Needle in a Haystack: Detecting Subtle Malicious Edits to Additive Manufacturing G-code Files

Caleb Beckwith, Harsh Sankar Naicker *(Member, IEEE)*, Svara Mehta, Viba R. Udupa *(Member, IEEE)*, Nghia Tri Nim, Varun Gadre, Hammond Pearce *(Member, IEEE)*, Gary Mac, Nikhil Gupta *(Senior Member, IEEE)*

***Abstract*—Increasing usage of Digital Manufacturing (DM) in safety-critical domains is increasing attention on the cybersecurity of the manufacturing process, as malicious third parties might aim to introduce defects in digital designs. In general, the DM process involves creating a digital object (as CAD files) before using a slicer program to convert the models into printing instructions (e.g. g-code) suitable for the target printer. As the g-code is an intermediate machine format, malicious edits may be difficult to detect, especially when the golden (original) models are not available to the manufacturer. In this work we aim to quantify this hypothesis through a red-team/blue-team case study, whereby the red-team aims to introduce subtle defects that would impact the properties (strengths) of the 3D printed parts, and the blue-team aims to detect these modifications in the absence of the golden models. The case study had two sets of models, the first with 180 designs (with 2 compromised using 2 methods) and the second with 4320 designs (with 60 compromised using 6 methods). Using statistical modelling and machine learning (ML), the blue-team was able to detect all the compromises in the first set of data, and 50 of the compromises in the second.**

I. Introduction

Digital Manufacturing (DM) is increasingly used in safety-critical applications across domains such as the aerospace, automotive, medical, and military sectors [1], [2]. Consequently, the potential target space for malicious third parties is increasing, with motivations for both espionage (information theft) and sabotage (compromising machines or model data).

One attractive target for malicious actors in this space is the `g-code` files which detail the machine instructions used to manufacture each part [3], [4]. These files may contain subtractive drill/mill commands in the case of CNC-style machines, or filament extrusion commands in the case of additive manufacturing (AM). The `g-code` files are often generated externally to the machines that execute them, and due to the lossy conversions in their generation, reverse-engineering the original model files for comparison is a difficult process [5]. As such, malicious edits to `g-code` may go unnoticed.

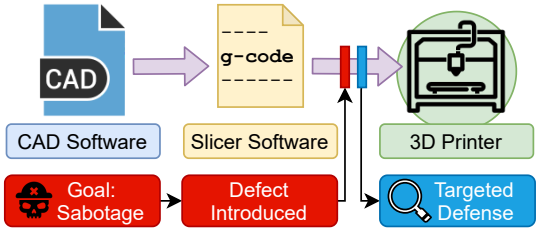

Fig. 1. The red-team sabotage the `g-code` instructions prior to printing. The blue-team aims to detect the compromised files.

In this work, we thus examine the design space for attacking and defending `g-code` through a red-team/blue-team exercise (as depicted in Fig. 1). Here, the red-team introduces defects into a sample set of `g-code` files and the blue-team detects the tampered files. The red-team focused on mechanisms to mutate `g-code` instructions to reduce part strength in small numbers of randomly chosen files (simulating a subtle attack). The blue-team then proposed and utilized statistical analysis and machine learning (ML) techniques as possible detection strategies before a blind evaluation. Overall, the red-team devised 6 different compromise methodologies, and the blue-team was able to detect 5 of these across two different datasets.

The rest of this paper is organized as follows. Section II covers the background and related work. Section III then describes the methodologies used by the red and blue teams. Section IV evaluates these methodologies and discusses the limitations of the approaches. Section V then concludes.

II. Background and Related Work

Before 3D CAD model files are converted to `g-code` through the use of a slicer program, they must usually be converted to stereolithography `STL` files—imperfect representations of the original data, where curved surfaces are discretized into tessellated triangles. When converted to `g-code`, these surface triangles are then encoded as lines of filament. Both conversions lose original design information and present opportunities for malicious actors to introduce defects into parts that may go unnoticed [6].

As such, just as CAD and STL files may be potential attack targets [7], `g-code` files must also be considered vulnerable. Example attacks have already included compromising the USB communication between PC and printer [3] and altering the 3D printer firmware responsible for interpreting the incoming `g-code` [8], [9].

C. Beckwith is with the Dept. of Mechanical Engineering, NYC College of Technology, Brooklyn, NY 11201 USA.

H. Naicker is with the School of Electronics Engineering, Vellore Institute of Technology, Kelambakkam, Chennai, Tamil Nadu 600127, India.

S. Mehta is with the Dept. of Mechanical Engineering, Indian Institute of Technology Goa, Ponda, Goa 403401, India.

V. R. Udupa is with the Dept. of Mechanical Engineering, National Institute of Technology Surathkal, Mangalore, Karnataka 575025, India.

N. Tri Nim is with the Science Division, New York University Abu Dhabi, Abu Dhabi, UAE.

V. Gadre is with the Dept. of Mechanical Engineering, Indian Institute of Technology Kanpur, Kanpur, Uttar Pradesh 208016, India.

H. Pearce (corresponding author, e-mail hammond.pearce@nyu.edu) is with the Dept. of Electrical and Computer Engineering, New York University, Brooklyn NY 11201 USA.

G. Mac and N. Gupta are with the Dept. of Mechanical and Aerospace Engineering, New York University, Brooklyn NY 11201 USA.

While there are proposals for analyzing the g-code to detect introduced defects through finite element analysis simulations [5], the process for this analysis is computationally intensive and difficult, requiring derivation of the original CAD models. As a result, other approaches have been investigated, including those from the information security domain (e.g. cryptographically signing production files to detect tampering [10]). However, depending on the attack model, tampering of the g-code data could occur prior to the signing of the files or after a verification has occurred.

Likewise, detecting compromised g-code has also been suggested via *side-channel analysis*: while the initial focus has been on demonstrating information leakage [11], [12], these same channels may be used for monitoring of part defect and malicious edits [13], [14]. Similarly, using a vision-based ML approach to detect defects during print has also been proposed [15]. However, these run-time approaches may only be performed during the printing process, at which point valuable resources (filament, machine time) have already been expended. As such, in this work we propose to detect defects in the g-code prior to the printing process.

III. Methodology

In this work we consider compromised g-code in the absence of the original CAD model files for validation. This threat vector is possible in manufacturing-as-a-service (MaaS), where only the production files may be provided to the manufacturer. The compromise could happen at any stage before manufacturing, e.g. by a malicious slicer program or from modifications by a third party transiting the data. A red-team/blue-team exercise was thus conducted with the two teams isolated from one another. The red-team, consisting of the latter three authors of the paper, constructed two datasets D1 and D2 which contained many 'good' (non-compromised) g-code files and a small number of 'bad' (compromised) g-code files. The blue-team, consisting of the first six authors of the paper, were tasked with isolating the 'bad' models.

A. The datasets

To generate each dataset, we take a single STL CAD model file and perform a rotate-then-slice using Ultimaker Cura. By using a number of distinct rotations with respect to the build plate, each generated g-code file is unique (as the g-code will be optimized for that particular rotation). Dataset D1 consists of a tensile test specimen (Fig. 2) like that in [8]. The CAD model is rotate-then-sliced by 1° 180 times in the Y direction, with slicer settings (skirt, 0.4mm nozzle, grid infill pattern, infill line distance 2mm). Dataset D2 consists of a bracket (Fig. 3) which is rotated 4320 times in total (rotate-then-slice by 0.25° 1440 times for each face against the print bed), with slicer settings (skirt, 0.4mm nozzle, cubic infill pattern, infill line distance 4mm).

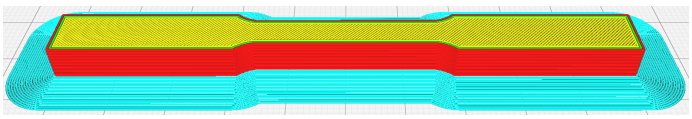

Fig. 2. The control specimen for Dataset D1 viewed in Ultimaker Cura.

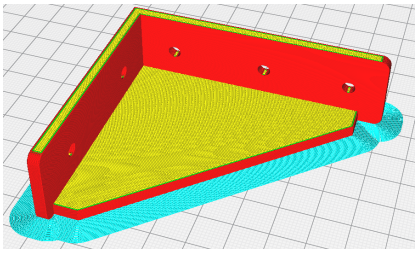

Fig. 3. The control specimen for Dataset D2 viewed in Ultimaker Cura.

TABLE I
RED-TEAM DATASET VULNERABILITY DESIGNS. 'N.D1'/'N.D2' INDICATES HOW MANY G-CODE FILES WERE COMPROMISED USING THIS METHOD IN THE 180-MODEL D1 DATASET / 4320-MODEL D2 DATASET RESPECTIVELY. 'RANGE' INDICATES HOW MUCH OF THE G-CODE FILE WAS ALTERED - EITHER THE MIDDLE 50 %, OR THE ENTIRE 100 %.

ID	n.D1?	n.D2?	Range	Description
1	2	10	50 %	Origin in [8]. Coverts every 4th G1 command to G0.
2		10	50 %	Converts every 4th G1 command to G0, and adds a G1 'blob' extrusion.
3		10	100 %	Origin in [8]. Reduces extrusion globally by 50%.
4		10	50 %	Every 4th G1 command has extrusion value set to previous extrusion.
5		10	50 %	Every 4th G1 command has extrusion value set to previous extrusion + 0.0001.
6		10	50 %	Deletes every 4th G1 line with no replacement.

B. Red-team: Vulnerability insertion

In this work, the compromises were designed similarly to the methodology for reducing the strength of printed models presented in [8]. Six different vulnerability strategies were devised, and are listed in Table I. They function primarily by altering the g-code G1 (linear move and extrude) command to introduce gaps (voids) in the printing process by subverting the absolute frame of reference used within common g-code. As an example, consider three back-to-back commands (G1 X1 Y1 E1), (G1 X5 Y1 E2), and (G1 X5 Y5 E3). The total extruded material is 1mm after the first command, 2mm after the second (it extrudes 1mm), and 3mm after the third (it also extrudes 1mm). If the Trojan alters the second command to (G0 X5 Y1), no material is extruded along the same route. Instead, the third command will now extrude 2mm, such that the total material used remains 3mm.

As such, ID1 and ID2 convert some G1 commands to G0 (linear move) commands, dropping the extrusions. ID2 adds an additional 'blob' extrusion (keeping the head in place and

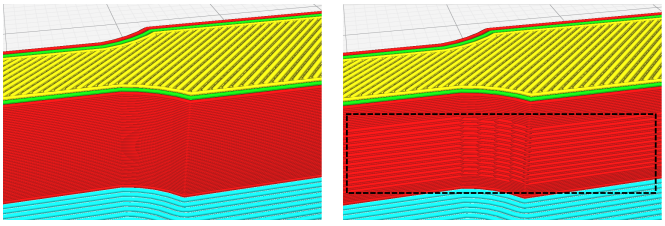

Fig. 4. Comparison of good (left) verses compromised (right) g-code in Dataset 1 by method ID1 when viewed in Ultimaker Cura. The dotted black box indicates the area with altered print commands.

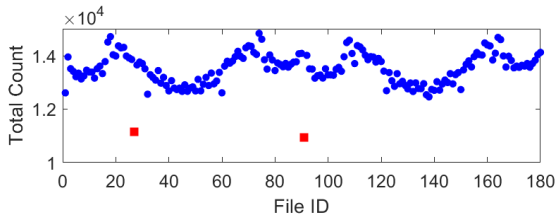

Fig. 5. Distribution of `G1` commands in Dataset D1. Outliers are in red.

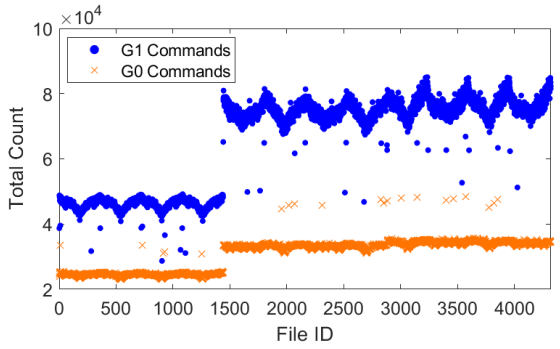

Fig. 6. Distribution of `G1` and `G0` commands in Dataset D2.

depositing the missing material) at the end of the original move. ID3 changes all `G1` commands to use 50 % less filament. ID4 and ID5 set some of the `G1` extrusions to negligible values, essentially making them `G0` commands. Finally, ID6 simply deletes some of the `G1` commands. Fig. 4 depicts ID1 on D1—with the middle 50 % of the part defective (lacking extrusions).

C. Blue-team: Vulnerability detection

1) Statistical Analysis: The first strategy utilized by the blue-team was to perform a statistical feature extraction of the `g-code` files. These included the number of layers, the boundaries of the values of `X`, `Y`, and `Z` commands, and the material length extruded. Given that the two most common `g-code` commands are `G0` and `G1`, the blue-team reasoned that any compromise would likely effect the statistical properties of these commands, and so they also included the number of the `G0` / `G1` commands. These features were then examined using `matplotlib` and `seaborn` in Python to visually identify outliers. As can be seen in Fig. 5, this strategy appeared to identify the two files with anomalous counts. Manual inspection of the `g-code` in Ultimaker Cura (e.g. Fig. 4) then confirmed these files were defective. This 'Basic Statistical Analysis' strategy was thus attempted over the larger Dataset D2. However, again using Cura, the blue-team determined that this single-feature approach was now returning false positives.

As such, the outlier detection was expanded to include both `G1` and `G0`, as depicted in Fig. 6. From here it can be concluded that files may have either 'too few' `G0` commands, 'too many' `G1` commands, or both. From this, it was possible to identify a total of 30 files that seemed to have defects when viewed in Cura. A further anomaly was identified when considering the extrusion values `E`. The vast majority of the files had 5 decimal points, however, some files had `E` commands with more or fewer decimals. These files were thus determined to be corrupted creating two distinct categories:

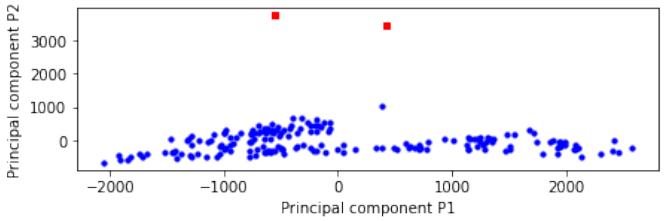

Fig. 7. Dataset D1 outlier detection using Principle Component Analysis and Agglomerative Clustering. Outliers in red.

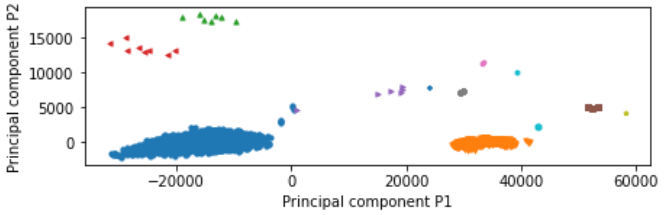

Fig. 8. Dataset D2 outlier detection using Principle Component Analysis and Mean Shift Algorithm. The two major clusters are the blue circles and orange down-arrows. Other cluster colors/shapes indicate outlier groups.

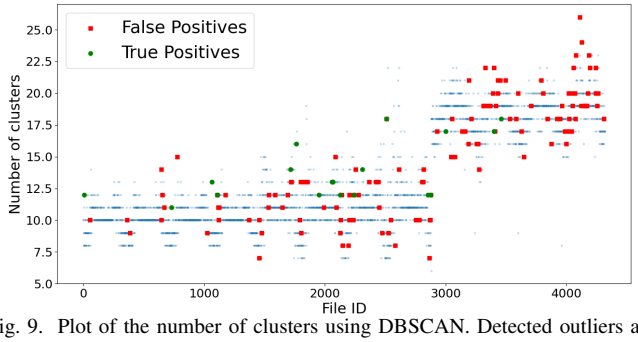

Fig. 9. Plot of the number of clusters using DBSCAN. Detected outliers are marked with green circles/red squares depending on their final correctness (not provided to blue-team).

Files that were given extra extrusion distance; and files that were given less extrusion distance. For each category, 10 more files were found for a total of 50 files. The blue-team termed this complete analysis the 'Combined Statistical Analysis'.

2) Machine learning (ML) based approach: The first ML-based method examined for identifying outliers involved using Principal Component Analysis (PCA) followed by various clustering algorithms. Here, the `g-code` files were converted to data frames using the `gcodeparser` package [16]. These data frames were then analyzed to produce a count of each command type along with the total count of lines present in the file. PCA was then utilized to reduce the dimensions of each data frame from 11 to 2. Following this, various clustering algorithms like agglomerative clustering, mean shift algorithm, etc. were used and scatter plots were obtained of the clusters for each dataset D1 (Fig. 7) and D2 (Fig. 8). Here, although the outliers from D1 were again identified, the D2 outliers were less consistent with the previously identified defective files.

An alternative approach was thus devised using clustering-based unsupervised learning, specifically, "density-based spatial clustering of applications with noise" (DBSCAN). Here, the earlier data frames were clustered in Python using `scikit`'s DBSCAN to produce Fig. 9. Here, outliers are

TABLE II
OVERALL DETECTION RESULTS

	Dataset D1				Dataset D2			
Method	**T.P.**	**F.P**	**T.N**	**F.N**	**T.P.**	**F.P.**	**T.N.**	**F.N.**
(Correct Value)	*2*	*0*	*178*	*0*	*60*	*0*	*4260*	*0*
Single Statistical Analysis	2	0	178	0	29	21	4260	10
Combined Statistical Analysis	2	0	178	0	50	0	4260	10
PCA and Clustering	2	0	178	0	28	7	4232	32
DBSCAN	1	0	178	1	35	7	4253	25

represented as spikes on the graph. However, while this method had some overlap with the results from the statistical approach, there were also other files identified.

IV. EVALUATION

The overall detection strategy results are presented in Table II, with each method paired with the detection results True Positive (T.P.) (i.e. defect present in `g-code` file and algorithm correctly detected this as an outlier), False Positive (F.P.) (i.e. no defect but wrongly identified as an outlier), True Negative (T.N.) (i.e. no defect and correctly identified as not an outlier), False Negative (F.N.) (i.e. defect present but wrongly identified as not an outlier) for each of the datasets D1 and D2. As can be seen, the different strategies proved largely successful at identifying the defects (outliers) in D1. However, none of the methods identified all defects in D2. The best approach, 'Combined Statistical Analysis', was able to detect all compromised files by all defect strategies other than those compromised by ID3 (which reduced extrusion globally by 50 %)—indeed, none of the detection strategies detected any files by ID3. This was a surprising result, as previous work [8] determined that this methodology of `g-code` compromise would be the most obvious when considering post-manufacturing checks. This is likely because the defect was global, and so none of the algorithms saw sudden changes in the files caused by the defect. Finally, while the machine-learning-based approaches show some promise, it appears they need further refinement before they will function as well as the 'Combined Statistical Analysis'.

Limitations: In this work the datasets were synthetically generated from the original models presented in Fig. 2 and 3. While the `g-code` for each individual rotation was quite distinct (e.g. Fig. 5 and 6), it is likely that the conceptual similarities between each model aided in the detection strategies. Further, our attack model assumed that a small minority of files would be compromised by the attackers. Future work could aim to evaluate both of these cases—(1), where datasets include models with unrelated geometries, and (2) some datasets where a majority or all files are compromised.

V. CONCLUSIONS

Automatically detecting defects in `g-code` is an important step towards the cybersecurity of Manufacturing-as-a-Service (MaaS) production processes. Here, models may be required to be validated without access to the original design files. In this work we approached this challenge as a red-team/blue-team exercise, and demonstrated how statistical and machine-learning-based approaches can identify faults. While the blue-team were able to identify most outliers (i.e. defective `g-code` files), the machine-learning-based approaches had reduced accuracy, indicating that further training data is required.

Future work in this area should focus on improving and refining the selected algorithms in conjunction with an expansion of the defect methodologies. In addition, it would be interesting to determine if there are any CAD features that might effect the overall success of the defect detection strategies.

ACKNOWLEDGMENT

This work was supported in part by National Science Foundation (NSF) SaTC DGE-1931724, NSF IRES OISE-1952479, and SecureAmerica Institute.

REFERENCES

[1] L. Nickels, "AM and aerospace: an ideal combination," *Metal Powder Report*, vol. 70, no. 6, pp. 300–303, Nov. 2015.

[2] S. K. Tiwari, S. Pande, S. Agrawal, and S. M. Bobade, "Selection of selective laser sintering materials for different applications," *Rapid Prototyping Journal*, vol. 21, no. 6, pp. 630–648, Jan. 2015.

[3] S. Moore, P. Armstrong, T. McDonald, and M. Yampolskiy, "Vulnerability analysis of desktop 3D printer software," in *2016 Resilience Week (RWS)*, Aug. 2016, pp. 46–51.

[4] N. Gupta, A. Tiwari, S. T. S. Bukkapatnam, and R. Karri, "Additive Manufacturing Cyber-Physical System: Supply Chain Cybersecurity and Risks," *IEEE Access*, vol. 8, pp. 47 322–47 333, 2020.

[5] N. G. Tsoutsos, H. Gamil, and M. Maniatakos, "Secure 3D Printing: Reconstructing and Validating Solid Geometries using Toolpath Reverse Engineering," in *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, ser. CPSS '17. New York, NY, USA: Association for Computing Machinery, Apr. 2017, pp. 15–20.

[6] F. Chen, G. Mac, and N. Gupta, "Security features embedded in computer aided design (CAD) solid models for additive manufacturing," *Materials & Design*, vol. 128, pp. 182–194, Aug. 2017.

[7] S. Belikovetsky, M. Yampolskiy, J. Toh, J. Gatlin, and Y. Elovici, "dr0wned – Cyber-Physical Attack with Additive Manufacturing," in *11th USENIX Workshop on Offensive Technologies (WOOT)*, 2017.

[8] H. Pearce, K. Yanamandra, N. Gupta, and R. Karri, "FLAW3D: A Trojan-based Cyber Attack on the Physical Outcomes of Additive Manufacturing," *arXiv:2104.09562 [cs]*, Apr. 2021. [Online]. Available: http://arxiv.org/abs/2104.09562

[9] S. B. Moore, W. B. Glisson, and M. Yampolskiy, "Implications of Malicious 3D Printer Firmware," *Hawaii International Conference on System Sciences 2017 (HICSS-50)*, Jan. 2017.

[10] D. R. Safford and M. Wiseman, "Hardware Rooted Trust for Additive Manufacturing," *IEEE Access*, vol. 7, pp. 79 211–79 215, 2019.

[11] J. Gatlin, S. Belikovetsky, S. B. Moore, Y. Solewicz, Y. Elovici, and M. Yampolskiy, "Detecting Sabotage Attacks in Additive Manufacturing Using Actuator Power Signatures," *IEEE Access*, vol. 7, pp. 133 421–133 432, 2019.

[12] L. M. G. Graves, J. Lubell, W. King, and M. Yampolskiy, "Characteristic Aspects of Additive Manufacturing Security From Security Awareness Perspectives," *IEEE Access*, vol. 7, pp. 103 833–103 853, 2019.

[13] S. Belikovetsky, Y. A. Solewicz, M. Yampolskiy, J. Toh, and Y. Elovici, "Digital Audio Signature for 3D Printing Integrity," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 5, pp. 1127–1141, May 2019.

[14] C. Liu, C. Kan, and W. Tian, "An Online Side Channel Monitoring Approach for Cyber-Physical Attack Detection of Additive Manufacturing." American Society of Mechanical Engineers Digital Collection, Jan. 2021.

[15] M. Farhan Khan, A. Alam, M. Ateeb Siddiqui, M. Saad Alam, Y. Rafat, N. Salik, and I. Al-Saidan, "Real-time defect detection in 3D printing using machine learning," *Materials Today: Proceedings*, vol. 42, pp. 521–528, Jan. 2021.

[16] A. Everitt, "gcodeparser: Python gcode parser," Jun. 2021. [Online]. Available: https://github.com/AndyEveritt/GcodeParser