**Project Purpose:** Manufacturing processes in the 21st century become increasingly more automated every day. Industrial processes as diverse as carpentry, metal engraving, glass scoring, and plasma arc welding—formerly the domain of hundreds of artisans or specialized assembly line workers—can now be directed by a single designer. Two pieces of software, CAD (Computer Aided Design) and CNC (Computer Numerical Control) make this automation possible. CAD is used to design a manufacturing process and its output product, while CNC directs the machinery that realized the CAD design. Unfortunately, both of these pieces of software, and thus the machines and physical products that they control, are often vulnerable to industrial sabotage and cyberterrorism. Sadly, many corporate leaders are oblivious or ill-informed as to these potential threat vectors. My goal for this semester was simple, though my methods were not: to develop a versatile tool with which corporations could easily and accurately model their Industrial Production Environments. The tool that I have created utilizes a built-in interpreter to identify vulnerabilities in the production chains it is given.

**Overview of Methodology:** For my semester MIC project, I worked with Professor Jules White, who has received a grant to investigate and strengthen cybersecurity in industrial environments. As I am not a Mechanical Engineering student, my knowledge of manufacturing processes and Material Science at the beginning of the class was admittedly limited. I read and reviewed two manufacturing guides, Chris Lefteri’s *Making It: Manufacturing Techniques for Product Design* and Rob Thompson’s *Manufacturing Processes For Design Professionals*, in order to be able to accurately model different manufacturing concepts. For my second step, I created a metamodel (described in detail below) in the MetaGME environment of the key concepts that I learned about in the manuals. Next, I worked with Professor White to create several sample models based on my metamodel, in order to identify any necessary additions that I needed to make to the metamodel to provide complete functionality. Finally, I wrote a model interpreter in C# that takes an industrial process model and generates a syntactically valid HTML5 page describing the model and its vulnerabilities.

**Metamodel Concepts:** The **Process** is the core concept around which my models revolve. There are a massive number of possible industrial processes. Inspired by Thompson’s manufacturing manual, which separates the processes it describes into categories, I divided up the set of all possible industrial processes into five subsets—**Forming Processes** (such as Casting or Injection Moulding), **Cutting Processes** (such as Electron Beam Machining), **Joining Processes** (such as Riveting or Brazing), **Finishing Processes** (such as Belt Sanding), and **Disposal Processes** (such as Trash Hauling or Recycling Sorting). These processes inherit from an abstract base model, Industrial\_Process, and can be dropped right into my main model, **Industrial\_Model**, intended to represent a “Factory Floor” of sorts containing many logistical operations and their related assets. My metamodel contains several other categories of First Class Object which can be placed into an Industrial\_Model, namely **Products** and **Assets.** There are four types of product—**Raw, Intermediate, Final,** and **Waste**—and four asset types—**Physical, Human, Software**, and **Sandbox**. Processes are linked to Products (which they either consume or produce) by InputProduct and OutputProduct ports. The relationship is not bi-directional, and thus a product and its role must be identified as either an input or an output within both the product as well as in the process that it is linked to. Processes also contain (or share) assets via a similar linkage.

**Modeling Vulnerabilities in Products**: In addition to serving as an input and/or output to a process, Products also contain internal data to model potential threat vectors to their composition. Each product may have a associated **Product\_Topology**, a model that describes the product. If a topology is not provided, then it is assumed that the product in question is invulnerable to tampering, even if associated processes are compromised. Within the Product\_Topology model, an unlimited number of **Vulnerability\_Facet** models can be added in. I give the example of a 3D printed cube. Its depth, scaling, hollow spaces, and raft can be altered, potentially compromising the cube in different ways. I allow the modeler to add Worst\_Case\_Scenario models inside of each Vulnerability\_Facet model; these are to be used by the modeler to indicate what could possibly go wrong if the specified Vulnerability\_Facet is compromised to the maximum extent.

**Interpreter:** I had initially planned to provide five functions in my C# Interpreter: Hardware Validation, Software Validation, Human Resources Validation, Thematic Analysis, and Cyberterror Vulnerability. While all of these validation steps would have been of some benefit to a corporate partner, Professor White encouraged me to focus on only the last component, Cyberterror Vulnerability. My interpreter reads in the root folder of a given model. Then, for each model contained in the root folder—that is, for each “Factory Floor” that the customer has modeled, a different section is auto-generated in an HTML file. Now, for each “Factory Floor,” its elements will be separated out by type. Software and Sandbox Asset Models will be processed in the first stage, all other models in the second stage. In the first stage, the version names of all Software and Sandbox Assets are collated. This information is stored in an attribute within those two models. Those names (which must be entered exactly as they appear in that software’s official description) will be checked against the National Institute of Standards and Technology Vulnerability Database, which should be included in the model directory as a .xml file. This adds a lot of flexibility for corporate developers, as they can substitute their own xml files for the one taken from NIST if they choose simply by changing one line of code in the interpreter. If any software assets are found to be unsafe, then all processes connected to that software, as well as all products output by processes connected to that software, are declared vulnerable. As the last step in the first stage of interpretation, all software and sandbox assets are output to the HTML file. Vulnerable assets appear in red boxes, while safe assets appear in green boxes. Lastly, in the second stage of model interpretation, all other nodes are output to the HTML file. If any of those nodes were marked vulnerable in the first stage of interpretation, they are printed in red with their worst-case-scenarios identified. Otherwise, they are printed in green. Finally, as a vestigial remnant of my employee validation idea, all employees with the title of Manager are marked safe and all others are highlighted in orange as possible sabotage threats. In a future version of this stage, I plan on identifying all workers with solo, unsupervised control over any one process as potential domestic terror threats—the idea being that managers would have passed more extensive background checks and should supervise less-trustworthy employees.

**Conclusion:** I believe that my interpreter and modeling platform have met or exceeded my mentor’s expectations, and it was a privilege getting to work on such an important project. I would like to thank Professor White for suggesting this topic to me, as well as for his advice throughout the process . Thank you also to Professor Sztipanovits for an excellent thematic overview of modeling languages in the MIC class, and to Patrik and István for their instruction in MetaGME.

**Works Cited:**

Lefteri, Chris. *Making It: Manufacturing Techniques for Product Design*, 2nd Edition. 2012. London:

Laurence King Publishing, 2013.

Thompson, Rob. *Manufacturing Processes For Design Professionals.* 2007. New York: Thames and

Hudson, 2014.