Toolpath Static Analysis Testing for Additive Manufacturing

By

Kyle Searle

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Computer Science and Software Engineering

Auburn, Alabama
August 8, 2020

Keywords: Additive Manufacturing, toolpath,
3D printing, static analysis, python

Approved by

David Umphress, Chair, Professor of CSSE
Mark Yampolskiy, Associate professor of CSSE
Drew Springall, Assistant professor of CSSE

ABSTRACT

Fused Deposition Modelling (FDM) 3D printers are the most common 3D printers in the world. However, most of the world's 3D printers have little-to-no security to prevent cyber-attacks. Most small businesses that employ 3D printing technology do so with minimal security. A possible approach to assuring 3D product and printer quality is static analysis testing. In reverse engineering, static analysis observes a file's inner workings before execution to determine if malicious elements are present. This thesis investigates and proposes how static analysis testing can be applied to Additive Manufacturing (AM). This process executes a series of test cases on values derived from the toolpath file before the physical printing process takes place to ensure that the data will not negatively affect the printer. With this approach, defects can be detected at the 3D model, stereolithography (STL), and toolpath levels of the 3D printing process on G-code based and non-G-code based toolpath files. This research effort demonstrates test cases that can prevent attacks on five printing attributes (including extruder temperature, heat bed temperature, infill, layer thickness, and fan speed). An attack on one or more of these attributes can cause the part's mechanical properties to change or damage the printer itself. These test cases determine appropriate slicer settings, and commands are within the printer's capabilities. This study demonstrates that a toolpath testing driven approach can effectively defend against cyber-physical attacks on 3D printers and the 3D printing process.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. CHAPTER-1: PROBLEM DESCRIPTION

Naturally, small businesses have minimal resources and an inherently large amount of risk. In combination with high overhead, such as a multimillion-dollar metal 3D printer, recovery from a significant incident becomes significantly less likely. To prevent such an incident, manufacturers must enact a level of security that reduces unnecessary risk.

In 2017, 31% of the U.S. economy was comprised of manufacturing, generating $6.0T in gross output. Medium-to-small enterprises comprise 98% of the manufacturing sector, with small enterprises representing 73% of the population. Here, medium enterprises have less than 500 employees, and small enterprises have 20 or fewer employees [9].

Most residential 3D printers use Fused Deposition Modelling (FDM) technology, meaning they rely on moving mechanical elements to lay down successive layers of heated polymer to form a 3D object [8].

Their appeal to end-users and small businesses is in their ability to quickly and cheaply produce physical models (also known as “rapid prototyping”) [6]. While FDM printers are appealing because of their affordability, residential and small businesses implementing 3D printing technology do not usually have enough capital to prioritize cyber-security, thus, leaving systems vulnerable to compromise and subsequent manipulations. Traditional manufacturing quality assurance (QA) equipment designed to test the print in post-production is available but not ordinarily affordable by most FDM printer owners. Further, not only is this QA equipment expensive, but organizations such as NIST [18] and NASA [28] have called for new QA techniques

to be developed because they cannot reliably detect defects in AM parts. This lack of security poses a significant problem since most 3D printers are residentially owned. This issue can lead to technical data theft (also known as intellectual property, IP theft), sabotage attacks (that can target the manufactured part, 3D printer, or environment), and illegal part manufacturing [32].

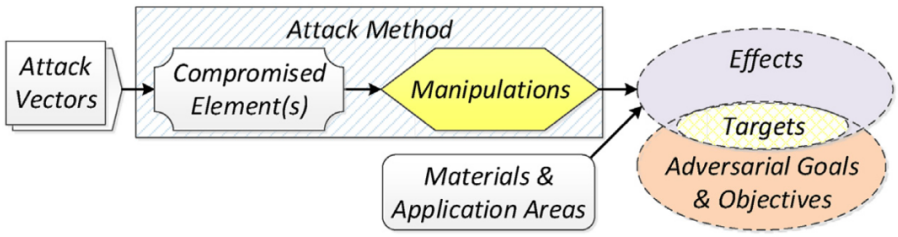

Figure 1: AM attack analysis framework [32]

3D printers can become compromised at various points in the 3D printing process. In Figure 1, "attack methods" are semantically identical manipulations introduced by different compromised elements [32]. Meaning, multiple attacks can have similar effects despite having different compromised elements. For instance, one attack method can compromise slicing software and sabotage the 3D printer by generating a toolpath file with inappropriate printer settings. Another attack method can intercept a file in transit before it reaches the printer, make malicious modifications, then send it to the printer. Another type of attack method can use reverse shells to exchange benign stereolithography (STL) files with malicious ones on the victim's host computer. Additionally, some attack methods can compromise 3D printer firmware by modifying the printing parameters such that they are beyond the printer's capabilities [33]. These attack methods can all result in similar effects but occur at different points of the 3D printing process. This list is not comprehensive; several other consequential types of attacks exist, such as those aimed at stealing intellectual property. However, this paper seeks to address sabotage attacks against AM equipment.

Several categories of actors carry out attacks against AM. Table 1 lists six actors and the areas they impact on conventional manufacturing. However, since this paper focuses on physical systems, we exclude financial theft/fraud, theft of IP or strategic plans, and regulatory. These areas are still important AM aspects but are beyond the scope of this paper.

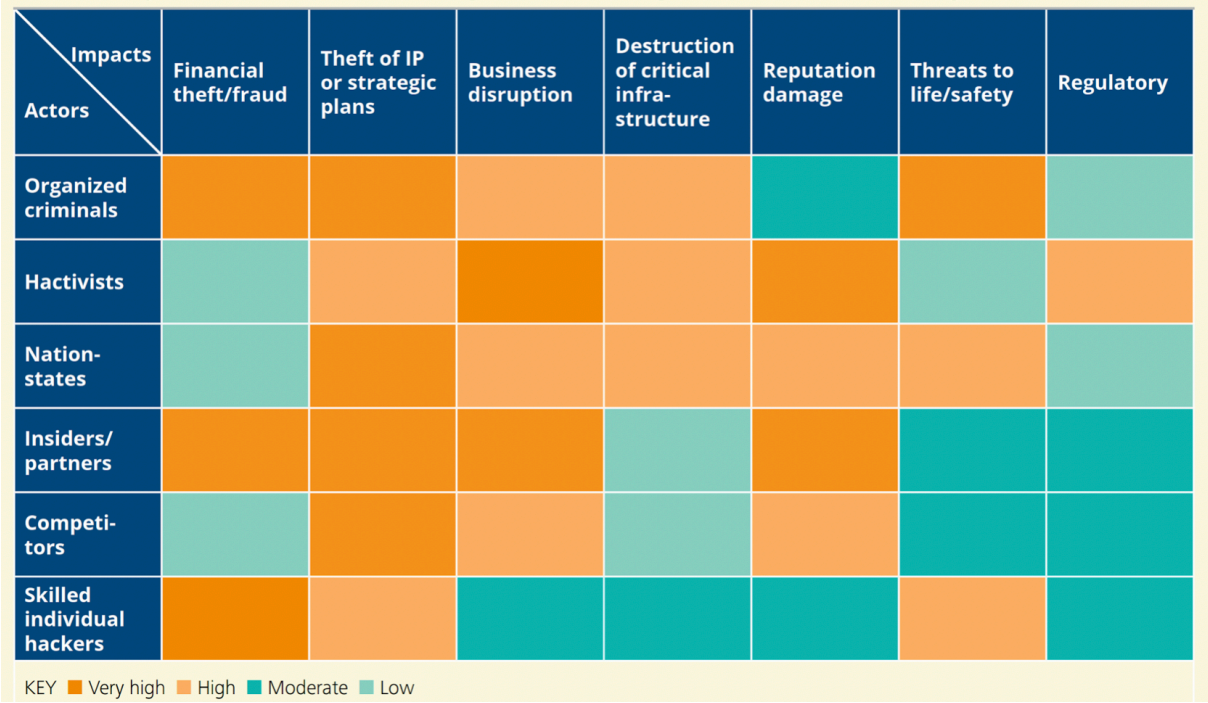

Table 1: Cyber threat actors and impacts: Heat map for the manufacturing sector [35]

3D printer sabotage affects both those inside and outside the AM organization. Additive manufacturing is composed of a diverse group of stakeholders. Owners, managers, purchasers of AM systems, AM print material suppliers, designers, vendors, purchasers, maintainers, and AM product disposers can compose just one AM organization [35].

While still considered an emerging technology, there are advantages to adopting AM over traditional (subtractive) manufacturing. For example, AM can reduce the number of assembly parts of a component. GE Aviation, for instance, replaced its traditional twenty-piece fuel injection nozzles to a single AM piece for planes like the Boeing 777X [36].

Technological progress, made possible only through AM, has made an impact across numerous industries. The automotive industry uses AM to increase prototyping and rapid product development [15]. In the medical industry, 3D printed surgical cutting and drill guides are used to aid in surgery. Additionally, AM orthopedic implants are used to lower medical costs [17]. AM molds for Invisalign orthodontic aligners in the dental industry and AM metal copings for crowns and bridges have reduced customer costs [11]. One of the early adopters of AM, the aerospace sector, has printed over 200 unique parts on 16 different commercial and military aircraft [30]. However, as AM's popularity rises, so does the accompanying security concerns. Therefore, we must prioritize 3D printer security so that technological progress is not slowed or halted.

Graves analyzes AM security from several perspectives of security awareness [10]. Here, Graves explains why we cannot apply the traditional manufacturing (Subtractive Manufacturing, SM) cyber-security approaches to AM. From an exposure to attack awareness perspective, subtle distinctions originate from the differences in the manufacturing environments. Among the environmental factors, AM has a higher exposure to potentially malicious actors, increased potential for fraudulent behavior by participants actors, and less challenging theft due to concentrated technical data. Further, substantial differences exist between AM and SM in manipulations of the attack analysis framework (see Figure 2). For example, in sabotage attacks, numerous manipulations can introduce defects unique to an additively manufactured part that could result in sabotage. While in SM, defects that reduce tensile strength or other parameters are limited to external defects for which detection with non-destructive evaluation (NDE) methods is rather trivial. While there is a significant overlap between AM and SM security, numerous aspects of AM are unique and require special considerations.

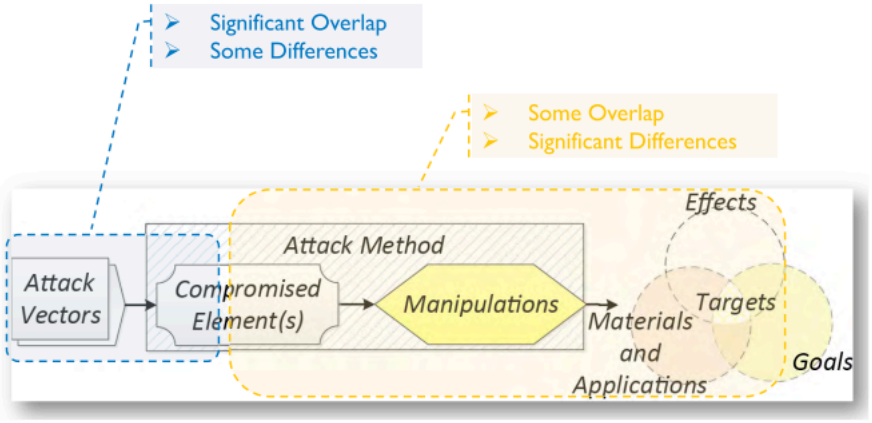

Figure 2: Overlap and differences between AM and SM securities [10]

This thesis proposes a static analysis process to develop test cases on toolpath files used with Fused Deposition Modelling 3D printers to enact a level of security that reduces unnecessary risk for FDM 3D printers. Static analysis, more commonly known in reverse engineering, observes a file's inner workings before execution to determine if malicious elements are present [22]. Thus, providing a level of certainty that the file will not sabotage a 3D printer.

Here, we create the toolpath static analysis testing (TSAT) process for Additive Manufacturing. This process executes a series of test cases before the physical printing process to ensure that the print will not negatively affect the printer. TSAT can be used to check for errors made by compromised files and software.

The goals of this thesis were to:

- Determine and create test cases that ensure the mitigation of an acceptable level of risk.
- Detect malicious defects made in the 3D printing process before printing and TSAT.
- Demonstrate that TSAT can be implemented on more than one type of toolpath file.
- Create testing environments that are easier to use than not use.

2. CHAPTER-2: PREVIOUS WORK

2.1 ADDITIVE MANUFACTURING PROCESSES

The American Society for Testing and Materials (ASTM) divides AM technology into seven distinct process categories [6] :

- Binder Jetting
- Direct Energy Deposition
- Material Extrusion
- Material Jetting
- Powder Bed Fusion
- Sheet Lamination

The most common 3D printing process is Materials Extrusion. Each AM process has several refinements, sometimes referred to as AM technologies. FDM is a technology that implements the Materials Extrusion process. In the FDM process, polymeric material (usually plastic) is fed into a heating chamber and liquified. The filament is then extruded through a small standardized nozzle (usually 0.4 and 0.35 mm) and deposited onto the print bed. Layer by layer, the printer creates a 3D object by fusing each layer to the previous layer [16].

FDM has no role in AM with metals. Instead, the predominating AM processes for commercial use are Powder Bed Fusion (PBF) and Direct Energy Deposition (DED) [32].

2.2 3D PRINTING PROCESS

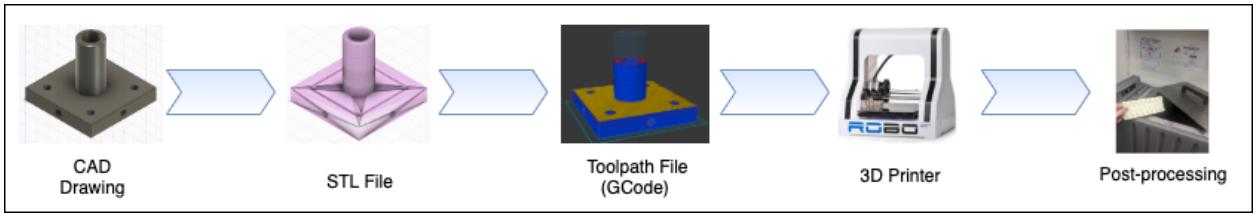

Figure 3: Additive Manufacturing process

Figure 3 presents the AM printing process, which consists of five steps to printing a part. First, we create a 3D model with Computer-Aided-Design (CAD) software. Then we convert our model to a Stereolithography (STL) file. Throughout the AM industry, we use STL files to capture all our parts' geometry.

Next, we use slicing software to divide our STL file into layers for our 3D printer to print. The compilation of these layers makes up the toolpath file (in this case, the G-code file). G-code is the list of instructions that our printer reads so that it can set the appropriate settings. Additionally, the G-code contains all the X-Y-Z axis movements the printer must make to print each layer to make our part. Finally, after the printer prints all the layers, we have a physical component. It is here we do all the post-production finishing touches.

2.3 3D PRINTERS AS A WEAPON

In Figure 4, Yampolskiy outlines how attacks on or with 3D printers can be performed [33]. For example, Yampolskiy's framework can be applied to the dr0wned study to sabotage 3D printed drone propeller blades [5]. In this case, the phishing email is the "attack vector," the controller PC is the "compromised element," the replaced STL file is the "manipulation," and the propeller breaking in mid-flight is the "effect." The resulting "goal" is the destruction of the drone using a "3D printer as a weapon" (3D-PaaW).

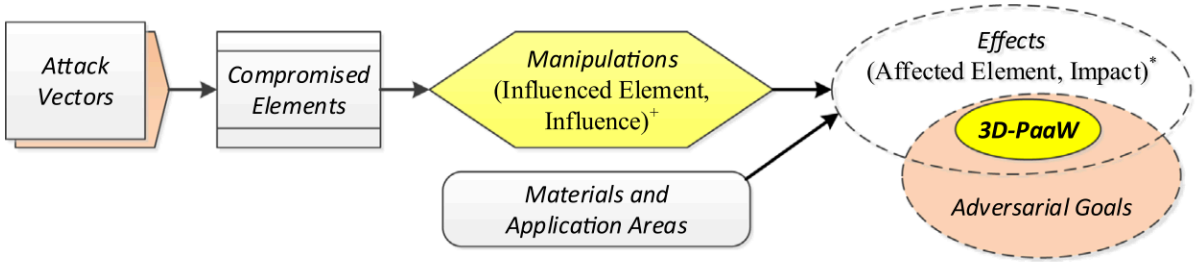

Figure 4: Attacks on/with 3D printers [33]

2.4 PREVIOUS WORKS ON DETECTING SABOTAGE ATTACKS

In response to NIST [18] and NASA's [28] call for new non-destructive evaluation (NDE) methods, several publications on the defense of AM sabotage attacks were published. Yampolskiy compiled a comprehensive and structural survey detailing these publications [32]. He notes three attack categories that are relevant to this thesis: 3D geometry, fill pattern/material, and overstressed areas.

2.4.1 3D GEOMETRY

In 3D geometry, Belikovetsky studies acoustic emanations on the printer and object specifications. Belikovetsky proceeds by carrying out Principal Component Analysis (PCA) [31] to observe attacks on the printer to create a signature with the audio of the test print. A comparison between the test and control print signatures determine if derivations are present in the G-code of the files [4].

Chhetri makes similar studies on impedance-based 3D geometry for object specification attacks. However, rather than a signature, Chhetri uses the printer's audio to reconstruct a model of the part. Then, when compared, derivations between the generated and original models indicate an attack [7].

Albakri also studies impedance-based 3D geometry with Independence-based Structural Health Monitoring (SHM). Here, Albakri implements SHM to detect dimensional and positional defects with a piezoelectric sensor attached to the printer. Further study is needed, but it might also be possible to detect internal porosity with this method [2].

Strum follows suit in studying impedance-based 3D geometry and building on the work of Albakri. Strum goes one step further by using the same piezoelectric sensors to inspect various layer intervals. A control print is used to create a baseline signature. Afterward, prints can be tested against the baseline signature to identify malicious parts. One advantage of Strum's method is early detection. Defects are identified during the printing process, rather than testing the completed part [24].

Vincent's approach is a quality control (QC) system with impedance-based verification of dimensional, positional, mass, and accuracy. He proceeds by attaching piezoelectric transducers (PZT) to a printed component. These sensors measure impedance and reveal deviations from the component's original design. The basis for Vincent's approach is that sophisticated attackers typically avoid a print's key quality characteristics (KQC). KQC being what traditional QC methods observe to detect part defects [27].

2.4.2 FILL PATTERN/MATERIAL

Bayens focuses on spatial verification in his work on fill patterns. Building on the work of Belikovetsky [4] and Chhetri [7], Bayens uses spatial verification to visualize nozzle movements instead of the sent G-code commands from the 3D printing software to detect compromised controller PCs or printer firmware. Additionally, to address Yampolskiy's [33] filament substitute concerns in Belikovetsky and Chhetri's works, Bayens verifies materials using Raman spectroscopy [3].

2.4.3 OVERSTRESSED AREAS

Tsoursos' work emphasizes structural integrity defects introduced during the 3D printing process. The first step of the two-step process is to use the toolpath (G-code) file to create an approximate model of the original part. The second step is to use Finite Element Analysis (FEA) [25] to observe the model under various stress conditions [26].

2.5 PREVIOUS COMMERCIAL SECURITY MEASURES

The US Airforce [34] and General Electric [12] are using blockchain technologies to create immutable and non-reputable digital ledgers for their complex supply chains. Companies, such as Identify3D, implement blockchain technology to provide industrial solutions to additive and subtractive manufacturers.

Identify3D protect their clients' data by creating a blockchain encrypted [1] secure container to host all their design and manufacturing files, which prevent the introduction of defects. Clients digitally sign these files to protect the files' integrity. The company then creates and associates a set of licenses within the secure container. These licenses define business and manufacturing rules that dictate how the files can be used. Contractual licenses assign business rules within designs, such as quantities, time, and authorized manufactures. Production licenses assign manufacturing rules, such as which machines are authorized to be used for the manufacturing process, build parameters, and material choice [37]. Once the licenses are generated, the client transfers both the container and license files to the appropriate machine via network or any other storage media that supports Identify3D technology. Upon license validation, the machine decrypts and authorizes the files for production [38].

While Identify3D offers security measures that prevent the introduction of defects, they still lack any form of analysis that can identify defects if they are already present. Therefore, adversaries injecting malicious commands in the generated toolpaths are still possible.

2.6 SUMMARY

Many previous works prevent AM sabotage using models or signatures generated from microphones, piezoelectric sensors, or cameras. However, these methods assume that identical parts are printed numerous times, which is not usually the case with residential FDM printers. Previous commercial security measures implement blockchain technology. While this technology is an effective method of preventing the introduction of defects, it has no means of determining if any defects exist in the toolpath files.

Attacks on AM can take place throughout the 3D printing process. Before printing, attacks on the 3D printing process take place in the toolpath file. Here, the toolpath file contains all the 3D printer operations, settings, and defects. Yampolskiy outlines how to perform attacks on or with 3D printers.

Attack vectors compromise elements that result in manipulations. These manipulations, with materials and application areas, produce effects on AM. Here, using 3D printers as a weapon can produce effects that may achieve an adversary's goals.

3. CHAPTER-3: SOLUTION

We developed TSAT to mitigate AM cybersecurity vulnerabilities by adapting Yampolskiy's attack on or with AM framework [33] to develop test cases from toolpath files that prevent AM sabotage. Our approach, TSAT, was developed with the following objectives in mind:

1. Determine and create test cases that ensure the mitigation of an acceptable level of risk.
2. Detect malicious defects made in the 3D printing process before printing and TSAT.
3. Demonstrate that TSAT can be implemented on more than one type of toolpath file.
4. Create testing environments that are easier to use than not use.

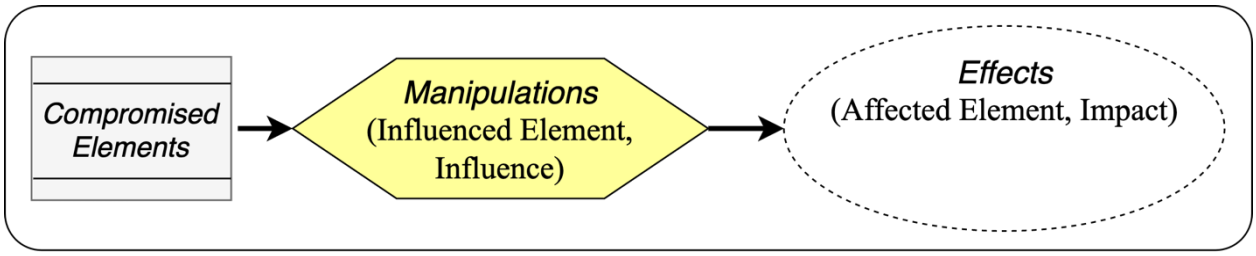

Figure 5: Adapted attack on/with AM framework (based on [33])

TSAT is not a software program; rather, it is a process to develop test cases that prevent AM sabotage. The process adapts Yampolskiy's [33] framework to emulate attacks on or with 3D printers to develop our test cases. Figure 5 demonstrates how we use the framework to develop test cases. First, we assume that elements of the 3D printing process have already been compromised. Next, we produce defects on behalf of the compromised elements (manipulations). Then, we observe the effects on the toolpath file and write test cases that detect the manipulations.

3.1 TSAT PROCESS

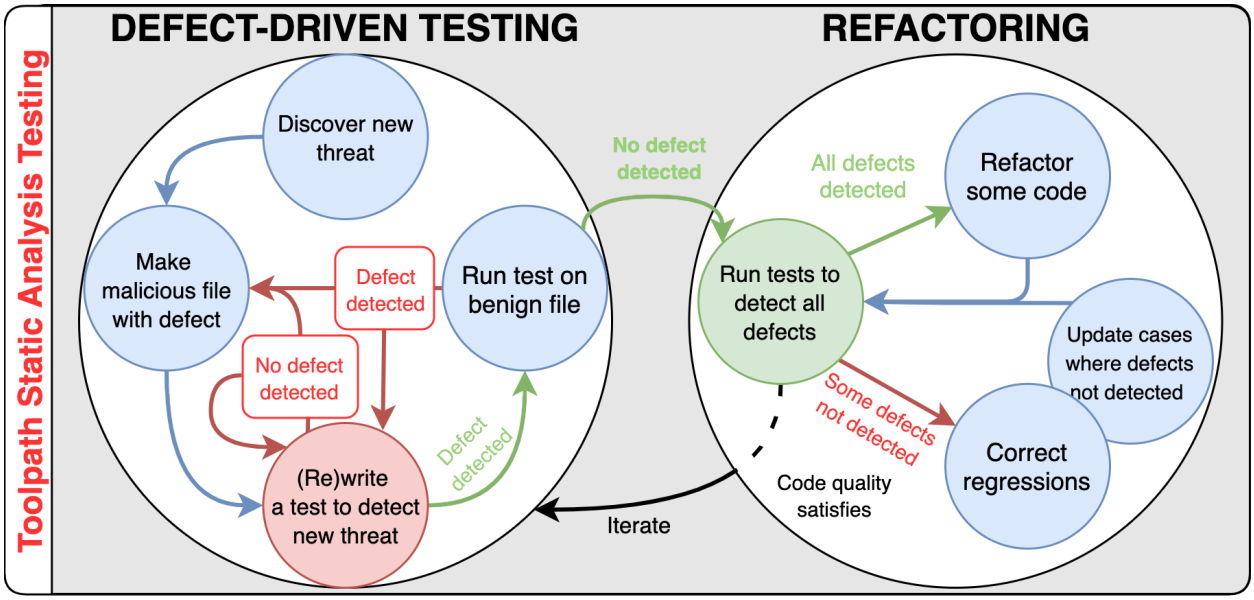

Figure 6: TSAT process

In Figure 6, the TSAT development cycle starts with the "defect-driven testing" phase. In this phase, we discover a new AM threat and recreate that threat in a toolpath file. Here, attacks can be made at any point in the 3D printing process as long as the toolpath file contains the desired defect. Then, we write and execute a test case that detects the malicious defect and observe the results ("red light" test). This stage is known as the "red light" stage of the TSAT process because we must keep returning to this step until we can detect the defect. If the defect is not recognized, we failed our objective and must determine if our test code or our malicious toolpath file is the cause. If our defect is detected, we achieved our objective and can move onto the next step. Next, we execute our test case on a benign toolpath file to demonstrate that our test case passes when the defect is not present in the file. Otherwise, we know our test case does not extract the correct values from the toolpath file and must repeat the previous steps.

We move from the defect-driven testing phase of the TSAT process to the refactoring phase. In this phase, we execute all our test cases on the malicious toolpath file. It is not likely

that all the test cases will detect defects from one toolpath file. For example, say, we implemented a test case that determines if the printer fan has started, and another test case that determines that the fan setting is below the fan's physical limits. It would be impossible to set the fan setting over the fan's physical limits without making a line that sets the fan setting. Therefore, one of these test cases would never detect a defect. Multiple malicious toolpath files are required to check if all the tests identify defects. When running the files through the TSAT program, we focus on two things: all test cases detect a defect at least once, and no test case results in an error.

If a previous test case is no longer identifying a defect, we need to confirm that our malicious toolpath file contains the appropriate defect. Otherwise, we need to update the test cases. We then run all our test cases again and continue this cycle until all test cases detect their corresponding defects.

Once we demonstrate that all the test cases can identify defects ("green light" test), we can start refactoring to improve our test cases. Once again, we execute all our test cases and continue this cycle until we finish all our improvements. Finally, we have reached the end of the TSAT process iteration. We start the cycle over once we identify a new threat.

TSAT projects apply the (TSAT) process in the development of each test case. TSAT is composed of short iterations, each resulting in a deliverable product every iteration of the process cycle. Each iteration of the cycle, we demonstrate a new test case that can detect a different type of cyber-attack.

3.1.1 TSAT PROCESS IN THE 3D PRINTING PROCESS

The TSAT process has a nonlinear relationship between a TSAT program and all the steps before printing in the 3D printing process. Figure 7 observes defects we introduce in each of the 3D printing process steps, followed by the remaining path back to the TSAT program.

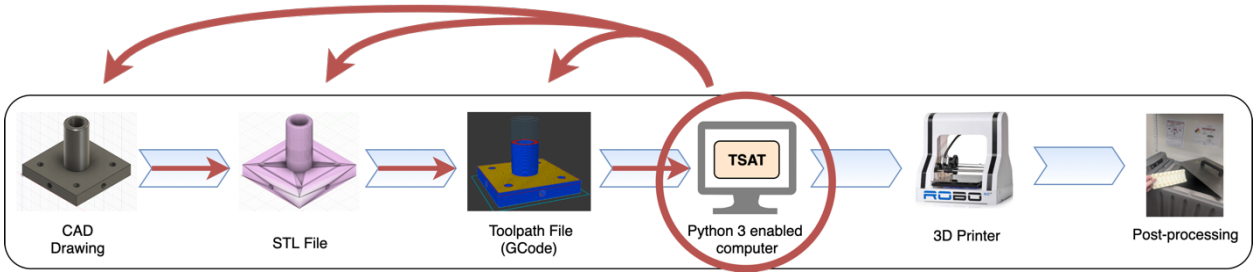

Figure 7: TSAT process in the 3D printing process

3.1.2 TSAT INDEPENDENCE

TSAT is the process of developing test cases that execute on files we intend to run on a known 3D printer. In reality, our test cases do not govern the 3D printer; therefore, any programming language can develop TSAT programs, assuming the language can compare values from a toolpath file to known printer parameter values.

TSAT is independent of the 3d printer or the parts the printer creates. The test case developer decides if the TSAT project is part-based, printer-based, or both.

3D printer operators typically develop TSAT test cases based on their existing printer infrastructure. The slicer, number of slicers, printer, toolpath file type, test case type, number of test cases are all independent because TSAT is simply a process of developing test cases to prevent AM sabotage.

3.1.3 TSAT COMMON PRACTICES

Ideally, the developers creating the TSAT projects are the 3D printer operator or someone with intimate knowledge about the printer's 3D printing process. For this reason, we can make the following assumptions:

- The slicer(s) that generate the toolpath files are known.
- The physical limitations of the 3D printer are known.
- The toolpath file type is known.

The following are common practices that we should implement in TSAT projects: reject unknown slicers, save individual printer configurations in standalone files, reject unknown toolpath file types.

3.1.3.1 Slicer

Only use slicers that append the name of the slicer that generated the toolpath file. We should have at least one test fail if the program cannot determine the slicer's name from the toolpath file. The test cases were developed by someone who knew which slicers we use for the 3D printer. Therefore, we should treat all other slicers as if an adversary compromised the 3D printing process's slicing step.

If an adversary sliced the toolpath file with the correct slicer, we need enough test cases to partially mitigate some risks. If they sliced the file with the wrong slicer but appended with the correct slicer heading, we should include enough test cases that are dependent on the correct slicer toolpath format to fail when given a file with the invalid format.

3.1.3.2 3D Printer Parameters File

When we execute the program, we should import standalone files that contain the printer's limitations. This file should remain unchanged for the duration of the printer's lifetime. One

exception to this rule is if we physically modify the printer, thus, changing the limitations (i.e., making the Z-axis taller). When we replace the printer or add multiple printers, we should create new files to store individual printer parameters; one for each printer. This way, our printer settings stay constant.

3.1.3.3 Toolpath File Type

Since we already know what file types our printer can accept, we should include a test case that fails when the toolpath file type is unknown. If we use multiple printers with different toolpath file types on the same TSAT program, each file type should have a unique class. That way, we skip test cases in inapplicable classes and conserve our resources.

3.2 SOLUTIONS FOR OBJECTIVES

3.2.1 HIGH-LEVEL DESIGN

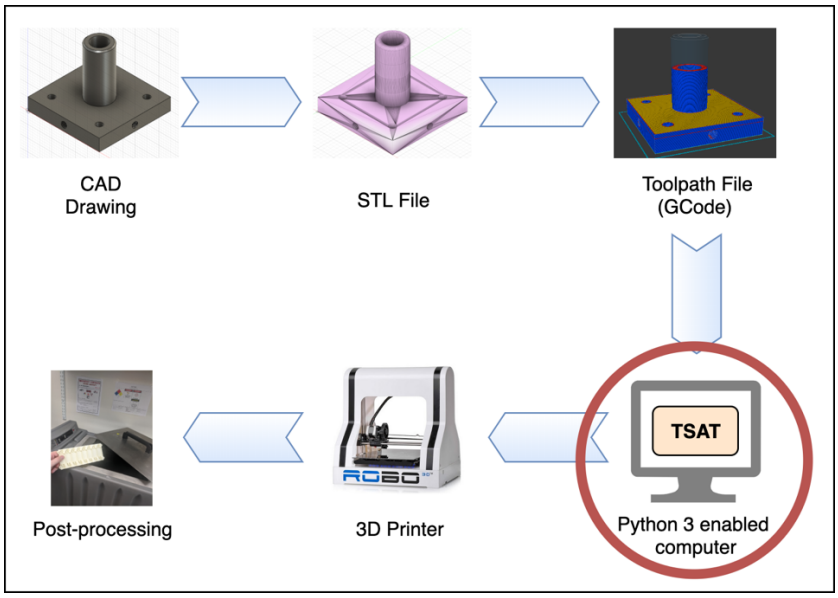

Figure 8: TSAT in the 3D printing process

Figure 8 shows how TSAT fits into the 3D printing process. Once the model is made, converted into an STL file, and sliced into G-code, we can begin the TSAT process. The TSAT

process begins with importing the G-code file into a Python 3 enabled device. TSAT inspects the G-code and runs tests to determine if the file can be printed on the given 3D printer.

A TSAT project composition is presented in Figure 9. There are two types of test cases: operator-based and printer-based test cases. Operator-based test cases are configured on a per-print basis by the 3D printer operator. The purpose of these test cases is to validate that the component meets the original design specifications. Printer-based test cases are tailored to the 3D printer's design capabilities. During the initial TSAT configuration, the 3D printer operator sets variables for these printer-based test cases. Ideally, the configuration file should not change for the lifetime of the printer. Both types of tests help determine potential component defects, accidental or malicious.

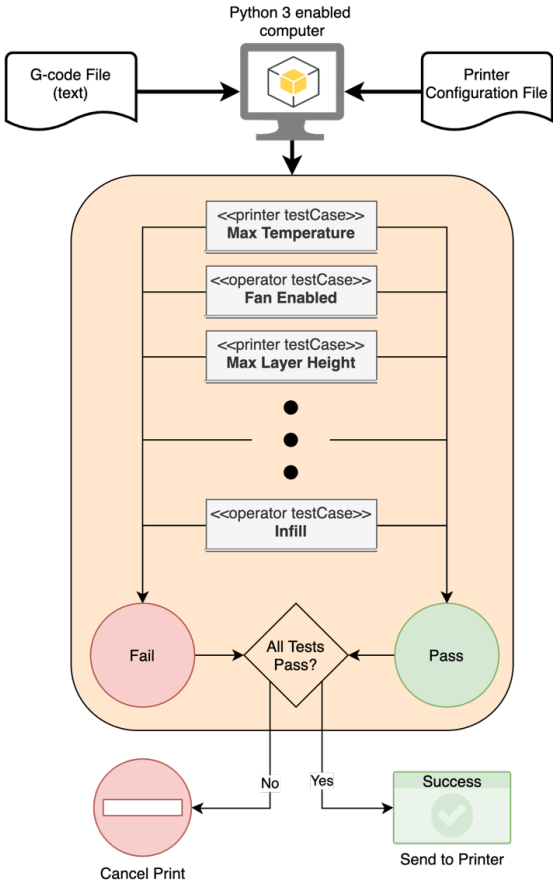

Figure 9: TSAT project composition

3.2.2 OBJECTIVE #1: TEST CASES

Objective: determine and create test cases that ensure the mitigation of an acceptable level of risk.

TSAT aims to mitigate some of the risks of printing a component before the printing process. To achieve this goal, we incorporate test cases that target five common areas involved in AM sabotage: extruder temperature, head bed temperature, fan speed, infill path, and layer thickness [8][19].

3.2.3 OBJECTIVE #2: PENETRATION TESTING

Objective: detect malicious defects made in the 3D printing process before printing and TSAT.

Multiple stages of the 3D printing process can introduce attacks on 3D printers. Therefore, it is vital to test TSAT at all these stages before printing. This project's approach is to intentionally create malicious files that, if not caught, would sabotage the printer. Therefore, to test our approach, malicious 3D model, STL, and toolpath files are needed.

3.2.4 OBJECTIVE #3: PRINTERS THAT USE CMB FILES

Objective: demonstrate that TSAT *can be implemented on more than one type of toolpath file.*

The most widespread 3D printer is residential FDM 3D printers that use G-code. However, not all FMD printers use G-code. Most Stratasys industrial 3D printers use Coordinate Machine Binary (CMB) files instead of G-code. Therefore, in addition to our G-code test cases, some of our test cases for this project need to execute on CMB files to demonstrate that TSAT applies to FDM printers that do not use G-code toolpath files.

3.2.5 OBJECTIVE #4: TSAT REPOSITORY

Objective: create testing environments that are easier to use than not use.

An ideal TSAT project contains test cases that confirm slicer settings and prevent AM sabotage (malicious or accidental). While writing specific test cases can be challenging, ideal users

only need to run the program before printing. This paper aims to establish a TSAT test case repository to create testing environments that are easier to use than not use.

The easier to write TSAT test cases, the more likely developers will develop additional test cases, which increase the TSAT project's value because each test case detects a defect. Increasing the project's number of test cases increases the number of defects the program can identify.

3.3 TSAT IMPLEMENTATION CHALLENGES

3.3.1 UNDERSTANDING G-CODE

G-code is essentially a list of X, Y, and Z-axis instructions that guide the print head to every print point. The G-code also describes how the printer should manufacture the part. G-code supplies the printer with settings such as how hot to heat the filament, or how fast the print head should move when traveling between points.

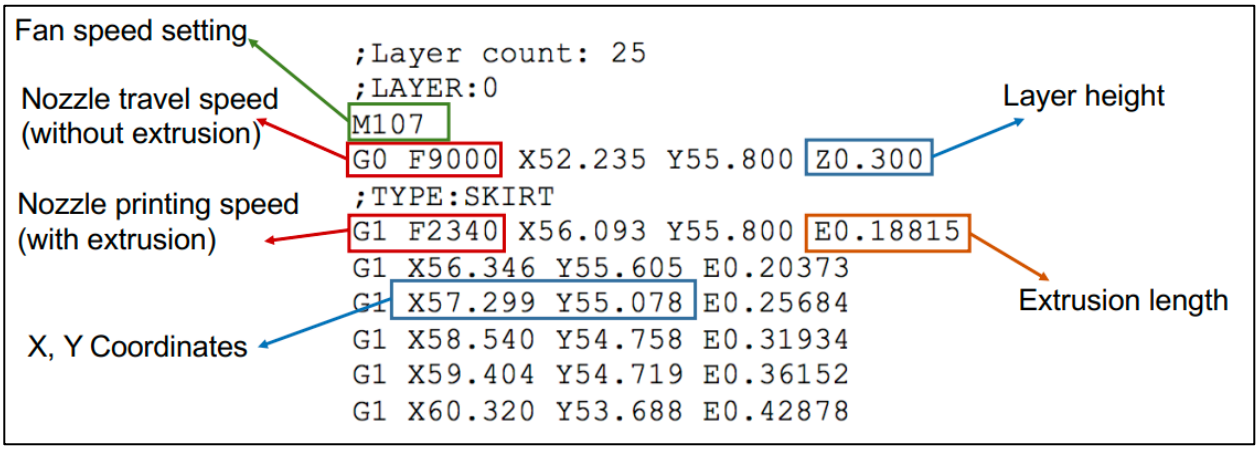

Figure 10: An example of the main body in G-Code [8]

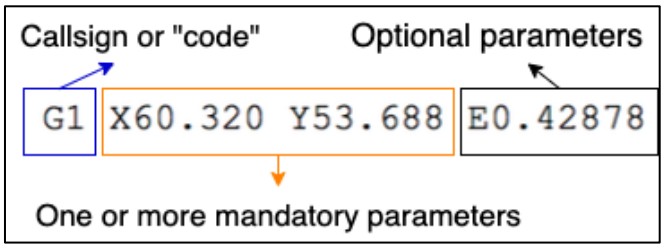

Figure 11: Individual g-code line composition

Figure 10 illustrates several ways the G-code configures 3D printer settings. The G-code implements these settings in two ways: callsigns and implied. Callsigns are provided at the beginning of each non-commented line (Figure 11). Callsigns describe the objective of each line of instructions. In Figure 10, the first non-commented line (comment lines begin with a semicolon) is “M107,” its purpose is to turn off the fan.

Some codes, such as “M107”, do not have any parameters. Other callsigns, such. as “G0”, have parameters that affect how the code is carried out by the printer. “G0” is the callsign to move the print head to a location without extruding any filament. Figure 12 depicts a mock G-code file where each line represents the only command for that layer. We observe that the last parameter in all the lines sets the layer height. We can also tell that the layer height is the sum of all the previous layer heights plus the proceeding layer’s height. We cannot assume that the layer thickness is equal to the first layer’s height because the first layer’s height can be set to a different amount. Therefore, since there is no callsign for setting the layer’s thickness, the layer height is implied whenever repositioning the printer head with the Z-axis parameter. If we wanted to find the layer thickness (in Figure 12’s case, it is 0.03), we would have to take a layer height value and subtract the previous layer height value (as long as the previous Z-axis value is not the first layer height).

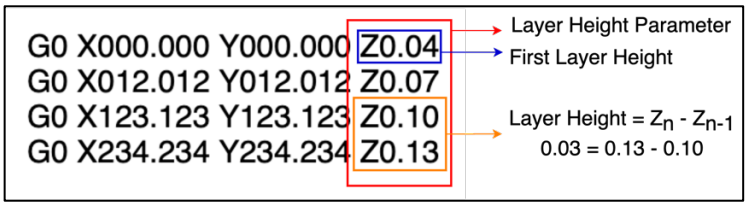

Figure 12: Mock layer height g-code

G-code provides all the information the printer needs to manufacture a component. Essentially, G-code is a road map. It provides the printer with a list of line-by-line instructions so that the print head can navigate the print area. These instructions are a legitimate sequence of instructions that operates the printer within allowable (and presumably safe) bounds.

In many cases, the G-code lines' format depends on the software that sliced the file. Many slicers, open and closed sourced, include the slicer's name in the comments of the toolpath file. Extracting this information can be vital to test cases that use regular expressions to capture values. In which case, the slicer's name is frequently located in the first few lines of the G-code file.

3.3.2 CHALLENGE 1: FILTERING THE G-CODE

Locating malicious instructions is challenging because even a small toolpath file may consist of many thousands of lines of G-code. This project's response to this issue is regular expressions (regex). Regex can filter large amounts of text by applying select parameters. For example, if only "G0" and "G1" commands are needed, we could apply the "[G][0-1]" regular expression to discard any other commands other than "G0" or "G1." Thus, simplifying the code necessary to write a test case.

3.3.3 CHALLENGE 2: ABSENCE OF G-CODE CODES

Numerous 3D printer settings translate directly to the toolpath file. For example, extruder temperature has a code value of "M109." However, not every 3D printer setting has a designated

G-code callsign. Layer height, for example, can only be found by taking the difference of Z-axis heights.

The absence of codes can pose a difficult challenge when writing advanced test cases. Without codes to reference, the amount of programming skill and G-code understanding rises exponentially. Unfortunately, there is no other way around the issue. For test cases that do not involve G-code callsigns, developers must analyze toolpath files to discover the relevant data and then compose algorithms that can replicate their discoveries.

3.3.4 CHALLENGE 3: G-CODE OPTIMIZATION

Individual lines of G-code are relatively easy to comprehend. However, due to the G-code's verbosity and lack of visualization, blocks of code can be unintuitive and misleading to the test case developer.

Infill layers are a case where G-code lines can challenge understandability. It is common practice for 3D printers to only use a small portion of filament for inner layers. The exposed portions of the print are known as "shells." Shells are one hundred percent filament. Only the amount of filament needed to maintain the print's integrity is used on the non-surface portions of the print to conserve resources. These internal structures are known as "infill" and are usually printed in a repeating pattern, such as a grid, triangles, or hexagons. Though, to optimize the number of moves the print head makes, rarely does the printer follow a "left-to-right" or a "top-to-bottom" pattern completely throughout the layer, which can make writing a standard algorithm challenging.

Instead of devising an algorithm to detect and assess infill toolpath from G-code instructions, we elected an indirect approach that relies on a convention of annotating G-code

instructions that carry out infills. In short, we searched comments for text such as “infill” to identify our desired values.

3.3.5 CHALLENGE 4: G-CODE VS. CMB

Edit As: Hex ▾ | Run Script ▾ | Run Template ▾

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000h:	33	20	33	20	0E	08	05	01	0A	D7	23	3C	F2	92	24	3F
0010h:	15	85	0E	3D	04	00	00	00	50	34	33	30	04	00	00	00
0020h:	53	52	33	30	BF	C7	66	40	BF	C7	66	40	39	FE	55	3C
0030h:	FF	2F	96	40	FF	2F	96	40	F1	2C	91	3F	89	8F	28	39
0040h:	EA	78	CC	39	6B	B3	52	39	8F	FC	C1	39	23	B0	83	3C
0050h:	77	BE	1F	3D	2C	9C	A4	3C	50	8D	17	3D	37	C3	CD	3F
0060h:	9A	99	79	40	03	9A	00	40	CD	CC	6C	40	6E	00	00	00
0070h:	A5	72	03	00	1A	00	00	00	57	72	69	74	74	65	6E	20
0080h:	62	79	20	43	61	74	61	6C	79	73	74	45	58	21	21	21
0090h:	21	21	01	00	00	00	0F	00	00	00	31	20	49	6E	63	68
00A0h:	20	43	79	6C	69	6E	64	65	72	0A	00	00	00	38	2E	32
00B0h:	20	28	34	33	33	39	29	04	00	00	00	70	61	69	61	03
00C0h:	00	00	00	54	31	36	03	00	00	00	54	31	36	46	07	00
00D0h:	00	10	03	00	00	00	11	00	00	00	04	00	00	00	12	00
00E0h:	00	00	08	00	00	00	14	00	00	00	04	00	00	00	11	01
00F0h:	00	00	00	11	30	00	00	00	12	D4	8B	E7	3E	67	21	8D
0100h:	3F	12	56	18	E7	3E	52	19	8D	3F	12	15	A5	75	3E	86
0110h:	C9	82	3F	12	C0	9D	74	3E	89	B8	82	3F	12	61	BB	73
0120h:	3E	A2	A0	82	3F	12	E6	DD	88	3D	35	DA	56	3F	12	3F

Figure 13: Stratasys CMB File Raw Hex Data [13]

Because G-code was developed to be portable among different machining devices [14], G-code can be somewhat easy to read for humans. Other 3D printer languages, such as CMB files, were not designed for human consumption. Rather than using text, CMB files store information as binary numbers of varying lengths, but usually in four-byte longs. Raw CMB files become understandable to humans only when viewed in hexadecimal format (Figure 13).

TSAT is not bound to a 3D printing language. Instead, it is a process of developing test cases to prevent AM sabotage based on the individual 3D printer. Since G-code is the most common 3D printing language, G-code was the focus of this paper. However, depending on the printer’s architecture, it might not be applicable.

4. CHAPTER-4: SOLUTION VALIDATION

4.1 TSAT HARDWARE AND SOFTWARE

Below is a list of software specifications to implement the TSAT process and validate the results:

- Ubuntu (version 19.04 64-bit) Virtual Machine
- Memory: 3.8 GiB
- Processor: Intel® Core™ i7-8750H CPU @ 2.20GHz × 3
- Disk Size: 21.5 GB
- Python 3.7.3
- CAD Modeling Software / Mesh Editing Software: Fusion 360
- Slicing Software: MatterControl / Slic3r
- Text Editor: Sublime Text 3

4.2 TEST CASES

In all the following test cases, we use the following approach:

- Take the toolpath file path as an input.
- Convert the toolpath file contents to a string.
- Scan the (toolpath file contents) string for specific variables and their respective values.
- Compare values to known values that are safe to run on the printer.
- It is safe to assume that if any test fails, the input file is not safe to run on the 3D printer.

We proceed with two types of test cases: printer-based and operator based. First, we test that values from the toolpath file are under their corresponding (predetermined) limits that the printer can handle. Second, we test to see if the values in our toolpath file match the values set by the printer operator for a specific part.

While the operations to execute the two types of test cases are similar, it is important to note why we perform these two types of tests. The printer-based tests prevent AM equipment sabotage, and the operator-based tests avoid sabotage to the printed part. Our TSAT program implements both types of tests in each of the five common areas involved in AM sabotage. Nevertheless, to eliminate unnecessary redundancy, either the printer-base or operator-based will be covered in the following sections, but not both.

ROW	***CALLSIGN***	***PARAMETERS***	***DESCRIPTION***	***EXAMPLE***	***REGEX***
1	M104	Temperature[S]	Set extruder temperature (not waiting)	M104 S100	M104 S(\d+)
2	M109	Temperature[S]	Set extruder temperature (waits till reached)	M109 S100	M109 S(\d+)
3	M140	Temperature[S]	Set bed target temperature (not waiting)	M140 S40	M140 S(\d+)
4	M190	Temperature[S]	Set bed target temperature (waits till reached)	M190 S50	M190 S(\d+)
5	M106	Value [S 0-255]	set fan speed to S and start	M106 S123	M106 S(\d+)

Table 2: G-code commands

Table 2 describes the callsigns that we will use in the subsequent test cases (where applicable). After the G-code file contents have been converted to a single string, for each element in the "callsign" column, we apply the corresponding regular expression from the "regex" column. Parts of each regular expression are used to locate the corresponding G-code callsign; other parts are used to extract the desired data.

For example, let us take the string “M106 S0\nM104 S200\nG1 X1 Y2 Z3 ;some comment”, and we want to determine the set extruder temperature (no waiting). Then, our desired callsign is “M104” and our corresponding regular expression is “M104 S(\d+)”. Our regular expression would search our string for the substring “M104 S”, followed by one or more digits. Parenthesis in regular expressions indicates the values we want to return. Therefore, after we apply the regular expression to our string, it locates “M104 S200” and returns “200”.

4.2.1 EXTRUDER TEMPERATURE

G-code callsigns: M104 and M109 (rows 1 and 2 in table 2)

First, we read the G-code file and store the contents in a string variable. Second, we use regular expressions to filter the (G-code contents) string for the M104 and M106 variables and store the respective values in a list. At this point, we have a list of values and some comments that our slicer included. Finally, we iterate through the list and compare each value to our predetermined max temperature value. Our test passes or fails if any of the list elements are higher than the max temperature value. See Appendix-A to view the test case code.

4.2.2 BED TEMPERATURE

G-code callsigns: M140 and M190 (rows 3 and 4 in table 2)

Our approach to writing bed temperature test cases is nearly identical to the test cases used to test extruder head temperature. First, we use the following regular expression: “M140 S(\d+)” (substitute a ‘9’ for the ‘4’ to test for M190) on the file contents string to capture the M140 or M190 variables and values. A list stores the results of the filter. Finally, we compare the values in our list to zero and our predetermined set bed temperature value. Our test passes or fails if the value does not equal zero (where the G-code has start and exit procedures), or our predetermined set bed temperature value.

4.2.3 FAN SPEED

G-code callsign: M106 (row 5 in table 1)

First, we use the following regular expression to capture the callsign: ‘M106’ If the result is “None” (null), then our G-code does not contain any commands to turn on the fan. The lack of fan commands could result from our printer not having any fans, a slicer error, or malicious intent. If the printer does not have any fans, then this test case should be omitted from testing. Otherwise, the 3D printer operator needs to investigate the file for malicious activity and decide if they should proceed with the print.

4.2.4 LAYER THICKNESS

Layer thickness is one of the key parameters for influencing mechanical properties in FDM printers [19]. For example, PLA specimens have higher tensile strength with small values of layer thickness [19]. Moreover, if an adversary were to increase the layer thickness for PLA parts, or change the layer thickness throughout the part, they could weaken the printed parts. Increasing the layer thickness past the printer’s limits could disable the printer by causing the filament to clog in the print head. To prevent such cases, test cases that check for inconsistent or inappropriate values can be developed by determining the layer thickness values.

A specific variable does not indicate layer thickness in the G-code. Instead, we use the following regular expressions and store the results in a list:

- “G1 Z(-*\d+\.*\d*) *F*\d*\.*\d*\nG1 E”
- “G1 Z(-*\d+\.*\d*) *F*\d*\.*\d*\n”

Multiple regular expressions are required as a result of the various slicing software used to create the G-code for this project. Therefore, if the 3D printer operator only uses one slicer, only one regular expression is necessary.

Each section of this regular expression has a different role in finding the layer height. “G1 Z”: layer height is equal to the difference between “G1 Z” commands. Therefore, we want to eliminate any commands that do not contain “G1 Z.”

"(-*\d+\.*\d*)" : Next, we capture the Z-axis digits. The layer height is will likely be a floating-point integer, but we still need to interpret integer values when the layer height equals a whole number.

" *F*\d*\.*\d*": Different slicers process layer height differently. We include this part of the regular expression for slicers that append feed rate values while adjusting the layer height (“G1 Z1.2 F3456,” for example).

“\nG1 E”: Often, ‘G1 Z’ commands are used for increasing layer height and repositioning the extruder head. Repositions do not make any extrusions. Therefore, we eliminate any G1 command that is not followed by another G1 line that contains an extrusion. This portion of the command is the only difference between the two regular expressions. Two regular expressions exist because one slicer does not make any printer head repositions during layers, and the other does. The program knows when to apply each regular expression by searching the first G-code line to determine which software performed the slicing. While we cannot say that all slicing software operates in this fashion, the individual or organization can guarantee they know which slicers they have in operation. Therefore, if we, the 3D printer operator, develop test cases based on our known slicing software, we can guarantee the name of the slicer will be in the G-code file, assuming our software is supposed to append the slicer’s name in the comments of the G-code file.

Some places in the G-code use negative “Z” values to reposition the print head when moving to a different part of the layer, but these values are discarded by the “*F\d*\.*\d*\nG1 E”

section of the regular expression. However, we want to keep both positive and negative Z-axis values that apply to layer thickness to help search for operator errors and malicious activity.

Now we have a list of all the proper Z-axis heights. We proceed by iterating the elements of the list, each iteration labeling the current and next elements. As stated previously, the layer height is the difference between the Z-axis at different layers. Because we find the layer height by subtracting the current element from the next element, the test cases pass or fail if the difference is not zero (because there are several infill portions in each layer) or our predetermined layer height.

4.2.5 INFILL

Similar to layer height, there is no explicit and specific callsign reference to determine the infill portions of a 3D printed component. Instead, we search the toolpath file for comments on the settings used throughout the G-code. Slicers place these comments in the toolpath file to increase readability.

We proceed by first determining which slicer created the toolpath file. Here, regular expressions are unnecessary; all we need to do is search for is “MatterSlice” or “Slic3r,” the two slicers used for this project. If neither of the slicers created the toolpath file, the test fails because the program cannot accurately determine the infill density variable.

Now that we know which slicer was used, we can determine which regular expression to apply to determine the infill variable. If MatterSlice was used, our regular expression is "; infillPercent = (-*\d+\.*\d*)". If Slic3r was used, then our regular expression is "; fill_density = (-*\d+\.*\d*)%".

After our regular expression is applied and our value extracted, the infill test cases are merely comparing the extracted value to the limits set in the program. If they are out of bounds or not equal to the value set by the 3D print operator, the test fails.

4.2.6 CONCLUSION

The above test cases describe how to approach scenarios from one or more slicers. There are various other ways to approach these methods. The focus on the sections above should not be on how these test cases are demonstrated. Instead, the focus should be on that it is possible to show that these test cases are possible on FDM printers.

TSAT is a process that, once implemented on an individual 3D printer, can allow the operator to foresee problems before sending a component to the printer. Thus, this process can save time, money, and prevent printer damage.

4.3 PENETRATION TESTING VALIDATION

4.3.1 CAD MODEL FILE

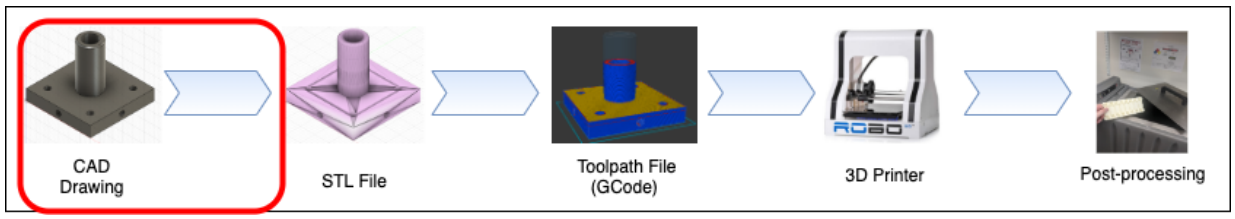

Figure 14: CAD model attacks in the 3D printing process

For AM cyber-attacks, not just sabotage, the CAD model file is the most valuable in terms of information. These files contain all the part's geometric date. Figure 14 depicts where attacks on CAD model files take place in the 3D printing process.

From an adversarial point of view, at this point in the 3D printing process, we do not have access to printer controls, such as extruder temperature. We could insert voids or adjust angles to

change or weaken the mechanical properties, thus sabotaging the part [23]. These types of attacks primarily affect the print, but it is conceivable that a coordinated attack on multiple 3D printers could result in a power outage, thus sabotaging the printer [32]. Additionally, these attacks can cause side effects that consume an abnormal amount of electricity, materials, and time. Nonetheless, our attack method is limited to printing objects outside the printer's X, Y, and X-axis limits.

Figure 15 depicts the benign CAD model file that we weaponize to sabotage a 3D printer. We begin by opening the benign file in CAD modeling software (in this case, Fusion 360). Next, we use the scaling tool to scale the component 15 times its original size. Our component increases in size from 16 mm (Figure 15) to 245mm (Figure 16). We also use Fusion 360 to export and convert our malicious CAD model file, as well as the original benign file, to STL files.

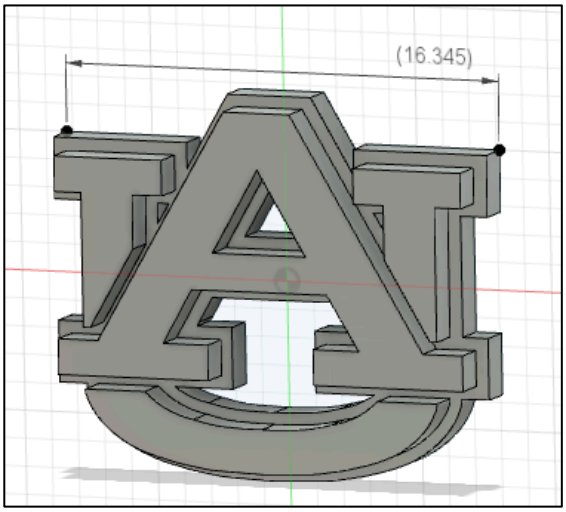

Figure 15: Benign CAD model file (16 mm)

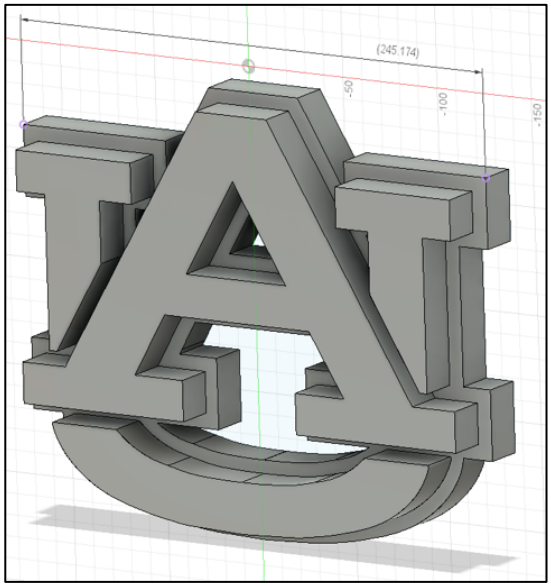

Figure 16: Malicious CAD model file (245 mm)

We next use Slic3r, to slice our STL files to G-code by first indicating that the print bed is twice as large as it is physically (Figure 17). Then, we slice our file and export the G-code (Figure 18).

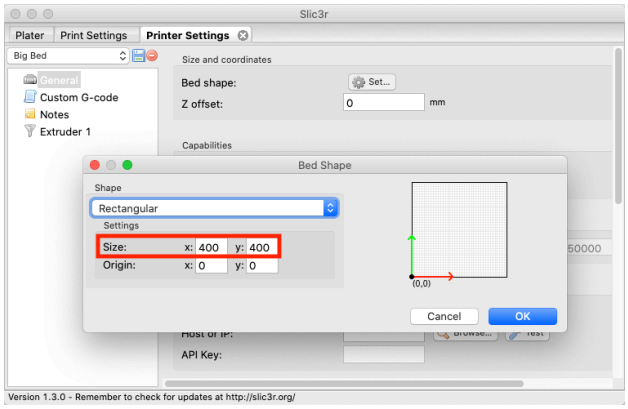

Figure 17: Print bed configuration

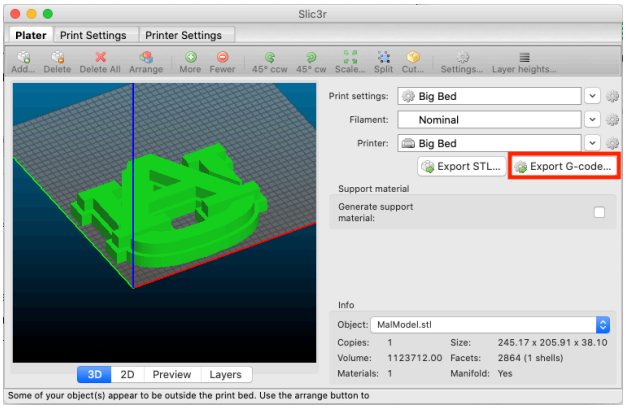

Figure 18: STL to G-code

Finally, we feed the G-code files into our TSAT program and view the results (see Appendix B.1 and B.2). Our benign file passes all tests, whereas our malicious file fails two tests: "exceeds max X size" and "exceeds max Y size." The latter file fails because the target printer's X and Y limits were set in the program before the attack. Therefore, when the G-code values do not meet the program's printer requirements, the file fails to print.

To manufactures, theft, or loss of these files can be costly. However, because CAD model files can still be edited at this stage, attacks at this point in the 3D printing process are at an increased risk of being discovered.

4.3.2 STL File

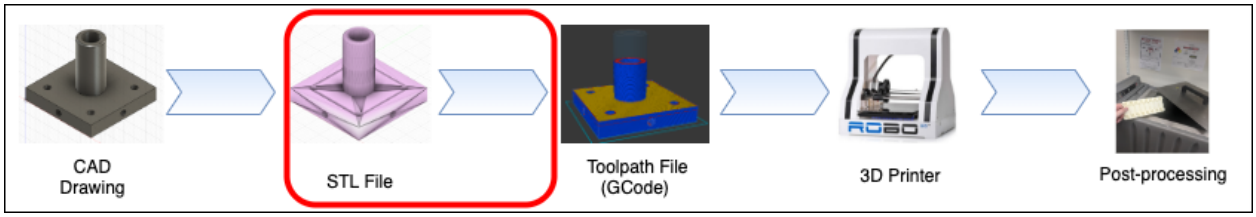

Figure 19: STL attacks in the 3D printing process

Once we convert the model to an STL file, only the data that describe the part's surface remains. All other information describing the model is lost. This data, once represented by complex mathematical equations, is replaced with triangular planes, called facets. The use of facets makes

reverse-engineering the component a challenge, but STL files still contain all the data needed to fabricate the part's geometry. It is still possible to edit the model by changing the vertices of the facets [23]. These vertices are how we attack the STL component. Figure 19 shows where attacks on STL files take place in the 3D printing process.

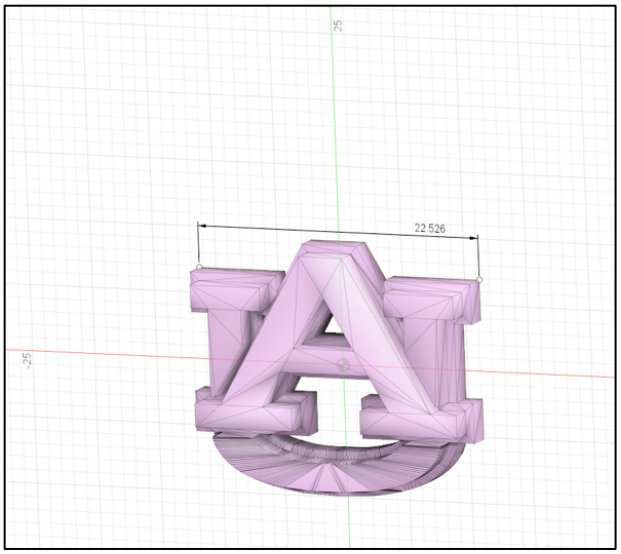

Figure 20: Benign STL file

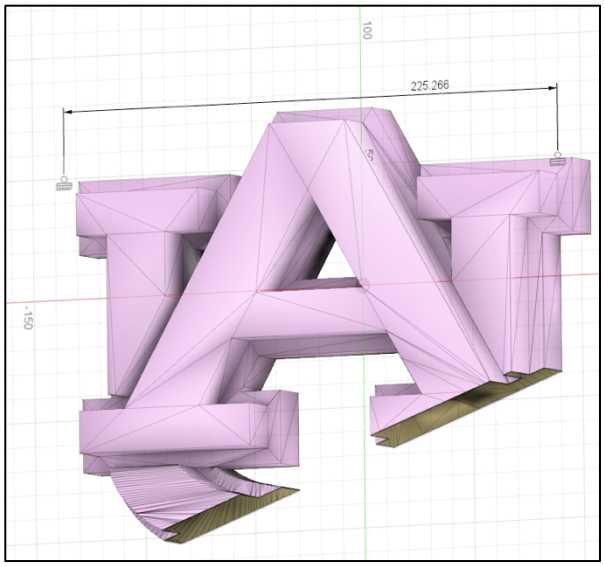

Figure 21: Malicious STL file

We previously made our defects in the CAD model stage in the 3D printing process; now, we begin our attack at the STL file stage. We start by opening the STL in Fusion 360 to view the component's mesh. Our attack uses the same approach as our previous attack on the CAD model. We will use the mesh's vertices to scale the mesh ten times its original size. Figures 20 and 21 show that the mesh's length has increased from 22 mm to 225 mm, a distance that exceeds the target printer's limits. We retain a copy of the original mesh to compare the results when running the files through the TSAT program.

Once the mesh is at the desired size, we export the STL files from Fusion 360 and open the STL files in Slic3r. We use the same extended print bed settings from our last attack and slice the STL's to a G-code format. Next, we run our newly created G-code files though TSAT to see if the program deems them safe to print on the 3D printer.

The results observed in Appendix B.3 and B.4 are as follows: the benign G-code file meets all the requirements stored in TSAT. However, the malicious G-code file fails the same tests as our last attack: “exceeds max X size” and “exceeds max Y size.” Thus, the TSAT program has deemed the latter file unfit to be printed on the target printer.

4.3.3 TOOLPATH FILE

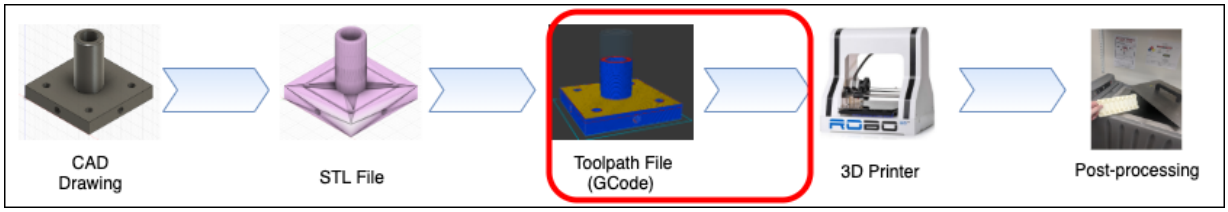

Figure 22: Tool file attacks in the 3D printing process

Figure 22 indicates where toolpath (G-code) file attacks take place. Toolpath files offer the most flexibility in what the attack is trying to accomplish. These files are often seen as less valuable because toolpath files are bound to a specific 3D printer upon creation. However, these files also contain all the printer’s operations, which can be altered. Therefore, our next attack affects the extruder temperature, print bed temperature, and the print head boundaries.

Our attack takes place on a toolpath file that has already been sliced. To increase diversity and demonstrate that TSAT can be implemented on multiple slicers, these files were sliced with MatterSlice, the default slicer for our target printer. We open the toolpath file in a text editor (in this case, Sublime Text 3) and change the values observed in Figures 23 and 24. We save the file as a new file to observe the initial and altered values in the TSAT results.

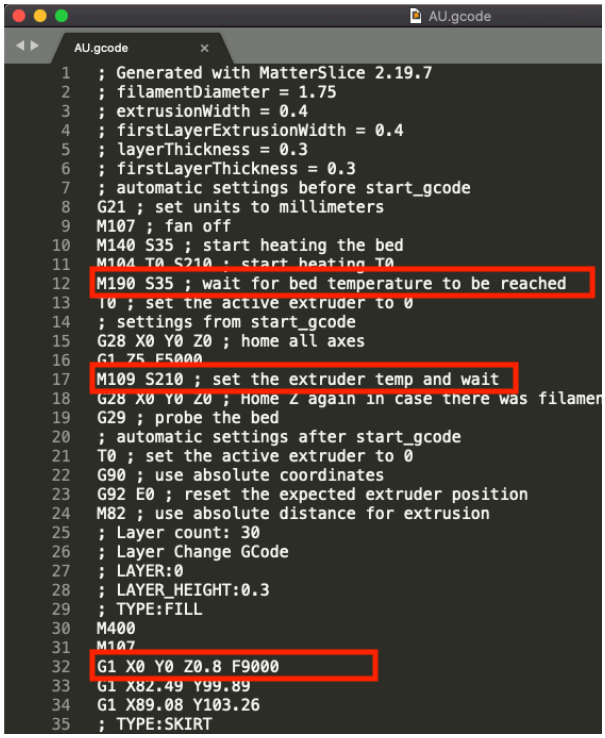

Figure 23: Benign g-code file

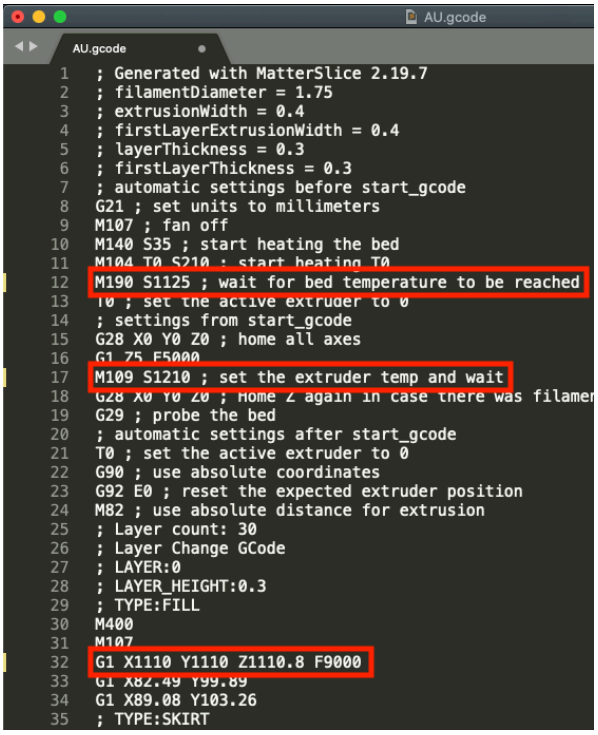

Figure 24: Malicious toolpath file

The results are as follows: the benign file failed one test case (see Appendix B.5), and the malicious file failed eight test cases (see Appendix B.6). In both files, the "fan never engaged" test failed because our slicer does not activate the fan on small prints to increase adhesion to the print bed. This error could be a result of a bug or a design decision made by the MatterSlice developers, or could indicate compromised slicing software.

The remaining failures directly correlate to our changes made to the toolpath file. When we increased the bed temperature from 35 to 1125 degrees Celsius, the "bed temperature" and "max bed temperature" tests failed. When we increased the extruder temperature from 210 to 1210 degrees Celsius, the "extruder temperature" and "max extruder temperature" tests failed. Finally, the "exceeds max X size," "exceeds max Y size," and "exceeds max Z size" tests failed when we modified X, Y, and Z printer head coordinates from "X0, Y0, Z0.8" to "X1110 Y1110 Z1110.8."

All these failures respond to the G-code values either not matching the part parameters or exceeding the printer parameters set in TSAT.

The TSAT program has deemed both files unfit for printing. It is up to the 3D printer operator to decide whether to send the prints to the printer. Since the malicious file failed numerous essential tests, the file is clearly unfit to send to the printer. The benign file also failed the TSAT test cases. However, it is not clear whether it should be sent to the printer. It is important to note that the 3D printer operator's decision is irrelevant. What is relevant is that the operator was given the opportunity to decide and justify that decision. This opportunity was made possible through TSAT. Moreover, these events happened before the printing process, instead of being discovered after the fact.

4.4 NON-G-CODE TOOLPATH FILE VALIDATION

4.4.1 UNDERSTANDING CMB FILES

Objective #3: develop TSAT so that it can be applied to other types of 3D printers.

To accomplish our third objective, we need to understand the underlying structure of CMB files. CMB files are binary files, with most data stored as floating-point numbers (or four bytes). These files are composed of three sections: header, toolpaths, and End-of-File [13]

AU4.cmb

000000d4	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000000	33	20	33	20	0e	08	09	01	0a	d7	23	3c	cb	af	66	3d	3 3 ....W#<K/f=
00000010	1a	ab	e7	3c	04	00	00	00	50	34	33	30	06	00	00	00	.+g<....P430....
00000020	50	34	30	30	53	52	66	5d	6a	40	69	f1	6e	40	39	fe	P400SRf]j@iqn@9~
00000030	55	3c	33	66	94	40	33	1c	92	40	10	50	8a	3e	6b	b3	U<3f.@3..@.P.>k3
00000040	52	39	ea	78	cc	39	79	a1	3d	39	90	06	9e	39	2c	9c	R9jxL9y!=9...9,.
00000050	a4	3c	77	be	1f	3d	27	26	94	3c	41	ea	f6	3c	03	9a	$<w>.='&.<Ajv<..

Figure 25: CMB file header section

In Figure 25, we can see the header section opened in a hex editor (Hex Editor Neo). In Figure 26, we can see the composition of the header section and their corresponding sizes.

Magic Number	(1 float)	Sample Text
Machine Type	(1 float)	Sample Text
Slice Height	(1 float)	Sample Text
Part Volume	(1 float)	Sample Text
Support Volume	(1 float)	Sample Text
Int # of Str Chars	(4 bytes)	Sample Text
Part Material Str	(n bytes)	Sample Text
Supp Material Str	(n bytes)	Sample Text
Part Min X	(1 float)	Sample Text
Part Min Y	(1 float)	Sample Text
Part Min Z	(1 float)	Sample Text
Part Max X	(1 float)	Sample Text
Part Max Y	(1 float)	Sample Text
Part Max Z	(1 float)	Sample Text

Figure 26: Sub-sections of CMB file header section

4.4.2 CMB FILE PENETRATION TESTING

The header section of the CMB file provides all the information we need to accomplish our objective. We proceed by manipulating the “Part Max X,” “Part Max Y,” and “Part Max Z” hex values. These values indicate the overall part size in their respective axes and are represented as inches and in Little-Endian. Figures 27 and 28 observe the max axes values (highlighted in various shades of gray) before and after modification. The red text indicates the modified values. We save this file as a new file to compare the resulting unchanged and modified values

AU4.cmb

```
000000d4  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00000000  33 20 33 20 0e 08 09 01 0a d7 23 3c cb af 66 3d  3 3 ..... .W#<K/f=
00000010  1a ab e7 3c 04 00 00 00 50 34 33 30 06 00 00 00  .+g<....P430....
00000020  50 34 30 30 53 52 66 5d 6a 40 69 f1 6e 40 39 fe  P400SRf]j@iqn@9~
00000030  55 3c 33 66 94 40 33 1c 92 40 10 50 8a 3e 6b b3  U<3f.@3..@.P.>k3
00000040  52 39 ea 78 cc 39 79 a1 3d 39 90 06 9e 39 2c 9c  R9jxL9y!=9...9,.
00000050  a4 3c 77 be 1f 3d 27 26 94 3c 41 ea f6 3c 03 9a  $<w>.='&.<Ajv<..
```

Figure 27: CMB file before modification

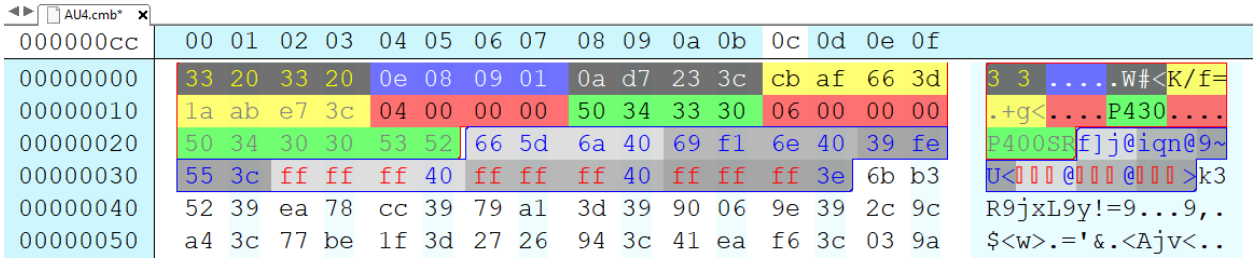

Figure 28: CMB file after modification

Similar to our G-code tests, we use the following approach:

- Take the toolpath file path as an input.
- Convert the toolpath file contents to a string.
- Apply the hexadecimal offset to reach the desired values.
- Compare toolpath values to known values that are safe to run on the computer.
- It is safe to assume that if any test fails, the input file is not safe to run on the 3D printer.

Our CMB file approach differs from our G-code file approach only in how we find the desired values. For G-code files, we used regular expressions. For CMB files, we use the hexadecimal number offset.

After the TSAT program reads in the hex file and eliminates the irrelevant max X, Y, and Z axes characters, we need to perform some manipulations before we begin our tests. First, we convert our hex values from little-endian to big-endian. Second, we convert our values from a hex string to an IEEE 754 floating-point representation. Finally, we convert our floating-point value from inches to millimeters. This puts our values in the appropriate format, allowing us to begin our test cases.

Each test case takes the value retrieved from the CMB file and compares it to the respective value set in the TSAT program. These fixed values represent the 3D printer's physical limitations. If the collected CMB file values exceed the values set in the TSAT program, we can determine that the printer is not capable of printing the file.

4.4.3 PENETRATION TESTING RESULTS

The results are as follows: both files skipped nineteen test cases. Skipped test cases are a result of incompatible file types. Thus, the TSAT program skipped nineteen G-code tests because we are testing a CMB file.

In addition to the nineteen skipped tests, our benign CMB file passed the max X, Y, and Z test cases (see Appendix B.7). When the TSAT program processed the defective CMB file (see Appendix B.8), three tests failed: “Exceeds Max X Size,” Exceeds Max Y Size,” and “Exceeds Max Z Size.”

These penetration tests indicate we can apply TSAT to 3D printers other than printers that exclusively use G-code toolpath files. Thus, we have successfully demonstrated that TSAT meets our third objective requirements

4.5 TSAT REPOSITORY

Not all 3D printer operators have to write their test cases. Test case repositories for slicing software, toolpath file types, programming languages, and methods for determining toolpath file values eliminates overhead for other developer’s TSAT projects. Here, the efforts of a few developers benefit individuals who may have limited programming experience.

This repository aims to encourage approachable testing environments that allow developers to generate test cases that follow the TSAT process. Thus, developing many test cases that compel the user to test toolpath files on TSAT projects before printing, rather than sending the file directly to the 3D printer.

To make test cases more available to developers, we created a GitHub repository [21] with a GNU General Public License v3.0. A synopsis of the license is as follows:

1. Anyone can copy, modify, and distribute this software.
2. Every distribution must include the license and copyright notice.
3. Anyone can privately use this software.
4. Anyone can use this software for commercial purposes.
5. Businesses solely built from this repository risk open sourcing their entire code base.
6. Modifications to the original repository must be indicated.
7. Any repository modifications must distribute the same license, GPLv3.
8. This software provides no warranty.
9. The software author or license is not liable for any damages inflicted by the software.

We incorporated the following four sections to help developers find TSAT project test cases relevant to their needs: languages, methods, slicers, and toolpath file types. The language section provides test cases written in various programming languages. The method section contains supplemental code to determine variables in the toolpath file. The slicer section aids developers in writing test cases on specific toolpath file slicing software. Finally, the toolpath file type section contains test cases implemented on various types of toolpath files.

One drawback of TSAT repositories is that they do not supply either “red light” or “green light” portions of the TSAT process. To complete the TSAT process, first, developers need to create their own defective toolpath files to pass the “red light” portion, then execute all the other test cases, which need to detect their corresponding test cases to pass the “green light” portion.

5. CHAPTER-5: CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

This research aimed to enact a level of security that reduced unnecessary risk for FDM 3D printers. Based on penetration results, we can conclude that a static analysis approach to designing test cases can detect defects in toolpath files before printing. The results indicate that developing programs that identify defects are a possible alternative to traditional QA equipment.

The TSAT process was proven effective at generating test cases that identify defects in G-code toolpath files. This approach identified defects that can sabotage 3D printer's at multiple stages before printing in the 3D printing process. Research results indicated that TSAT applied to other types of toolpath files, such as CMB files.

However, the research results also indicated the limited effects CAD model and STL files had on AM equipment sabotage. Moreover, these limitations extended to the test cases and made it easier to identify threats introduced in the CAD model and STL stages of the 3D printing process. This research could indicate that TSAT was more effective at detecting printer-based defects than part-based defects since the limitations of CAD model or STL attacks did not extend to part-based attacks. Thus, more test cases were needed to detect part-based defects in the toolpath file.

Side effects of an attack can be challenging to identify with TSAT test cases if the defect does not negatively affect the printer. Since no printer-based defects would be detected, it is unlikely the defect will not sabotage the AM equipment but will consume resources such as filament, electricity, and time.

The project successfully generated a test case for every iteration of the TSAT process. Furthermore, the more test cases the TSAT project accumulated, the more secure it became. This research indicated that the TSAT process could improve solutions to existing test cases while addressing new threats upon discovery.

As it stands, TSAT projects cannot stop sabotage attacks, but it can prevent them by creating test cases that analyze the contents of the 3D component's file before being sent to the printer. Residential FDM 3D printers are the most common 3D printer in the world. As a result, most of the world's 3D printers have little-to-no security because of the high overhead costs. Most small businesses that employ 3D printing technology do so with minimal security. Startup companies, who are exceptionally vulnerable, are particularly fond of AM [20]. Metal 3D printer sales increased by 80% in 2017 [29]. As the industry grows and continues to integrate IoT based systems into their printers, the more vulnerable they become to cyber-attacks [8]. While residential 3D printers are the most vulnerable to cyber-attacks, industrial 3D printers are still more appealing targets. One of the most significant benefits that TSAT provides is that it gives at least some security to those who, otherwise, have none.

5.2 FUTURE WORK

The more test cases added to TSAT programs, the more secure they become. More test cases on nozzle size, relative coordinates, multiple extruders, and extrusion amount need development to prevent additional printer sabotage.

Instead of STL files, Additive Manufacturing Files (AMF) are becoming more popular. AMF's are likely to replace STL files since, upon conversion, STL files lose a large amount of information from the CAD model file. Therefore, AMF files may be more vulnerable to

manipulation since they contain much more information. Research on AMF's is essential in future TSAT test cases.

Future efforts could develop TSAT test cases to target part-based attacks, such as void attacks, which insert voids into the printed component to fail under load unexpectedly.

TSAT would be most effective at the firmware level, meaning the printer itself. If the printer's firmware is compromised, benign components can damage the 3D printer. Future TSAT development needs to isolate the firmware from sending control signals to the printer. Thus, creating an interactive environment to execute test cases and observe how the printer would react in a live situation. This step would take TSAT from a static analysis process to a dynamic analysis process. Thus, taking a system that prevents sabotage and graduating to a system that stops sabotage.

REFERENCES

[1] Chris Adkins and Dana Ellis. 2018. NCMS-and-Identify3D-Protecting-the-Additive-Manufacturing-Workflow-with-Blockchain-Technology.pdf. Retrieved from https://identify3d.com/wp-ncms-digitalblockchain/

[2] Mohammed Albakri, Logan Sturm, Christopher B Williams, and Pablo Tarazaga. NON-DESTRUCTIVE EVALUATION OF ADDITIVELY MANUFACTURED PARTS VIA IMPEDANCE-BASED MONITORING. 16.

[3] Christian Bayens, Tuan Le, Luis Garcia, Raheem Beyah, Mehdi Javanmard, and Saman Zonouz. 2017. See No Evil, Hear No Evil, Feel No Evil, Print No Evil? Malicious Fill Patterns Detection in Additive Manufacturing. 1181–1198. Retrieved June 2, 2020 from https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/bayens

[4] Sofia Belikovetsky, Yosef Solewicz, Mark Yampolskiy, Jinghui Toh, and Yuval Elovici. 2017. Detecting Cyber-Physical Attacks in Additive Manufacturing using Digital Audio Signing. *arXiv:1705.06454 [cs]* (May 2017). Retrieved June 2, 2020 from http://arxiv.org/abs/1705.06454

[5] Sofia Belikovetsky, Mark Yampolskiy, Jinghui Toh, Jacob Gatlin, and Yuval Elovici. dr0wned – Cyber-Physical Attack with Additive Manufacturing. 16.

[6] F. Calignano, D. Manfredi, E. P. Ambrosio, S. Biamino, M. Lombardi, E. Atzeni, A. Salmi, P. Minetola, L. Iuliano, and P. Fino. 2017. Overview on Additive Manufacturing Technologies. *Proceedings of the IEEE* 105, 4 (April 2017), 593–612. DOI:https://doi.org/10.1109/JPROC.2016.2625098

[7] Sujit Rokka Chhetri, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. 2016. KCAD: kinetic cyber-attack detection method for cyber-physical additive manufacturing systems. In *Proceedings of the 35th International Conference on Computer-Aided Design* (ICCAD '16), Association for Computing Machinery, Austin, Texas, 1–8. DOI:https://doi.org/10/gfz6cr

[8] Yang Gao, Borui Li, Wei Wang, Wenyao Xu, Chi Zhou, and Zhanpeng Jin. 2018. Watching and Safeguarding Your 3D Printer: Online Process Monitoring Against Cyber-Physical Attacks. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3 (September 2018), 1–27. DOI:https://doi.org/10/ggk3b3

[9] Simon Goldenberg, John Brown, Jeff Haid, and John Ezzard. 2016. 3D opportunity and cyber risk management: Additive manufacturing secures the thread. Retrieved from https://www2.deloitte.com/us/en/insights/focus/3d-opportunity/3d-printing-cyber-risk-management.html

[10] Lynne M. G. Graves, Joshua Lubell, Wayne King, and Mark Yampolskiy. 2019. Characteristic Aspects of Additive Manufacturing Security From Security Awareness Perspectives. *IEEE Access* 7, (2019), 103833–103853. DOI:https://doi.org/10/gg5ng4
[11] Samuel H. Huang, Peng Liu, Abhiram Mokasdar, and Liang Hou. 2013. Additive manufacturing and its societal impact: a literature review. *Int J Adv Manuf Technol* 67, 5–8 (July 2013), 1191–1203. DOI:https://doi.org/10/f45nsk
[12] Ledger Insights. 2019. GE Research builds blockchain for 3D printing supply chain. *Ledger Insights - enterprise blockchain*. Retrieved July 28, 2020 from https://www.ledgerinsights.com/ge-research-blockchain-3d-printing-additive-manufacturing/
[13] Bryan Kessel. 2015. Characterizing and Defending Against Cyber Security Vulnerabilities in Additive Manufacturing. University of Virginia.
[14] Thomas R. Kramer, Frederick M. Proctor, and Elena R. Messina. 2000. The NIST RS274NGC Interpreter - Version 3. (August 2000). Retrieved July 9, 2020 from https://www.nist.gov/publications/nist-rs274ngc-interpreter-version-3
[15] R. Leal, F. M. Barreiros, L. Alves, F. Romeiro, J. C. Vasco, M. Santos, and C. Marto. 2017. Additive manufacturing tooling for the automotive industry. *Int J Adv Manuf Technol* 92, 5–8 (September 2017), 1671–1676. DOI:https://doi.org/10/gbvbn9
[16] Brian N. Turner, Robert Strong, and Scott A. Gold. 2014. A review of melt extrusion additive manufacturing processes: I. Process design and modeling. *Rapid Prototyping Journal* 20, 3 (January 2014), 192–204. DOI:https://doi.org/10/f6fhmc
[17] Jayanthi Parthasarathy. 2014. 3D modeling, custom implants and its future perspectives in craniofacial surgery. *Ann Maxillofac Surg* 4, 1 (2014), 9. DOI:https://doi.org/10/gcb386
[18] Joan Pellegrino, Tommi Makila, Shawna McQueen, and Emmanuel Taylor. 2016. *Measurement science roadmap for polymer-based additive manufacturing*. National Institute of Standards and Technology, Gaithersburg, MD. DOI:https://doi.org/10.6028/NIST.AMS.100-5
[19] Diana Popescu, Aurelian Zapciu, Catalin Amza, Florin Baciu, and Rodica Marinescu. 2018. FDM process parameters influence over the mechanical properties of polymer specimens: A review. *Polymer Testing* 69, (August 2018), 157–166. DOI:https://doi.org/10/gd5kvc
[20] Ress Roberts and Alkaios Bournias Varotsis. 2020. 3D printing trends 2020: industry highlights and market trends. Retrieved from https://www.3dhubs.com/get/trends/
[21] Ash Searle. 2020. *hassearle/TSAT*. Retrieved July 23, 2020 from https://github.com/hassearle/TSAT
[22] Michael Sikorski and Andrew Honig. 2012. *Practical malware analysis: the hands-on guide to dissecting malicious software*. No Starch Press, San Francisco.
[23] L D Sturm, C B Williams, J A Camelio, J White, and R Parker. CYBER-PHYSICAL VULNERABILITIES IN ADDITIVE MANUFACTURING SYSTEMS. 13.
[24] Logan Sturm, Mohammed Albakri, Christopher B Williams, and Pablo Tarazaga. In-Situ Detection of Build Defects in Additive Manufacturing via Impedance-Based Monitoring. 21.
[25] Barna Szabó and Ivo Babuška. 1991. *Finite Element Analysis*. John Wiley & Sons.
[26] Nektarios Georgios Tsoutsos, Homer Gamil, and Michail Maniatakos. 2017. Secure 3D Printing: Reconstructing and Validating Solid Geometries using Toolpath Reverse Engineering. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*

(CPSS '17), Association for Computing Machinery, Abu Dhabi, United Arab Emirates, 15–20. DOI:https://doi.org/10/ggkzrk

[27] Hannah Vincent, Lee Wells, Pablo Tarazaga, and Jaime Camelio. 2015. Trojan Detection and Side-channel Analyses for Cyber-security in Cyber-physical Manufacturing Systems. *Procedia Manufacturing* 1, (January 2015), 77–85. DOI:https://doi.org/10/gfkcm8

[28] Jess M Waller, Bradford H Parker, Kenneth L Hodges, Eric R Burke, James L Walker, and Edward R Generazio. 2014. Nondestructive Evaluation of Additive Manufacturing. (2014), 47.

[29] TERRY T WOHLERS. 2018. *WOHLERS REPORT: 3d printing and additive manufacturing state of the industry.* WOHLERS Associates, FORT COLLINS.

[30] Terry T. Wohlers and Tim Caffrey. 2015. *Wohlers report 2015: 3D printing and additive manufacturing state of the industry annual worldwide progress report*. Wohlers Associates, Fort Collins, Colo.

[31] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems* 2, 1 (August 1987), 37–52. DOI:https://doi.org/10/bm8dnf

[32] Mark Yampolskiy, Wayne E. King, Jacob Gatlin, Sofia Belikovetsky, Adam Brown, Anthony Skjellum, and Yuval Elovici. 2018. Security of additive manufacturing: Attack taxonomy and survey. *Additive Manufacturing* 21, (May 2018), 431–457. DOI:https://doi.org/10.1016/j.addma.2018.03.015

[33] Mark Yampolskiy, Anthony Skjellum, Michael Kretzschmar, Ruel A. Overfelt, Kenneth R. Sloan, and Alec Yasinsac. 2016. Using 3D printers as weapons. *International Journal of Critical Infrastructure Protection* 14, (September 2016), 58–71. DOI:https://doi.org/10.1016/j.ijcip.2015.12.004

[34] 2019. Blockchain secures distributed additive manufacturing in the U.S. Air Force. *3D Printing Industry*. Retrieved July 28, 2020 from https://3dprintingindustry.com/news/blockchain-secures-distributed-additive-manufacturing-in-the-u-s-air-force-160928/

[35] Additive manufacturing and cyber risk management | Deloitte Insights. Retrieved February 19, 2020 from https://www2.deloitte.com/us/en/insights/focus/3d-opportunity/3d-printing-cyber-risk-management.html

[36] 3D Printing in Aerospace & Aviation | GE Additive. Retrieved February 9, 2020 from https://www.ge.com/additive/additive-manufacturing/industries/aviation-aerospace

[37] Protect – Identify3D. Retrieved July 28, 2020 from https://identify3d.com/protect/

[38] Enforce – Identify3D. Retrieved July 28, 2020 from https://identify3d.com/enforce/

APPENDIX A: SOURCE CODE

A.1 TARGET PRINTER CONFIGURATION FILE

```
# OPERATOR LIMITS
TEMP_VAR = 210
BED_TEMP_VAR = 35
INFILL_VAR = 20.0
LAYER_HEIGHT = 0.3
FAN_VAR = 255

# PRINTER LIMITS
MAX_TEMP_VAR = 220
MAX_BED_TEMP_VAR = 65
MAX_INFILL_VAR = 100.0
MIN_INFILL_VAR = 0.0
MAX_FAN_VAR = 255
MAX_X_SIZE = 200.0
MIN_X_SIZE = 0.1
MAX_Y_SIZE = 200.0
MIN_Y_SIZE = 0.1
MAX_Z_SIZE = 200.0
MIN_Z_SIZE = 0.1

# CMB FILE PRINTER LIMITS
CMB_MAX_X_SIZE = 200.0
CMB_MAX_Y_SIZE = 200.0
CMB_MAX_Z_SIZE = 200.0
```

A.2 HELPER FUNCTION FILE: CONVERT HEX STRING TO IEEE 754 FLOAT

```
def hex2Float(hexValues):
    result = 0.0
    binary = "{0:08b}".format(int(hexValues, 16))
    while len(binary) != 32:
        binary = "0" + binary
    sign = int(binary[0])
    exponent = binary[1:9]
    fraction = binary[10:]
    exponent = int(exponent, 2) -127
```

```
        exponent_ = -1
        remainder = 0
        for bit in fraction:
            remainder += int(bit) * (2**exponent_)
            exponent_ -= 1
        result = ((-1)**sign) * (1 + remainder) * (2**exponent)
        return result
```

A.3 TSAT MAIN

```
import unittest
import sys
import re
import ieeeConverter
import targetPrinter as p
```

#THE FOLLOWING ARE TEST CASES ARE FOR G-CODE TOOLPATH FILES

```
class GCode_TestCases(unittest.TestCase):
    GCODE_PATH = ""
    GCODE_INPUT = ""
    SKIP_TEST = None
    TEMP_HEADER1 = 'M104 S(\d+)'
    TEMP_HEADER2 = 'M109 S(\d+)'
    BED_TEMP_HEADER1 = 'M140 S(\d+)'
    BED_TEMP_HEADER2 = 'M190 S(\d+)'
    SLICER_MATTER_SLICE = 'Generated with MatterSlice'
    SLICER_SLIC3R = 'generated by Slic3r'
    FAN_HEADER = '(;)* *M106 S(-*\d+\.*\d*)'
    LAYER_HEIGHT_HEADER1 = 'G1 Z(-*\d+\.*\d*) *F*\d*\.*\d*\nG1 E'
    LAYER_HEIGHT_HEADER2 = 'G1 Z(-*\d+\.*\d*) *F*\d*\.*\d*\n'
    DIGITS = '(\d+\.\d+|\d+)'
    INFILL_HEADER1 = '; infillPercent = (-*\d+\.*\d*)'
    INFILL_HEADER2 = '; fill_density = (-*\d+\.*\d*)%'
    X_SIZE_HEADER = 'G1 X(-*\d+\.*\d*)'
    Y_SIZE_HEADER = 'G1 X*\d*\.*\d* *[Y](-*\d+.\d+|\d+)'
    Z_SIZE_HEADER = 'G1 *X*\d* *Y*\d* Z(-*\d+\.*\d+)'
    TEMP_VAR = p.TEMP_VAR
    MAX_TEMP_VAR = p.MAX_TEMP_VAR
    BED_TEMP_VAR = p.BED_TEMP_VAR
    MAX_BED_TEMP_VAR = p.MAX_BED_TEMP_VAR
    LAYER_HEIGHT = p.LAYER_HEIGHT_VAR
    FAN_VAR = p.FAN_VAR
    MAX_FAN_VAR = p.MAX_FAN_VAR
    INFILL_VAR = p.INFILL_VAR
    MAX_INFILL_VAR = p.MAX_INFILL_VAR
    MIN_INFILL_VAR = p.MIN_INFILL_VAR
```

```
MAX_X_SIZE = p.MAX_X_SIZE
MIN_X_SIZE = p.MIN_X_SIZE
MAX_Y_SIZE = p.MAX_Y_SIZE
MIN_Y_SIZE = p.MIN_Y_SIZE
MAX_Z_SIZE = p.MAX_Z_SIZE
MIN_Z_SIZE = p.MIN_Z_SIZE

@classmethod
def setUpClass(cls):
    if ".gcode" in cls.GCODE_PATH:
        cls.SKIP_TEST = False
        with open(cls.GCODE_PATH, 'r') as f:
            cls.GCODE_INPUT = f.read()
    else:
        cls.SKIP_TEST = True

def setUp(self):
    if self.SKIP_TEST == True:
        self.skipTest("Not G-Code file")

def test100_900_maxTemp(self):
    expectedResult = True
    actualResult = False

    m = re.findall(self.TEMP_HEADER1, self.GCODE_INPUT)
    for element in m:
        current = float(element)
        if current > self.MAX_TEMP_VAR:
            actualResult = "Extruder Temp Error: value(" + str(current) + ") > bounds("
                           + str(self.MAX_TEMP_VAR) + ")"
            break
        else:
            actualResult = True
    self.assertEqual(expectedResult, actualResult)

def test100_901_maxTemp(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.TEMP_HEADER2, self.GCODE_INPUT)
    for element in m:
        current = float(element)
        if current > self.MAX_TEMP_VAR:
            actualResult = "Extruder Temp Error: value(" + str(current) + ") > bounds("
                           + str(self.MAX_TEMP_VAR) + ")"
            break
        else:
```

```
            actualResult = True
    self.assertEqual(expectedResult, actualResult)

def test100_910_temp(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.TEMP_HEADER1, self.GCODE_INPUT)
    for element in m:
        current = float(element)
        if current != 0 and current != self.TEMP_VAR:
            actualResult = "Extruder Temp Error: value(" + str(current) + ") != value("
                        + str(self.TEMP_VAR) + ")"
            break
        else:
            actualResult = True
    self.assertEqual(expectedResult, actualResult)

def test100_911_temp(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.TEMP_HEADER2, self.GCODE_INPUT)
    for element in m:
        current = float(element)
        if current != 0 and current != self.TEMP_VAR:
            actualResult = "Extruder Temp Error: value(" + str(current) + ") != value("
                        + str(self.TEMP_VAR) + ")"
            break
        else:
            actualResult = True
    self.assertEqual(expectedResult, actualResult)

def test100_920_bedTemp(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.BED_TEMP_HEADER1, self.GCODE_INPUT)
    if len(m) == 0:
        actualResult = True
    else:
        for element in m:
            current = float(element)
            if current != 0 and current != self.BED_TEMP_VAR:
                actualResult = "Bed Temp Error: value(" + str(current) + ") != value("
                            + str(self.BED_TEMP_VAR) + ")"
                break
            else:
                actualResult = True
```

```
    self.assertEqual(expectedResult, actualResult)

def test100_921_bedTemp(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.BED_TEMP_HEADER2, self.GCODE_INPUT)
    if len(m) == 0:
        actualResult = True
    else:
        for element in m:
            current = float(element)
            if current != 0 and current != self.BED_TEMP_VAR:
                actualResult = "Bed Temp Error: value(" + str(current) + ") != value("
                               + str(self.BED_TEMP_VAR) + ")"
                break
            else:
                actualResult = True
    self.assertEqual(expectedResult, actualResult)

def test100_930_bedTempMax(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.BED_TEMP_HEADER1, self.GCODE_INPUT)
    if len(m) == 0:
        actualResult = True
    else:
        for element in m:
            current = float(element)
            if current > self.MAX_BED_TEMP_VAR:
                actualResult = "Bed Temp Error: value(" + str(current) + ") > bounds("
                               + str(self.MAX_BED_TEMP_VAR) + ")"
                break
            else:
                actualResult = True
    self.assertEqual(expectedResult, actualResult)

def test100_931_bedTempMax(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.BED_TEMP_HEADER2, self.GCODE_INPUT)
    if len(m) == 0:
        actualResult = True
    else:
        for element in m:
            current = float(element)
            if current > self.MAX_BED_TEMP_VAR:
```

```
                actualResult = "Bed Temp Error: value(" + str(current) + ") > bounds("
                                + str(self.MAX_BED_TEMP_VAR) + ")"
                break
            else:
                actualResult = True
    self.assertEqual(expectedResult, actualResult)

def test200_900_fanNeverEngaged(self):
    expectedResult = True
    actualResult = False
    m = re.search(self.FAN_HEADER, self.GCODE_INPUT)
    actualResult = True if m != None else "fan never engaged"
    self.assertEqual(expectedResult, actualResult)

def test200_910_fan(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.FAN_HEADER, self.GCODE_INPUT)
    if len(m) == 0:
        actualResult = True
    else:
        for element in m:
            if ';' in element[0]:
                continue
            current = float(element[1])
            if current != 0 and current != self.FAN_VAR:
                actualResult = "Fan Value Error: value(" + str(current)
                                + ") != value(" + str(self.FAN_VAR) + ")"
                break
            else:
                actualResult = True
    self.assertEqual(expectedResult, actualResult)

def test200_920_fanMax(self):
    expectedResult = True
    actualResult = False
    m = re.findall(self.FAN_HEADER, self.GCODE_INPUT)
    if len(m) == 0:
        actualResult = True
    else:
        for element in m:
            if ';' in element[0]:
                continue
            current = float(element[1])
            if current > self.MAX_FAN_VAR:
                actualResult = "Fan Value Error: value(" + str(current)
```

```
                                    + ") > max value(" + str(self.FAN_VAR) + ")"
                    break
                else:
                    actualResult = True
        self.assertEqual(expectedResult, actualResult)

    def test300_900_layerHeight(self):
        expectedResult = True
        actualResult = False
        header = ''
        skip = False
        if self.SLICER_MATTER_SLICE in self.GCODE_INPUT:
            header = self.LAYER_HEIGHT_HEADER1
        elif self.SLICER_SLIC3R in self.GCODE_INPUT:
            header = self.LAYER_HEIGHT_HEADER2
        else:
            actualResult = "Unknown slicer. Unable to determine layer height"
            skip = True

        if skip == False:
            m = re.findall(header, self.GCODE_INPUT)
            mLength = len(m)
            previous = next_ = None
            for index, element in enumerate(m):
                current = float(element)
                if index < 1:
                    # skip because 1st layer height may be different
                    continue
                if index < (mLength -1):
                    next_ = float(m[index + 1])
                    diff2 = round(next_ - current, 3)
                    if diff2 != 0.0 and diff2 != self.LAYER_HEIGHT:
                        actualResult = "layer height error: value(" + str(next_) + ") != value("
                                       + str(self.LAYER_HEIGHT) + ")"
                        break
                actualResult = True
        self.assertEqual(expectedResult, actualResult)

    def test500_900_infill(self):
        expectedResult = True
        actualResult = False
        m = re.search(self.SLICER_MATTER_SLICE, self.GCODE_INPUT)
        n = re.search(self.SLICER_SLIC3R, self.GCODE_INPUT)
        if m != None:
            o = re.search(self.INFILL_HEADER1, self.GCODE_INPUT)
            infill1 = float(o.group(1))
```

```
        if infill1 != self.INFILL_VAR:
            actualResult = "Infill Error: value(" + str(infill1) + ") != value("
                           + str(self.INFILL_VAR) + ")"
        else:
            actualResult = True
    elif n != None:
        p = re.search(self.INFILL_HEADER2, self.GCODE_INPUT)
        infill2 = float(p.group(1))
        if infill2 != self.INFILL_VAR:
            actualResult = "Infill Error: value(" + str(infill2) + ") != value("
                           + str(self.INFILL_VAR) + ")"
        else:
            actualResult = True
    else:
        actualResult = "Unknown slicer. Unable to determine infill density"
    self.assertEqual(expectedResult, actualResult)

def test500_910_infillMax(self):
    expectedResult = True
    actualResult = False
    m = re.search(self.SLICER_MATTER_SLICE, self.GCODE_INPUT)
    n = re.search(self.SLICER_SLIC3R, self.GCODE_INPUT)
    if m != None:
        o = re.search(self.INFILL_HEADER1, self.GCODE_INPUT)
        infill1 = float(o.group(1))
        if infill1 > self.MAX_INFILL_VAR:
            actualResult = "Infill Error: value(" + str(infill1) + ") > value("
                           + str(self.MAX_INFILL_VAR) + ")"
        else:
            actualResult = True
    elif n != None:
        p = re.search(self.INFILL_HEADER2, self.GCODE_INPUT)
        infill2 = float(p.group(1))
        if infill2 > self.MAX_INFILL_VAR:
            actualResult = "Infill Error: value(" + str(infill2) + ") > value("
                           + str(self.MAX_INFILL_VAR) + ")"
        else:
            actualResult = True
    else:
        actualResult = "Unknown slicer. Unable to determine infill density"
    self.assertEqual(expectedResult, actualResult)

def test500_920_infillMin(self):
    expectedResult = True
    actualResult = False
    m = re.search(self.SLICER_MATTER_SLICE, self.GCODE_INPUT)
```

```
    n = re.search(self.SLICER_SLIC3R, self.GCODE_INPUT)
    if m != None:
        o = re.search(self.INFILL_HEADER1, self.GCODE_INPUT)
        infill1 = float(o.group(1))
        if infill1 < self.MIN_INFILL_VAR:
            actualResult = "Infill Error: value(" + str(infill1) + ") < value("
                           + str(self.MIN_INFILL_VAR) + ")"
        else:
            actualResult = True
    elif n != None:
        p = re.search(self.INFILL_HEADER2, self.GCODE_INPUT)
        infill2 = float(p.group(1))
        if infill2 < self.MIN_INFILL_VAR:
            actualResult = "Infill Error: value(" + str(infill2) + ") < value("
                           + str(self.MIN_INFILL_VAR) + ")"
        else:
            actualResult = True
    else:
        actualResult = "Unknown slicer. Unable to determine infill density"
    self.assertEqual(expectedResult, actualResult)

def test600_900_exceedsMaxXSize(self):
    expectedResult = False
    actualResult = True

    m = re.findall(self.X_SIZE_HEADER, self.GCODE_INPUT)
    elementX = None
    for index, element in enumerate(m):
        elementX = element
        if float(element) > self.MAX_X_SIZE:
            actualResult = " X value(" + str(elementX) + ") > X-axis bounds("
                           + str(self.MAX_X_SIZE) + ")"
            break
        elif index == len(m)-1:
            actualResult = False
    self.assertEqual(expectedResult, actualResult)

def test600_910_exceedsMinXSize(self):
    expectedResult = False
    actualResult = True
    m = re.findall(self.X_SIZE_HEADER, self.GCODE_INPUT)
    elementX = None
    for index, element in enumerate(m):
        elementX = element
        if float(element) < self.MIN_X_SIZE:
            if index == 0: continue
```

```
            actualResult = " X value(" + str(elementX) + ") < X-axis bounds("
                           + str(self.MIN_X_SIZE) + ")"
            break
        elif index == len(m)-1:
            actualResult = False
    self.assertEqual(expectedResult, actualResult)

def test600_920_exceedsMaxYSize(self):
    expectedResult = False
    actualResult = True
    m = re.findall(self.Y_SIZE_HEADER, self.GCODE_INPUT)
    elementY = None
    for index, element in enumerate(m):
        elementY = element
        if float(element) > self.MAX_Y_SIZE:
            actualResult = " Y value(" + str(elementY) + ") > Y-axis bounds("
                           + str(self.MAX_Y_SIZE) + ")"
            break
        elif index == len(m)-1:
            actualResult = False
    self.assertEqual(expectedResult, actualResult)

def test600_930_exceedsMinYSize(self):
    expectedResult = False
    actualResult = True
    m = re.findall(self.Y_SIZE_HEADER, self.GCODE_INPUT)
    elementY = None
    for index, element in enumerate(m):
        elementY = element
        if float(element) < self.MIN_Y_SIZE:
            if index == 0: continue
            actualResult = " Y value(" + str(elementY) + ") < Y-axis bounds("
                           + str(self.MIN_Y_SIZE) + ")"
            break
        elif index == len(m)-1:
            actualResult = False
    self.assertEqual(expectedResult, actualResult)

def test600_940_exceedsMaxZSize(self):
    expectedResult = False
    actualResult = True
    m = re.findall(self.Z_SIZE_HEADER, self.GCODE_INPUT)
    elementZ = None
    for index, element in enumerate(m):
        elementZ = element
        if float(element) > self.MAX_Z_SIZE:
```

```
                actualResult = "Z value(" + str(elementZ) + ") > Z-axis bounds("
                               + str(self.MAX_Z_SIZE) + ")"
                break
            elif index == len(m)-1:
                actualResult = False
        self.assertEqual(expectedResult, actualResult)

    def test600_950_exceedsMinZSize(self):
        expectedResult = False
        actualResult = True
        m = re.findall(self.Z_SIZE_HEADER, self.GCODE_INPUT)
        elementZ = None
        for index, element in enumerate(m):
            elementZ = element
            if float(element) < self.MIN_Z_SIZE:
                if index == 0: continue
                actualResult = "Z value(" + str(elementZ) + ") < Z-axis bounds("
                               + str(self.MIN_Z_SIZE) + ")"
                break
            elif index == len(m)-1:
                actualResult = False
        self.assertEqual(expectedResult, actualResult)

# THE FOLLOWING TEST CASES ARE FOR CMB TOOLPATH FILES

class CMB_TestCases(unittest.TestCase):
    CMB_PATH = ""
    CMB_INPUT = ""
    SKIP_TEST = None
    CMB_MAX_X_SIZE = p.CMB_MAX_X_SIZE
    CMB_MAX_Y_SIZE = p.CMB_MAX_Y_SIZE
    CMB_MAX_Z_SIZE = p.CMB_MAX_Z_SIZE
    expectedResult = False
    actualResult = True

    @classmethod
    def setUpClass(cls):
        if ".cmb" in cls.CMB_PATH:
            cls.SKIP_TEST = False
            with open(cls.CMB_PATH, 'rb') as f:
                cls.CMB_INPUT = f.read().hex()
        else:
            cls.SKIP_TEST = True

    def setUp(self):
        if self.SKIP_TEST == True:
```

```
        self.skipTest("Not CMB file")
    self.expectedResult = False
    self.actualResult = True

def test700_900_cmb_exceedsMaxXSize(self):
    index_ = 0
    part_max_X = ""
    temp = ""
    hexList = []
    for line in self.CMB_INPUT:
        for element in line:
            temp += element
            if (index_ + 1) % 2 == 0:
                hexList.append(temp)
                temp = ""
            index_ += 1
            if index_ == 200:
                break
    for index, value in enumerate(hexList):
        if index >= 50 and index <= 53:
            part_max_X = value + part_max_X
        elif index > 53:
            break
        index += 1

    floatMaxX_in = ieeeConverter.hex2Float(part_max_X)
    floatMaxX_mm = floatMaxX_in / 0.0393700787
    if floatMaxX_mm > self.CMB_MAX_X_SIZE:
        self.actualResult = " X value(" + str(floatMaxX_mm) + ") > X-axis bounds("
                            + str(self.CMB_MAX_X_SIZE) + ")"
    else:
        self.actualResult = False
    self.assertEqual(self.expectedResult, self.actualResult)

def test700_910_cmb_exceedsMaxYSize(self):
    index_ = 0
    part_max_Y = ""
    temp = ""
    hexList = []
    for line in self.CMB_INPUT:
        for element in line:
            temp += element
            if (index_ + 1) % 2 == 0:
                hexList.append(temp)
                temp = ""
            index_ += 1
```

```
            if index_ == 200:
                break
    for index, value in enumerate(hexList):
        if index >= 54 and index <= 57:
            part_max_Y = value + part_max_Y
        elif index > 58:
            break
        index += 1
    floatMaxY_in = ieeeConverter.hex2Float(part_max_Y)
    floatMaxY_mm = floatMaxY_in / 0.0393700787
    if floatMaxY_mm > self.CMB_MAX_Y_SIZE:
        self.actualResult = " Y value(" + str(floatMaxY_mm) + ") > Y-axis bounds("
                            + str(self.CMB_MAX_Y_SIZE) + ")"
    else:
        self.actualResult = False
    self.assertEqual(self.expectedResult, self.actualResult)

def test700_920_cmb_exceedsMaxZSize(self):
    index_ = 0
    part_max_Z = ""
    temp = ""
    hexList = []
    for line in self.CMB_INPUT:
        for element in line:
            temp += element
            if (index_ + 1) % 2 == 0:
                hexList.append(temp)
                temp = ""
            index_ += 1
            if index_ == 200:
                break
    for index, value in enumerate(hexList):
        if index >= 58 and index <= 61:
            part_max_Z = value + part_max_Z
        elif index > 62:
            break
        index += 1

    floatMaxZ_in = ieeeConverter.hex2Float(part_max_Z)
    floatMaxZ_mm = floatMaxZ_in / 0.0393700787
    floatMaxZ_mm = round(floatMaxZ_mm)
    if floatMaxZ_mm > self.CMB_MAX_Z_SIZE:
        self.actualResult = " Z value(" + str(floatMaxZ_mm) + ") > Z-axis bounds("
                            + str(self.CMB_MAX_Z_SIZE) + ")"
    else:
        self.actualResult = False
```

```
        self.assertEqual(self.expectedResult, self.actualResult)

# FOLLOWING ALLOWS TO PASS G-CODE/CMB FILE ARGUMENT IN TERMINAL

if __name__ == '__main__':
    if len(sys.argv) > 1:
        filePath = sys.argv.pop()
        GCode_TestCases.GCODE_PATH = filePath
        CMB_TestCases.CMB_PATH = filePath
    unittest.main()
```

APPENDIX B: TEST RESULTS

B.1 BENIGN MODEL

```
has@ubuntu:~/Documents/Thesis/code/config$ python3 gCTests.py /home/ha
s/Documents/Thesis/code/stlFiles/MaliciousFiles/BenModel.gcode
sss..................
----------------------------------------------------------------------
Ran 22 tests in 0.013s

OK (skipped=3)
```

B.2 MALICIOUS MODEL

```
has@ubuntu:~/Documents/Thesis/code/config$ python3 gCTests.py /home/ha
s/Documents/Thesis/code/stlFiles/MaliciousFiles/MalModel.gcode
sss.............F.F...
======================================================================
FAIL: test600_900_exceedsMaxXSize (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 702, in test600_900_exceedsMaxXSize
    self.assertEqual(expectedResult, actualResult)
AssertionError: False != ' X value(200.537) > X-axis bounds(200.0)'

======================================================================
FAIL: test600_920_exceedsMaxYSize (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 768, in test600_920_exceedsMaxYSize
    self.assertEqual(expectedResult, actualResult)
AssertionError: False != ' Y value(223.878) > Y-axis bounds(200.0)'

----------------------------------------------------------------------
Ran 22 tests in 0.665s

FAILED (failures=2, skipped=3)
```

B.3 BENIGN STL

```
has@ubuntu:~/Documents/Thesis/code/config$ python3 gCTests.py /home/ha
s/Documents/Thesis/code/stlFiles/MaliciousFiles/BenSTL.gcode
sss..................
----------------------------------------------------------------------
Ran 22 tests in 0.023s

OK (skipped=3)
```

B.4 MALICIOUS STL

```
has@ubuntu:~/Documents/Thesis/code/config$ python3 gCTests.py /home/ha
s/Documents/Thesis/code/stlFiles/MaliciousFiles/MalSTL.gcode
sss.............F..F..
======================================================================
FAIL: test600_900_exceedsMaxXSize (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 702, in test600_900_exceedsMaxXSize
    self.assertEqual(expectedResult, actualResult)
AssertionError: False != ' X value(221.409) > X-axis bounds(200.0)'

======================================================================
FAIL: test600_930_exceedsMinYSize (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 800, in test600_930_exceedsMinYSize
    self.assertEqual(expectedResult, actualResult)
AssertionError: False != ' Y value(-0.120) < Y-axis bounds(0.1)'

----------------------------------------------------------------------
Ran 22 tests in 0.366s

FAILED (failures=2, skipped=3)
```

B.5 BENIGN G-CODE

```
has@ubuntu:~/Documents/Thesis/code/config$ python3 gCTests.py /home/ha
s/Documents/Thesis/code/stlFiles/MaliciousFiles/BenGCode.gcode
sss........F..........
======================================================================
FAIL: test200_900_fanNeverEngaged (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 330, in test200_900_fanNeverEngaged
    self.assertEqual(expectedResult, actualResult)
AssertionError: True != 'fan never engaged'

----------------------------------------------------------------------
Ran 22 tests in 0.135s

FAILED (failures=1, skipped=3)
```

B.6 MALICIOUS G-CODE

```
has@ubuntu:~/Documents/Thesis/code/config$ python3 gCTests.py /home/has/Do
cuments/Thesis/code/stlFiles/MaliciousFiles/MalGCode.gcode
sss.F.F.F.FF....F.F.F.
======================================================================
FAIL: test100_901_maxTemp (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 111, in test100_901_maxTemp
    self.assertEqual(expectedResult, actualResult)
AssertionError: True != 'Extruder Temp Error: value(1210.0) > bounds(220)'

======================================================================
FAIL: test100_911_temp (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 171, in test100_911_temp
    self.assertEqual(expectedResult, actualResult)
AssertionError: True != 'Extruder Temp Error: value(1210.0) != value(210)'

======================================================================
FAIL: test100_921_bedTemp (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 237, in test100_921_bedTemp
    self.assertEqual(expectedResult, actualResult)
AssertionError: True != 'Bed Temp Error: value(1115.0) != value(35)'
```

```
======================================================================
FAIL: test100_931_bedTempMax (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 305, in test100_931_bedTempMax
    self.assertEqual(expectedResult, actualResult)
AssertionError: True != 'Bed Temp Error: value(1115.0) > bounds(65)'

======================================================================
FAIL: test200_900_fanNeverEngaged (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 330, in test200_900_fanNeverEngaged
    self.assertEqual(expectedResult, actualResult)
AssertionError: True != 'fan never engaged'

======================================================================
FAIL: test600_900_exceedsMaxXSize (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 702, in test600_900_exceedsMaxXSize
    self.assertEqual(expectedResult, actualResult)
AssertionError: False != ' X value(1110) > X-axis bounds(200.0)'

======================================================================
FAIL: test600_920_exceedsMaxYSize (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 768, in test600_920_exceedsMaxYSize
    self.assertEqual(expectedResult, actualResult)
AssertionError: False != ' Y value(1110) > Y-axis bounds(200.0)'

======================================================================
FAIL: test600_940_exceedsMaxZSize (__main__.V3DPTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 832, in test600_940_exceedsMaxZSize
    self.assertEqual(expectedResult, actualResult)
AssertionError: False != 'Z value(1110.8) > Z-axis bounds(200.0)'

----------------------------------------------------------------------
Ran 22 tests in 0.117s

FAILED (failures=8, skipped=3)
```

B.7 BENIGN CMB

```
has@ubuntu:~/Documents/Thesis/code/config$ python3 gCTests.py /home/ha
s/Documents/Thesis/code/stlFiles/MaliciousFiles/BenCMB.cmb
...sssssssssssssssss
----------------------------------------------------------------------
Ran 22 tests in 0.091s

OK (skipped=19)
```

B.8 DEFECTIVE CMB

```
has@ubuntu:~/Documents/Thesis/code/config$ python3 gCTests.py /home/ha
s/Documents/Thesis/code/stlFiles/MaliciousFiles/MalCMB.cmb
FFFsssssssssssssssss
======================================================================
FAIL: test700_900_cmb_exceedsMaxXSize (__main__.CMBTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 939, in test700_900_cmb_exceedsMaxXSize
    self.assertEqual(self.expectedResult, self.actualResult)
AssertionError: False != ' X value(203.19997598393635) > X-axis bounds
(200.0)'

======================================================================
FAIL: test700_910_cmb_exceedsMaxYSize (__main__.CMBTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 1001, in test700_910_cmb_exceedsMaxYSize
    self.assertEqual(self.expectedResult, self.actualResult)
AssertionError: False != ' Y value(51657.56631245634) > Y-axis bounds(
200.0)'

======================================================================
FAIL: test700_920_cmb_exceedsMaxZSize (__main__.CMBTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "gCTests.py", line 1064, in test700_920_cmb_exceedsMaxZSize
    self.assertEqual(self.expectedResult, self.actualResult)
AssertionError: False != ' Z value(25089759334242889909876697734182485
36064) > Z-axis bounds(200.0)'

----------------------------------------------------------------------
Ran 22 tests in 0.095s

FAILED (failures=3, skipped=19)
```

ACKNOWLEDGEMENTS

First and foremost, I want to thank my advisor Dr. David Umphress from the computer science and software engineering department at Auburn University, Alabama. It has been an honor to work under him as his graduate research assistant and lab administrator. My entire life, up until this point, I have been a student. I have had many professors, good and bad. But I have never met someone that can match the deep love of teaching that Dr. Umphress has.

He is a shining example of what a true leader should be and someone I look up too. He is genuinely one of the most amazing people I have had the privilege of working under and consider him a friend.

While there is no doubt that I would not be where I am in life if I had not met Dr. Umphress, I would also not be the person I am today without him. The lessons and experience I've learned under his leadership will forever encourage me to "do cool things."

I would also like to thank Dr. Louis Payton, who helped me discover my love for 3D printing and giving me the tools to make it possible. I recognize and appreciate the support of my defense committee members: Dr. Mark Yampolskiy and Dr. Drew Springall, for contributing their time, interest, and invaluable advice.

I would also like to acknowledge a few friends and colleagues: Bhargav Joshi, Charlie Harper, Neda Topuz, Hamza Alkofahi, Demarcus Campbell, Lakshmi Krishnaprasad, and Jordan Sosnowski. They all inspire me to push past my limits and are all genuinely exceptional people. It

is both an honor and a privilege to be their friends. I do not doubt that they will all do amazing things with their lives.

I would also like to thank my school, Auburn University. The resources, people, and experiences here, in my opinion, cannot be matched anywhere else in the world.

Last but not least, I would like to thank my family. My mother and father, Steve and Susan, for their financial support through college and making me live up to my potential. My wife, Morgan, has always stood by my side and supported me. She has given me unconditional love and support and challenges me to be a better person every day. I consider myself am a lucky man to have such deep and loving relationships from my many friends and family.

Kyle Ash Searle
Auburn University
Aug 2020

VITA

KYLE ASH SEARLE
Cell: +1 (334)-399-2285
Email: a.searle274@gmail.com

Academic Record

- **Auburn University (MS with thesis), AL, USA** (Jan 2019 – Aug 2020)
  Computer Science and Software Engineering (Awarded Degree)
- **Auburn University (Undergraduate), AL, USA** (Aug 2014 – Dec 2019)
- Computer Science and Software Engineering
- **Auburn University at Montgomery** (Aug 2013 – May 2014)
  Computer Science and Software Engineering

Work Experience

- **Auburn Cyber Research Center (ACRC), Auburn University, AL** (May 2018– present)
  Position – Lab Administrator and Graduate/Undergraduate Research Assistant
  Description – I worked with Dr. David Umphress (ACRC Director) to maintain and improve the ACRC labs. I was responsible for managing and maintaining ACRC's Stratasys 3D printer. Performing tasks such as changing out filaments, troubleshooting, printer supply inventory, training coworkers how to use the printer, managing the printer's chemical bath and training coworkers safety procedures, maintaining safety logs, maintaining safety signage, CAD drawing and printing anything that anyone needs (as long as it's cyber-related), maintaining ACRC inventory, optimizing lab efficiency, research and put together appraisals when seeking to buy new equipment for labs, install or set up new equipment, help professors with equipment/classes (when related to Dr. Umphress or ACRC), and help with anything Ethical Hacking Club needs. I also performed research on various projects necessary for the lab (setting up a VPN server, SAMBA server, and NAT switches.)
- **Online Commerce Group, Montgomery, AL** (May 2016 – Aug 2017)
  Position – Full stack developer / I.T. / Helpdesk Intern
  Description –I worked with Gerry Monroe, Joel Hutchinson, and Trey Harvell Sr. working on backend/frontend development, research, IT administration, help desk, and hardware maintenance.
- **Rheem Manufacturing (Water Heating Division), Montgomery, AL** (May 2014 – Jan 2017)
- *Position* – I.T. COOP Intern
- *Description* –I worked with Ashley Boyd, Perry Warren, and many others performing I.T. administration, helpdesk, documentation, research, and computer replacements.

Major Academic Projects Accomplished (Jan 2019 – Aug 2020)

- **Toolpath Static Analysis Testing for Additive Manufacturing**
  Applying static analysis testing to 3D printers to detect malicious and non-malicious defects in toolpath files before the printing process.

Area of Interest

- 3D Printing (Additive Manufacturing)
- Cyber Security
- Full-Stack Development
- Computer Networks
- Software Process
- Embedded Systems