaVal/2 - radius) + "," + ogiveHeight + ")");

svgRoc.select("#finTap")
  .transition()
  .duration(200)
  .ease("linear")
  .attr("d", lineTaper(taperedArr));
updateFinShape = function() {
  switch($("input[name=finShape]:checked").val()){
    case "rectangular" :
      svgRoc.select("#finRect")
        .transition()  
        .duration(10)
        .style("stroke-width", 2) 
      svgRoc.select("#finEll")
        .transition() 
        .duration(10)
        .style("stroke-width", 0) 
      svgRoc.select("#finTap")
        .transition()  
        .duration(10)
        .style("stroke-width", 0) 
      break;
    case "elliptical" :
      svgRoc.select("#finRect")
        .transition() 
        .duration(10)
        .style("stroke-width", 0) 
      svgRoc.select("#finEll")
        .transition() 
        .duration(10)
        .style("stroke-width", 2) 
      svgRoc.select("#finTap")
        .transition() 
        .duration(10)
        .style("stroke-width", 0) 
      break;
    case "tapered" :
      svgRoc.select("#finRect")
        .transition() 
        .duration(10)
        .style("stroke-width", 0) 
      svgRoc.select("#finEll")
        .transition() 
        .duration(10)
        .style("stroke-width", 0) 
  }
}
svgRoc.select("#finTap")
    .transition()
    .duration(10)
    .style("stroke-width", 2)
break;
default:
    alert("fin shape not found");
}]
}

updateFinNum = function() {
    switch($("input[name=finNum]:checked").val()){
        case "3" :
            svgRoc.select("#fin_3_1")
                .transition()
                .duration(10)
                .style("stroke-width", 2)
            svgRoc.select("#fin_3_2")
                .transition()
                .duration(10)
                .style("stroke-width", 0)
            svgRoc.select("#fin_3_3")
                .transition()
                .duration(10)
                .style("stroke-width", 0)
            svgRoc.select("#fin_3_4")
                .transition()
                .duration(10)
                .style("stroke-width", 0)
            svgRoc.select("#fin_3_5")
                .transition()
                .duration(10)
                .style("stroke-width", 2)
            svgRoc.select("#fin_3_6")
                .transition()
                .duration(10)
                .style("stroke-width", 2)
        break;
        case "4" :
            svgRoc.select("#fin_3_1")
                .transition()
                .duration(10)
.style("stroke-width", 2)

    svgRoc.select("#fin_3_2")
    .transition()
    .duration(10)
    .style("stroke-width", 2)

    svgRoc.select("#fin_3_3")
    .transition()
    .duration(10)
    .style("stroke-width", 2)

    svgRoc.select("#fin_3_4")
    .transition()
    .duration(10)
    .style("stroke-width", 2)

    svgRoc.select("#fin_3_5")
    .transition()
    .duration(10)
    .style("stroke-width", 0)

    svgRoc.select("#fin_3_6")
    .transition()
    .duration(10)
    .style("stroke-width", 0)

    break;
    default :
    alert("fin number not found");
}

$(document).ready(function(){

    // set constants
    densityAir = 1.22, // kg/m^3
    gravity = 9.81, // m/s^2
    densityBalsa = 144.2, // kg/m^3
    finThickness = 0.0015875, // m, = 1/16"
    mLL = 0.0001, // kg
    SLL = 0.00000302, // m^2
    SLLW = 0.0006144; // m^2

    //rocketMotor units: kg, N*s, N
    //massProp = mass including propellant, massEmpty = mass after firing
    //thrust = average value
    rocketMotors = {

A8 : {massProp: 0.01389, massEmpty: 0.01019, thrust: 4.03, impulse: 2.15},
B4 : {massProp: 0.0182, massEmpty: 0.010, thrust: 12.75, impulse: 4.85},
B6 : {massProp: 0.0188, massEmpty: 0.0097, thrust: 5.03, impulse: 4.90},
C6 : {massProp: 0.0241, massEmpty: 0.0094, thrust: 4.74, impulse: 9},

};

bodyMasses = {
  18 : {body: 0.18, nose: 2.3},
  24 : {body: 0.24, nose: 7.5}
};

margin = {top: 10, right: 10, bottom: 10, left: 10},
padding = {top: 10, right: 10, bottom: 10, left: 10},
outer_width = 170,
outer_height = 630,
inner_width = outer_width - margin.left - margin.right,
inner_height = outer_height - margin.top - margin.bottom;
radians = Math.PI,
points = 10,
lenMin = 200;
lenMax = 450,
hMin = 25,
hMax = 75,
wMin = 75,
wMax = 150,
ogiveAngle = 0.107
finThPx = 3; // fin thickness in pixels for svg

// initialize sliders
$( "#slider-len" ).slider({
  range: "min",
  value: 330,
  min: lenMin,
  max: lenMax,
  step: 10,
  slide: function( event, ui ) {
    $( "#amount-len" ).val( ui.value );
```javascript
$( "#slider-plan-height" ).slider({
    range: "min",
    value: 50,
    min: hMin,
    max: hMax,
    step: 5,
    slide: function( event, ui ) {
        $( "#amount-plan-height" ).val( ui.value );
    }
});

$( "#slider-plan-width" ).slider({
    range: "min",
    value: 110,
    min: wMin,
    max: wMax,
    step: 5,
    slide: function( event, ui ) {
        $( "#amount-plan-width" ).val( ui.value );
    }
});

// read values from sliders, radio buttons

$( "#amount-len" ).val( $( "#slider-len" ).slider( "value" ) );
$( "#amount-plan-height" ).val( $( "#slider-plan-height" ).slider( "value" ) );
$( "#amount-plan-width" ).val( $( "#slider-plan-width" ).slider( "value" ) );

diaVal = $("input[name=rocketDia]:checked").val(); // mm
lenVal = $("#slider-len").slider("option", "value"); // mm
motorVal = $("input[name=rocketMotor]:checked").val();
shapeVal = $("input[name=finShape]:checked").val();
finVal = $("input[name=finNum]:checked").val();
heightVal = $("#slider-plan-height").slider("option", "value"); // mm
widthVal = $("#slider-plan-width").slider("option", "value"); // mm

radius = diaVal * 9,
```
ogiveHeight = diaVal * 3.05;

var radArray = [];

for (i = 0; i < points; i++) {
    radArray.push(diaVal*9);
}

var ogiveArr = d3.zip(radArray, points);

// define scales and lines/paths for drawing

angleOgL = d3.scale.linear()
    .domain([0, points-1])
    .range([1.5*radians, (1.5+ogiveAngle)*radians]);

lineOgL = d3.svg.line.radial()
    .interpolate("basis")
    .tension(0)
    .angle(function(d, i) { return angleOgL(i); });

angleOgR = d3.scale.linear()
    .domain([0, points-1])
    .range([(0.5-ogiveAngle)*radians, 0.5*radians]);

lineOgR = d3.svg.line.radial()
    .interpolate("basis")
    .tension(0)
    .angle(function(d, i) { return angleOgR(i); });

finScaleH = d3.scale.linear()
    .domain([hMin, hMax])
    .range([hMin, hMax]);

finScaleW = d3.scale.linear()
    .domain([wMin, wMax])
    .range([wMin, wMax]);

lineTaper = d3.svg.line()
    .x(function(d) { return d[0]; })
    .y(function(d) { return d[1]; })
lenScale = d3.scale.linear()
    .domain([lenMin, lenMax])
    .range([lenMin, lenMax]);

var taperedArr = [
    // points for tapered fin
    [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
    [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
    [inner_width/2 + widthVal/2, lenScale(lenVal) - (heightVal / 3)],
    [inner_width/2 + diaVal/2, lenScale(lenVal)],
    [inner_width/2 - diaVal/2, lenScale(lenVal)],
    [inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal / 3)],
    [inner_width/2 - widthVal/2, lenScale(lenVal) - (heightVal * 2/3)],
    [inner_width/2 - diaVal/2, lenScale(lenVal) - heightVal],
    [inner_width/2 + diaVal/2, lenScale(lenVal) - heightVal],
];

svgRoc = d3.select(document.getElementById("svgRocket"))
    .attr("width", outer_width)
    .attr("height", outer_height)
    .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

svgRoc.append("rect") // border
    .attr("x", -margin.left)
    .attr("y", -margin.top)
    .attr("height", outer_height)
    .attr("width", outer_width)
    .attr("class", "line")
    .attr("id", "svgBorder")

svgRoc.append("rect") // rectangular fin
    .attr("x", inner_width/2 - finScaleW(widthVal)/2)
    .attr("y", lenScale(lenVal) - finScaleH(heightVal))
    .attr("height", finScaleH(heightVal))
```xml
.attr("width", finScaleW(widthVal))
.attr("class", "line")
.attr("id", "finRect")
.style("stroke-width", 0);

svgRoc.append("ellipse") // elliptical fin
  .attr("cx", inner_width/2)
  .attr("cy", lenScale(lenVal) - finScaleH(heightVal) /2)
  .attr("ry", finScaleH(heightVal)/2)
  .attr("rx", finScaleW(widthVal)/2)
  .attr("class", "line")
  .attr("id", "finEll")
  .style("stroke-width", 0);

svgRoc.append("path") // tapered fin
  .attr("class", "line")
  .attr("id", "finTap")
  .attr("d", lineTaper(taperedArr))
  .style("stroke-width", 0);

svgRoc.append("rect") // fill body tube to hide fin intersection
  .attr("x", inner_width/2 - diaVal/2)
  .attr("y", ogiveHeight)
  .attr("height", lenScale(lenVal) - ogiveHeight)
  .attr("width", diaVal)
  .attr("class", "opaque")
  .attr("id", "fillRect")
  .style("stroke-width", 0);

svgRoc.append("line") // left vertical body tube
  .attr("x1", inner_width/2 - diaVal/2)
  .attr("y1", ogiveHeight)
  .attr("x2", inner_width/2 - diaVal/2)
  .attr("y2", lenScale(lenVal))
  .attr("class", "line")
  .attr("id", "leftBtLine");

svgRoc.append("line") // right vertical body tube
  .attr("x1", inner_width/2 + diaVal/2)
  .attr("y1", ogiveHeight)
  .attr("x2", inner_width/2 + diaVal/2)
```
596  .attr("y2", lenScale(lenVal))
597  .attr("class", "line")
598  .attr("id", "rightBtLine");
599
600  svgRoc.append("line") // base
601    .attr("x1", inner_width/2 - diaVal/2)
602    .attr("y1", lenScale(lenVal))
603    .attr("x2", inner_width/2 + diaVal/2)
604    .attr("y2", lenScale(lenVal))
605    .attr("class", "line")
606    .attr("id", "baseline");
607
608  svgRoc.append("path") // left arc of ogive
609    .attr("class", "line")
610    .attr("id", "left0gLine")
611    .attr("d", lineOgl(ogiveArr))
612    .attr("transform", "translate(" + (inner_width/2 - diaVal/2 + radius) + "," + ogiveHeight + ")");
613
614  svgRoc.append("path") // right arc of ogive
615    .attr("class", "line")
616    .attr("id", "right0gLine")
617    .attr("d", lineOgR(ogiveArr))
618    .attr("transform", "translate(" + (inner_width/2 + diaVal/2 - radius) + "," + ogiveHeight + ")");
619
620  updateFinShape();
621
622  topViewX = inner_width/2,
623  topViewY = lenScale(lenMax) + wMax/2 + 10;
624
625  svgRoc.append("rect")
626    .attr("x", topViewX - finScaleW(widthVal)/2)
627    .attr("y", topViewY - finThPx/2)
628    .attr("height", finThPx)
629    .attr("width", finScaleW(widthVal)/2)
630    .attr("class", "line")
631    .attr("id", "fin_3_1")
632    .style("stroke-width", 0);
633
634  svgRoc.append("rect")
635    .attr("x", topViewX - finScaleW(widthVal)/2)
636    .attr("y", topViewY - finThPx/2)
svgRoc.append("rect")
    .attr("x", topViewX - finScaleW(widthVal)/2)
    .attr("y", topViewY - finThPx/2)
    .attr("height", finThPx)
    .attr("width", finScaleW(widthVal)/2)
    .attr("class", "line")
    .attr("id", "fin_3_4")
    .attr("transform", "rotate(" + 270 + ", " + topViewX + ", " + topViewY + ")")
    .style("stroke-width", 0);

svgRoc.append("rect")
    .attr("x", topViewX - finScaleW(widthVal)/2)
    .attr("y", topViewY - finThPx/2)
    .attr("height", finThPx)
    .attr("width", finScaleW(widthVal)/2)
    .attr("class", "line")
    .attr("id", "fin_3_5")
    .attr("transform", "rotate(" + 120 + ", " + topViewX + ", " + topViewY + ")")
    .style("stroke-width", 0);

svgRoc.append("rect")
    .attr("x", topViewX - finScaleW(widthVal)/2)
    .attr("y", topViewY - finThPx/2)
    .attr("height", finThPx)
    .attr("width", finScaleW(widthVal)/2)
    .attr("class", "line")
    .attr("id", "fin_3_6")
    .attr("transform", "rotate(" + 180 + ", " + topViewX + ", " + topViewY + ")")
    .style("stroke-width", 0);
.attr("y", topViewY - finThPx/2)
.attr("height", finThPx)
.attr("width", finScalarW(widthVal)/2)
.attr("class", "line")
.attr("id", "fin_3_6")
.attr("transform", "rotate(" + 240 + ", " + topViewX + ", " + topViewY + ")")
.style("stroke-width", 0);

svgRoc.append("circle")
  .attr("cx", topViewX)
  .attr("cy", topViewY)
  .attr("r", diaVal/2)
  .attr("class", "line")
  .attr("id", "topViewCircle");

updateFinNum();

// update dimensions as sliders change

$('slider-plan-height').on("slide", function(event, ui) {
  updateHeight();
});

$('slider-plan-height').on("slidechange", function(){
  updateHeight();
});

$('slider-plan-width').on("slide", function(event, ui) {
  updateWidth();
});

$('slider-plan-width').on("slidechange", function(){
  updateWidth();
});

$('slider-len').on("slide", function(event, ui) {
  updateLength();
});

$('slider-len').on("slidechange", function(){
  updateLength();
});
$("input[name=rocketDia]").on("change", function(){
    updateDia();
});

$("input[name=finShape]").on("change", function(){
    updateFinShape();
});

$("input[name=finNum]").on("change", function(){
    updateFinNum();
});
B.2 The Beer Distribution Game

B.2.1 DOME Model

```python
1 # import random
2
3 # fc = factory
4 # rw = regional warehouse
5 # wh = wholesaler
6 # rt = retailer
7
8 round_out = round_in + 1
9
10 # the stock at the end of the round initially set equal to
11    stock at beginning of round
12
13 fc_stock_out = fc_stock
14 rw_stock_out = rw_stock
15 wh_stock_out = wh_stock
16 rt_stock_out = rt_stock
17
18 # first, accept incoming goods
19 fc_stock_out += fc_incoming_goods
20 fc_incoming_goods = 0
21 fc_incoming_goods += fc_incoming_goods
22 rw_stock_out += rw_incoming_goods
23 wh_stock_out += wh_incoming_goods
24 rt_stock_out += rt_incoming_goods
25 rw_incoming_goods = 0
26 wh_incoming_goods = 0
27 rt_incoming_goods = 0
28
29 # pay for incoming goods
30 fc_budget -= fc_incoming_goods * fc_cost_incoming_goods
31 rw_budget -= rw_incoming_goods * rw_cost_incoming_goods
32 wh_budget -= wh_incoming_goods * wh_cost_incoming_goods
33 rt_budget -= rt_incoming_goods * rt_cost_incoming_goods
34
35 # read incoming orders
36 fc_incoming_orders = rw_placed_orders
37 rw_incoming_orders = wh_placed_orders
```
wh_incoming_orders = rt_placed_orders
rt_incoming_orders = random.random() * (maxDemand - minDemand) + minDemand

# ship outgoing goods
if fc_stock_out >= (fc_incoming_orders + fc_open_orders):
    fc_outgoing_goods = (fc_incoming_orders + fc_open_orders)
    fc_stock_out -= fc_outgoing_goods
else:
    fc_outgoing_goods = fc_stock
    fc_open_orders += (fc_incoming_orders - fc_stock)
    fc_stock_out = 0

if rw_stock_out >= (rw_incoming_orders + rw_open_orders):
    rw_outgoing_goods = (rw_incoming_orders + rw_open_orders)
    rw_stock_out -= rw_outgoing_goods
else:
    rw_outgoing_goods = rw_stock
    rw_open_orders += (rw_incoming_orders - rw_stock)
    rw_stock_out = 0

if wh_stock_out >= (wh_incoming_orders + wh_open_orders):
    wh_outgoing_goods = (wh_incoming_orders + wh_open_orders)
    wh_stock_out -= wh_outgoing_goods
else:
    wh_outgoing_goods = wh_stock
    wh_open_orders += (wh_incoming_orders - wh_stock)
    wh_stock_out = 0

if rt_stock_out >= (rt_incoming_orders + rt_open_orders):
    rt_outgoing_goods = (rt_incoming_orders + rt_open_orders)
    rt_stock_out -= rt_outgoing_goods
else:
    rt_outgoing_goods = rt_stock
    rt_open_orders += (rt_incoming_orders - rt_stock)
    rt_stock_out = 0

# get paid for outgoing goods
fc_budget += fc_outgoing_goods * fc_value_outgoing_goods
rw_budget += rw_outgoing_goods * rw_value_outgoing_goods
wh_budget += wh_outgoing_goods * wh_value_outgoing_goods
rt_budget += rt_outgoing_goods * rt_value_outgoing_goods
# pay backorder cost
fc_budget -= fc_open_orders * fc_cost_open_orders
rw_budget -= rw_open_orders * rw_cost_open_orders
wh_budget -= wh_open_orders * wh_cost_open_orders
rt_budget -= rt_open_orders * rt_cost_open_orders

# pay holding cost
fc_budget -= fc_stock_out * fc_cost_stock_holding
rw_budget -= rw_stock_out * rw_cost_stock_holding
wh_budget -= wh_stock_out * wh_cost_stock_holding
rt_budget -= rt_stock_out * rt_cost_stock_holding

# place orders with user input
rw_placed_orders = rw_order
wh_placed_orders = wh_order
fc_placed_orders = fc_order
rt_placed_orders = rt_order
B.2.2 Interface

```javascript
formAdd = function() {

    // first, send values to DOME
    rw_order
    wh_order
    fc_order
    rt_order
    round_in
    minDemand
    maxDemand
    fc_stock
    wh_stock
    rt_stock
    rw_stock
    fc_incoming_goods
    rw_incoming_goods
    wh_incoming_goods
    rt_incoming_goods

    // [ DOME interfacing code goes here ]

    // store DOME's returned values in variables

    var fcInv = $('inputFcInv').val();
    var rtInv = $('inputRtInv').val();
    var rwInv = $('inputRwInv').val();
    var whInv = $('inputWhInv').val();
    var fcOrd = $('inputFcOrd').val();
    var rtOrd = $('inputRtOrd').val();
    var rwOrd = $('inputRwOrd').val();
    var whOrd = $('inputWhOrd').val();
    var fcBud = $('inputFcBud').val();
    var rtBud = $('inputRtBud').val();
    var rwBud = $('inputRwBud').val();
    var whBud = $('inputWhBud').val();

    // push new data to arrays so it can be added to the graphs

};
```
```javascript
var wk = dataInv.length/4;

dataInv.push({
  stage: "Factory",
  week: wk,
  inventory: Number(fcInv)));
dataInv.push({
  stage: "Retail",
  week: wk,
  inventory: Number(rtInv)));
dataInv.push({
  stage: "Warehouse",
  week: wk,
  inventory: Number(rwInv)));
dataInv.push({
  stage: "Wholesale",
  week: wk,
  inventory: Number(whInv)));
dataOrd.push({
  stage: "Factory",
  week: wk,
  orders: Number(fcOrd)));
dataOrd.push({
  stage: "Retail",
  week: wk,
  orders: Number(rtOrd)));
dataOrd.push({
  stage: "Warehouse",
  week: wk,
  orders: Number(rwOrd)));
dataOrd.push({
  stage: "Wholesale",
  week: wk,
  orders: Number(whOrd)));
dataBud.push({
  stage: "Factory",
  week: wk,
  budget: Number(fcBud)));
dataBud.push({
  stage: "Retail",
  week: wk,
  budget: Number(rtBud)));
dataBud.push({
  stage: "Warehouse",
  week: wk,
  budget: Number(rwBud)));
dataBud.push({
  stage: "Wholesale",
  week: wk,
  budget: Number(whBud)));

// set up material needed to update graphs, such as nests and scales

var stagesInv = d3.nest()
  .key(function(d) { return d.stage; })
  .sortKeys(function(a,b) {
    return sortOrder.indexOf(a) - sortOrder.indexOf(b);
  })
  .entries(dataInv);

stagesInv.forEach(function(s) {
  s.maxInv = Math.max(0, d3.max(s.values, function(d) {
    return d.inventory; }));
  s.minInv = Math.min(0, d3.min(s.values, function(d) {
    return d.inventory; }));
});
```
```javascript
var stagesOrd = d3.nest()
    .key(function(d) { return d.stage; })
    .sortKeys(function(a,b) {
        return sortOrder.indexOf(a) - sortOrder.indexOf(b);
    })
    .entries(dataOrd);

stagesOrd.forEach(function(s) {
    s.maxOrd = Math.max(0, d3.max(s.values, function(d) {
        return d.orders; }));
    s.minOrd = Math.min(0, d3.min(s.values, function(d) {
        return d.orders; }));
});

var stagesBud = d3.nest()
    .key(function(d) { return d.stage; })
    .sortKeys(function(a,b) {
        return sortOrder.indexOf(a) - sortOrder.indexOf(b);
    })
    .entries(dataBud);

stagesBud.forEach(function(s) {
    s.maxBud = Math.max(0, d3.max(s.values, function(d) {
        return d.budget; }));
    s.minBud = Math.min(0, d3.min(s.values, function(d) {
        return d.budget; }));
});

stagesInv.forEach(function(d) {
    svgInv.select("#datalineInv" + d.key)
        .transition()
        .duration(10)
        .attr("d", lineInv(d.values))
});

stagesOrd.forEach(function(d) {
    svgOrd.select("#datalineOrd" + d.key)
        .transition()
        .duration(10)
```

stagesBud.forEach(function(d) {
    svgBud.select("#datalineBud" + d.key)
        .transition()
        .duration(10)
        .attr("d", lineBud(d.values))
});

xScaleInv = d3.scale.linear()
    .range([0, width])
    .domain([
    d3.min(stagesInv, function(s) { return s.values[0].week; }),
    d3.max(stagesInv, function(s) { return s.values[ s.values.length - 1].week; })
    ]);

yScaleInv = d3.scale.linear()
    .range([height, 0])
    .domain([
    d3.min(stagesInv, function(d) { return d.minInv; }),
    d3.max(stagesInv, function(d) { return d.maxInv; })
    ]);

xAxisInv = d3.svg.axis()
    .scale(xScaleInv)
    .orient("bottom")
    .ticks(5);

yAxisInv = d3.svg.axis()
    .scale(yScaleInv)
    .orient("left")
    .ticks(5);

xScaleOrd = d3.scale.linear()
    .range([0, width])
    .domain([
    d3.min(stagesOrd, function(s) { return s.values[0].week; }),
    d3.max(stagesOrd, function(s) { return s.values[ s.values.length - 1].week; })
    ]);
d3.max(stagesOrd, function(s) { return s.values[s.values.length - 1].week; })

yscaleOrd = d3.scale.linear()
    .range([height, 0])
    .domain([d3.min(stagesOrd, function(d) { return d.minOrd; })),
               d3.max(stagesOrd, function(d) { return d.maxOrd; })]

xAxisOrd = d3.svg.axis()
    .scale(xScaleOrd)
    .orient("bottom")
    .ticks(5);

yAxisOrd = d3.svg.axis()
    .scale(yScaleOrd)
    .orient("left")
    .ticks(5);

xScaleBud = d3.scale.linear()
    .range([0, width])
    .domain([d3.min(stagesBud, function(s) { return s.values[0].week; })),
               d3.max(stagesBud, function(s) { return s.values[s.values.length - 1].week; })]

yscaleBud = d3.scale.linear()
    .range([height, 0])
    .domain([d3.min(stagesBud, function(d) { return d.minBud; })),
               d3.max(stagesBud, function(d) { return d.maxBud; })]

xAxisBud = d3.svg.axis()
    .scale(xScaleBud)
.orient("bottom")
.ticks(5);

yAxisBud = d3.svg.axis()
  .scale(yScaleBud)
  .orient("left")
  .ticks(5);

// begin transitions for adding data

d3svgB.select("#xaxisInv")
  .transition()
  .delay(10)
  .duration(750)
  .call(xAxisInv);

d3svgB.select("#yaxisInv")
  .transition()
  .delay(10)
  .duration(750)
  .call(yAxisInv);

stagesInv.forEach(function(d) {
  d3svgB.select("#datalineInv" + d.key)
    .transition()
    .delay(10)
    .duration(750)
    .attr("d", lineInv(d.values))
});

d3svgB.select("#zeroline")
  .transition()
  .delay(10)
  .duration(750)
  .attr("y1", yScaleInv(0))
  .attr("y2", yScaleInv(0));

d3svgO.select("#xaxisOrd")
  .transition()
  .delay(10)
  .duration(750)
  .call(xAxisOrd);
```javascript
svgOrd.select("#yaxisOrd")
  .transition()
  .delay(10)
  .duration(750)
  .call(yAxisOrd);

stagesOrd.forEach(function(d) {
  svgOrd.select("#datalineOrd" + d.key)
    .transition()
    .delay(10)
    .duration(750)
    .attr("d", lineOrd(d.values))
});

svgOrd.select("#zeroline")
  .transition()
  .delay(10)
  .duration(750)
  .attr("y1", yScaleOrd(0))
  .attr("y2", yScaleOrd(0));

svgBud.select("#xaxisBud")
  .transition()
  .delay(10)
  .duration(750)
  .call(xAxisBud);

svgBud.select("#yaxisBud")
  .transition()
  .delay(10)
  .duration(750)
  .call(yAxisBud);

stagesBud.forEach(function(d) {
  svgBud.select("#datalineBud" + d.key)
    .transition()
    .delay(10)
    .duration(750)
    .attr("d", lineBud(d.values))
});

svgBud.select("#zeroline")
  .transition()
.delay(10)
.duration(750)
.attr("y1", yScaleBud(0))
.attr("y2", yScaleBud(0));
}

switchStage = function() {

  switch(($("input[name=stageRadio]:checked").val()){

    case "fc":
      $("#fcInput").show();
      $("#whInput").hide();
      $("#rwInput").hide();
      $("#rtInput").hide();
      break;

    case "rw":
      $("#rwInput").show();
      $("#fcInput").hide();
      $("#whInput").hide();
      $("#rtInput").hide();
      break;

    case "wh":
      $("#whInput").show();
      $("#fcInput").hide();
      $("#rwInput").hide();
      $("#rtInput").hide();
      break;

    case "rt":
      $("#rtInput").show();
      $("#fcInput").hide();
      $("#whInput").hide();
      $("#rwInput").hide();
      break;

    default:
      alert("Stage not found");
  }

}

toggleSeries = function() {

  if($("input[name=showChartCheckbox]").is(":checked")) {

    }
else {
    switch ($("input[name=stageRadio]:checked").val()){
    case "fc":
        $("#datalineInvFactory").show();
        $("#datalineInvWholesale").hide();
        $("#datalineInvWarehouse").hide();
        $("#datalineInvRetail").hide();
        $("#datalineOrdFactory").show();
        $("#datalineOrdWholesale").hide();
        $("#datalineOrdWarehouse").hide();
        $("#datalineOrdRetail").hide();
        $("#datalineBudFactory").show();
        $("#datalineBudWholesale").hide();
        $("#datalineBudWarehouse").hide();
        $("#datalineBudRetail").hide();
        break;
    case "rw":
        $("#datalineInvWarehouse").show();
        $("#datalineInvFactory").hide();
        $("#datalineInvWholesale").hide();
        $("#datalineInvRetail").hide();
        $("#datalineOrdFactory").hide();
        $("#datalineOrdWholesale").hide();
        $("#datalineOrdWarehouse").hide();
        $("#datalineOrdRetail").hide();
        $("#datalineBudFactory").hide();
        $("#datalineBudWholesale").hide();
        $("#datalineBudWarehouse").hide();
        $("#datalineBudRetail").hide();
        break;
    case "wh":
        break;
    default:
        break;
    }
}
```javascript
$("#datalineInvWholesale").show();
$("#datalineInvFactory").hide();
$("#datalineInvWarehouse").hide();
$("#datalineInvRetail").hide();
$("#datalineOrdWholesale").show();
$("#datalineOrdFactory").hide();
$("#datalineOrdWarehouse").hide();
$("#datalineOrdRetail").hide();
$("#datalineBudWholesale").show();
$("#datalineBudFactory").hide();
$("#datalineBudWarehouse").hide();
$("#datalineBudRetail").hide();
break;

    case "rt":
      $("#datalineInvRetail").show();
      $("#datalineInvFactory").hide();
      $("#datalineInvWholesale").hide();
      $("#datalineInvWarehouse").hide();
      $("#datalineOrdRetail").show();
      $("#datalineOrdFactory").hide();
      $("#datalineOrdWholesale").hide();
      $("#datalineOrdWarehouse").hide();
      $("#datalineBudRetail").show();
      $("#datalineBudFactory").hide();
      $("#datalineBudWholesale").hide();
      $("#datalineBudWarehouse").hide();
      break;
      default:
        alert("Stage not found");
    }
  }
};

$(document).ready(function(){
  // set up constants

  margin = {top: 10, right: 10, bottom: 10, left: 10},
padding = {top: 10, right: 70, bottom: 20, left: 20},
outer_width = 320,
outer_height = 150,
inner_width = outer_width - margin.left - margin.right,
inner_height = outer_height - margin.top - margin.bottom,
```
width = inner_width - padding.left - padding.right,
height = inner_height - padding.top - padding.bottom;

fc_cost_incoming_goods = 1,
RW_cost_incoming_goods = 4,
wh_cost_incoming_goods = 7,
rt_cost_incoming_goods = 10,
fc_value_outgoing_goods = 4,
rw_value_outgoing_goods = 7,
wh_value_outgoing_goods = 10,
rt_value_outgoing_goods = 13,
fc_cost_open_orders = 2,
RW_cost_open_orders = 2,
wh_cost_open_orders = 2,
rt_cost_open_orders = 2,
fc_cost_stock_holding = 1.5,
RW_cost_stock_holding = 1.5,
rt_cost_stock_holding = 1.5,
wh_cost_stock_holding = 1.5;

var color = d3.scale.category10();

sortOrder = ["Retail", "Wholesale", "Warehouse", "Factory");

// initial data set (week 0)

dataInv = [
    {stage: "Factory", week: 0, inventory: 6 },
    {stage: "Warehouse", week: 0, inventory: 6 },
    {stage: "Wholesale", week: 0, inventory: 6 },
    {stage: "Retail", week: 0, inventory: 6 },
];

dataOrd = [
    {stage: "Factory", week: 0, orders: 6 },
    {stage: "Warehouse", week: 0, orders: 6 },
    {stage: "Wholesale", week: 0, orders: 6 },
    {stage: "Retail", week: 0, orders: 6 },
];
dataBud = [
    {stage: "Factory", week: 0, budget: 6 },
    {stage: "Warehouse", week: 0, budget: 6 },
    {stage: "Wholesale", week: 0, budget: 6 },
];
```javascript
var stagesInv = d3.nest()
  .key(function(d) { return d.stage; })
  .sortKeys(function(a,b) {
    return sortOrder.indexOf(a) - sortOrder.indexOf(b);
  })
  .entries(dataInv);

stagesInv.forEach(function(s) {
  s.maxInv = Math.max(0, d3.max(s.values, function(d) {
    return d.inventory; }));
  s.minInv = Math.min(0, d3.min(s.values, function(d) {
    return d.inventory; }));
});

xScaleInv = d3.scale.linear()
  .range([0, width])
  .domain([
    d3.min(stagesInv, function(s) { return s.values[0].week; }),
    d3.max(stagesInv, function(s) { return s.values[s.values.length - 1].week; })
  ]);}

yScaleInv = d3.scale.linear()
  .range([height, 0])
  .domain([
    d3.min(stagesInv, function(d) { return d.minInv; }),
    d3.max(stagesInv, function(d) { return d.maxInv; })
  ]);}

xAxisInv = d3.svg.axis()
  .scale(xScaleInv)
  .orient("bottom")
  .ticks(5);

yAxisInv = d3.svg.axis()
  .scale(yScaleInv)
```
```javascript
    .orient("left")
    .ticks(5);

    lineInv = d3.svg.line()
        .x(function(d) { return xScaleInv(d.week); })
        .y(function(d) { return yScaleInv(d.inventory); })
        .interpolate("linear");

    svgInv = d3.select(document.getElementById("inventoryGraph"))
        .data(stagesInv)
        .attr("width", outer_width)
        .attr("height", outer_height)
        .append("g")
        .attr("transform", "translate(" +
            padding.left+margin.left + "," +
            padding.top+margin.top + ")"));

    svgInv.append("clipPath")
        .attr("id", "chartAreaInv")
        .append("rect")
        .attr("x", 0)
        .attr("y", 0)
        .attr("width", width)
        .attr("height", height);

    stagesInv.forEach(function(d,i) {

        svgInv.append("path")
            .attr("class", "dataline")
            .attr("id", "datalineInv" + d.key)
            .attr("d", lineInv(d.values))
            .attr("clip-path", "url(#chartAreaInv)")
            .style("stroke", function() {
                return d.color = color(d.key); })

        svgInv.append("text")
            .attr("x", width + margin.right)
            .attr("y", padding.top + i*legendspace)
            .attr("class", "legend")
            .style("fill", function() {
                return d.color = color(d.key); })
            .text(d.key);
```

var stagesOrd = d3.nest()
  .key(function(d) { return d.stage; })
  .sortKeys(function(a,b) {
    return sortOrder.indexOf(a) - sortOrder.indexOf(b);
  })
  .entries(dataOrd);

stagesOrd.forEach(function(s) {
s.maxOrd = Math.max(0, d3.max(s.values, function(d) {
    return d.orders; }}));

s.minOrd = Math.min(0, d3.min(s.values, function(d) {
    return d.orders; }}));

xScaleOrd = d3.scale.linear()
    .range([0, width])
    .domain([
        d3.min(stagesOrd, function(s) { return s.values[0].week; }),
        d3.max(stagesOrd, function(s) { return s.values[ s.values.length - 1].week; })
    ]);}

yScaleOrd = d3.scale.linear()
    .range([height, 0])
    .domain([]}
        d3.min(stagesOrd, function(d) { return d.minOrd; }),
        d3.max(stagesOrd, function(d) { return d.maxOrd; })
    ]);}

xAxisOrd = d3.svg.axis()
    .scale(xScaleOrd)
    .orient("bottom")
    .ticks(5);

yAxisOrd = d3.svg.axis()
    .scale(yScaleOrd)
    .orient("left")
    .ticks(5);

lineOrd = d3.svg.line()
    .x(function(d) { return xScaleOrd(d.week); })
    .y(function(d) { return yScaleOrd(d.orders); })
    .interpolate("linear");

svgOrd = d3.select(document.getElementById("ordersGraph"))
    .data(stagesOrd)
    .attr("width", outer_width)
.attr("height", outer_height)
.append("g")
  .attr("transform", "translate(0)"
       padding.left+margin.left) + "," +
       padding.top+margin.top) + ")");

svgOrd.append("clipPath")
 .attr("id", "chartAreaOrd")
 .append("rect")
 .attr("x", 0)
 .attr("y", 0)
 .attr("width", width)
 .attr("height", height);

stagesOrd.forEach(function(d,i) {

  svgOrd.append("path")
   .attr("class", "dataline")
   .attr("id", "datalineOrd" + d.key)
   .attr("d", lineOrd(d.values))
   .attr("clip-path", "url(#chartAreaOrd)")
   .style("stroke", function() {
       return d.color = color(d.key); })

  svgOrd.append("text")
   .attr("x", width + margin.right)
   .attr("y", padding.top + i*legendSpace)
   .attr("class", "legend")
   .style("fill", function() {
       return d.color = color(d.key); })
   .text(d.key);
});

svgOrd.append("g")
 .attr("class", "axis")
 .attr("id", "xaxisOrd")
 .attr("transform", "translate(0," + height + ")")
 .call(xAxisOrd)
 .append("text")
   .attr("class", "label")
   .attr("id", "xaxistext")
   .attr("x", width)
   .attr("y", -6)
.text("Week");

svgOrd.append("g")
  .attr("class", "axis")
  .attr("id", "yaxisOrd")
  .call(yAxisOrd)
  .append("text")
    .attr("class", "label")
    .attr("id", "yaxistext")
    .attr("transform", "rotate(-90)")
    .attr("y", 14)
    .attr("x", 0)
    .text("Orders");

svgOrd.append("g")
  .append("line")
    .attr("id", "zeroline")
    .attr("y1", yScaleOrd(0))
    .attr("y2", yScaleOrd(0))
    .attr("x1", 0)
    .attr("x2", width);

var stagesBud = d3.nest()
  .key(function(d) { return d.stage; })
  .sortKeys(function(a,b) {
    return sortOrder.indexOf(a) - sortOrder.indexOf(b);
  })
  .entries(dataBud);

stagesBud.forEach(function(s) {
  s.maxBud = Math.max(0, d3.max(s.values, function(d) {
    return d.budget; }));
  s.minBud = Math.min(0, d3.min(s.values, function(d) {
    return d.budget; }));
});

xScaleBud = d3.scale.linear()
  .range([0, width])
  .domain([
    d3.min(stagesBud, function(s) { return s.values[0].week; }),
    d3.max(stagesBud, function(s) { return s.values[}]

134
```javascript
    s.values.length - 1].week; })
  ];

  yScaleBud = d3.scale.linear()
    .range([height, 0])
    .domain([d3.min(stagesBud, function(d) { return d.minBud; }),
             d3.max(stagesBud, function(d) { return d.maxBud; })])

  xAxisBud = d3.svg.axis()
    .scale(xScaleBud)
    .orient("bottom")
    .ticks(5);

  yAxisBud = d3.svg.axis()
    .scale(yScaleBud)
    .orient("left")
    .ticks(5);

  lineBud = d3.svg.line()
    .x(function(d) { return xScaleBud(d.week); })
    .y(function(d) { return yScaleBud(d.budget); })
    .interpolate("linear");

  svgBud = d3.select(document.getElementById("budgetGraph"))
    .data(stagesBud)
    .attr("width", outer_width)
    .attr("height", outer_height)
    .append("g")
    .attr("transform", "translate(" +
                    padding.left+margin.left + "," +
                    padding.top+margin.top + ")");

  svgBud.append("clipPath")
    .attr("id", "chartAreaBud")
    .append("rect")
    .attr("x", 0)
    .attr("y", 0)
    .attr("width", width)
  ```
.attr("height", height);

stagesBud.forEach(function(d,i) {

    svgBud.append("path")
        .attr("class", "dataline")
        .attr("id", "datalineBud" + d.key)
        .attr("d", lineBud(d.values))
        .attr("clip-path", "url(#chartAreaBud)")
        .style("stroke", function() {
            return d.color = color(d.key); })

    svgBud.append("text")
        .attr("x", width + margin.right)
        .attr("y", padding.top + i*legendSpace)
        .attr("class", "legend")
        .style("fill", function() {
            return d.color = color(d.key); })
        .text(d.key);

});

svgBud.append("g")
    .attr("class", "axis")
    .attr("id", "xaxisBud")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxisBud)
    .append("text")
        .attr("class", "label")
        .attr("id", "xaxistext")
        .attr("x", width)
        .attr("y", -6)
        .text("Week");

svgBud.append("g")
    .attr("class", "axis")
    .attr("id", "yaxisBud")
    .call(yAxisBud)
    .append("text")
        .attr("class", "label")
        .attr("id", "yaxistext")
        .attr("transform", "rotate(-90)")
        .attr("y", 14)
        .attr("x", 0)
.text("Costs (")

svgBud.append("g")
   .append("line")
   .attr("id", "zeroline")
   .attr("y1", yScaleBud(0))
   .attr("y2", yScaleBud(0))
   .attr("x1", 0)
   .attr("x2", width);

// other setup tasks: initializing and hiding/showing inputs and chart series

switchStage();
toggleSeries();
$("#orderButton").show();

// change event toggles

$("input[name=stageRadio]").on("change", function(){
    switchStage();
toggleSeries();
$("#orderButton").show();
});

$("input[name=showChartCheckbox]").on("change", function()
    toggleSeries();
});

});

});

//