Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS USA 2025 - Selected Papers from the 25th Annual Digital Forensics Research Conference USA

Leveraging memory forensics to investigate and detect illegal 3D printing activities

Hala Ali [a,*], Andrew Case [b], Irfan Ahmed [a]

[a] *Department of Computer Science, Virginia Commonwealth University, USA*
[b] *Volatility Foundation, USA*

A R T I C L E I N F O

Keywords:
3D printing
Memory forensics
Object reconstruction
Ultimaker cura
Volatility 3

A B S T R A C T

As 3D printing is widely adopted across critical sectors, malicious users exploit this technology to produce illegal tools for criminal activities. The increasing availability of affordable 3D printers and the limitations of current regulations highlight the urgent need for robust forensic capabilities. While existing research focuses on the physical forensics of printed objects, the digital aspects of 3D printing forensics remain underexplored, resulting in a significant investigative gap. This paper introduces *SliceSnap*, a novel memory forensics framework that analyzes the volatile memory of slicing software, which is essential for converting 3D models into printer-executable G-code instructions. Our investigation focuses on Ultimaker Cura, the most popular Python-based slicing tool. By leveraging the Python garbage collector and conducting structural analysis of its objects, *SliceSnap* can extract the mesh data of 3D models, G-code instructions, slicing settings, detailed 3D printer metadata, and logging information. Given the potential for slicing software compromises, our framework extends beyond artifact extraction to include the complementary analysis tool, *G-parser*. This tool detects malicious G-code manipulations by finding the discrepancies between the original settings and those extracted from the G-code. Evaluation results demonstrated the effectiveness of *SliceSnap* in recovering design files and G-code of various criminal tools, such as firearms and TSA master keys, with 100% accuracy, in addition to providing detailed information about the slicing software and 3D printer. The evaluation also analyzed the temporal persistence of memory artifacts across critical stages of Cura's lifecycle. Moreover, through *G-parser*, the framework successfully detected the G-code manipulations conducted by our novel attack vector that targets G-code during inter-process communication within the software. Implemented as Volatility 3 plugins, *SliceSnap* provides investigators with automated capabilities to detect 3D printing-related criminal activities.

1. Introduction

3D printing, also known as additive manufacturing (AM), is widely used in various critical sectors, such as healthcare and energy supply (Budzik et al., 2022). This technology enables the manufacture of complex geometries with high precision through a layer-by-layer construction process using a wide range of materials, including plastics, ceramics, metals, and composites (Hasanov et al., 2021). Due to the widespread availability of cost-effective 3D printers, malicious users exploit this technology to produce illegal tools that cannot be traced, such as firearms (Stilgherrian, 2017) and keys (Moon, 2016).

Despite efforts by the U.S. Department of State through the International Traffic in Arms Regulations (ITAR) to restrict the proliferation of unauthorized 3D-printed tools (Li et al., 2018) and state-level initiatives requiring background checks for 3D printer purchases (Piltch, 2023), numerous incidents continue to be reported. In 2023, three young people were arrested for producing a 3D-printed ghost gun at an East Harlem daycare in New York City (Miller, 2023). In the same year, a student was also arrested for bringing a 3D-printed gun to Chavez High School in Houston, Texas (Carpenter, 2023). More recently, in 2024, a 26-year-old man was arrested for killing the United Healthcare CEO using a 3D-printed ghost gun and silencer in New York City (Sundby et al., 2024).

The rise in incidents of illicit 3D-printed items and the ineffectiveness of current regulations (Bryans, 2015) highlight the urgent need for advanced forensic techniques in 3D printing investigations. While existing research has focused primarily on physical forensics, such as analyzing surface features of the printed object to identify source 3D

* Corresponding author.
E-mail addresses: alih16@vcu.edu (H. Ali), andrew@dfir.org (A. Case), iahmed3@vcu.edu (I. Ahmed).

https://doi.org/10.1016/j.fsidi.2025.301925

printers (Li et al., 2018; Shim et al., 2023; Shim and Hou, 2023), digital forensics remains underexplored. Current approaches are limited to recovering design and log files from the filesystem of 3D printer control systems (Garland et al., 2024a; Rais et al., 2023) or recognizing printed objects through the analysis of design files that are assumed to be available (Ma et al., 2020; Pham et al., 2018).

To address these limitations in digital forensic capabilities, our efforts focus on analyzing the slicing software, a critical component in the 3D printing workflow that runs on the 3D printer controller PC to convert design files into printer-executable G-code instructions. These instructions control the operations of the 3D printer, including the print head movements and heating settings to produce the objects (Ali et al., 2025). However, malicious users can hide their activities on the filesystem, such as securely wiping the design and generated G-code files after successful prints, clearing software logs, and using private browsing to obtain 3D models. Therefore, we introduce *SliceSnap*, a memory forensics framework that analyzes the volatile memory of slicing software. Our investigation focuses on Ultimaker Cura, one of the most popular slicing applications (Bricknell, 2025, Bricknell, 2023).

Since Ultimaker Cura is written in Python, *SliceSnap* investigates the Python garbage collector to access and analyze the object structures, allowing extensive data extraction. The framework's capabilities encompass the extraction and processing of 3D model mesh data to reconstruct design files (.stl), as well as the recovery of G-code instructions for external file generation (.gcode). The presence of G-code in memory serves as crucial digital evidence, indicating the execution of the printing job (Garland et al., 2024a). Furthermore, *SliceSnap* recovers the complete set of slicing settings utilized in G-code generation and extracts detailed printer metadata, including device identification, network configuration, and hardware specifications. These forensic capabilities are implemented as Volatility 3 plugins (Volatility Foundation, 2017), facilitating automated evidence extraction and analysis processes.

Prior research demonstrated various security vulnerabilities in slicing software. Through static and dynamic analysis, Moore et al. (2016) identified critical security flaws in popular slicing software, including Ultimaker Cura, that enabled unauthorized G-code modifications. Kurkowski et al. (2022) illustrated how malware targeting slicing software memory could access and change G-code instructions, while Belikovetsky et al. (2017) demonstrated attackers' ability to compromise victim systems for 3D models manipulations and developed worms targeting G-code files in memory. Our analysis of Ultimaker Cura revealed an additional vulnerability[1] that enables stealthy G-code manipulation during the slicing process. Unlike mesh data, which is stored and transmitted as binary-encoded arrays, Cura transmits G-code in plaintext over unencrypted inter-process communication (IPC) between the *Cura* and *CuraEngine* processes. This vulnerability allows malicious actors to intercept and modify G-code before it reaches the application GUI. In criminal investigations, suspects could leverage knowledge of these vulnerabilities as part of their defense strategy (i.e., Trojan horse defense), claiming their systems were compromised and the illegal G-code was generated without their knowledge. In such cases, *SliceSnap* allows investigators to extract both the design files and G-code from memory, compare them with those generated from a clean reference system, and definitively prove or disprove the suspect's claims.

Moreover, such vulnerabilities allow malicious manipulations of G-code settings, leading to visual deformities and altered physical properties of printed objects (Rais et al., 2021b, 2021a), thus causing weapons that appear normal but fail catastrophically when fired. From a digital forensics perspective, determining whether defects were accidental or deliberate is crucial for establishing criminal intent. Therefore, our framework extends beyond artifact extraction to include malicious G-code manipulation detection through the proposed complementary analysis tool, *G-parser*. This tool analyzes the recovered G-code to identify the critical instructions that control the heating, cooling, and layer height parameters of the printing process. By comparing these parameters with the baseline settings extracted by the framework, *G-parser* can detect discrepancies and flag them as malicious modifications, thus confirming the criminal intent.

As discussed in the evaluation, *SliceSnap* demonstrated 100% accuracy in recovering design files and G-code for a diverse set of 3D weapon models (PX4, Beretta Prop Gun, Grenade Container, Magazine, Revolver, Bullets, AK_47 Part, and Revolver Cylinder) and security-critical items (TSA Master Key and Handcuff Key). The results were validated using Jaccard similarity and Levenshtein distance metrics (Garland et al., 2024a; Haldar and Mukhopadhyay, 2011). Additionally, *SliceSnap* proved its cross-version adaptability by working with different versions of Ultimaker Cura, each running with a different Python version and operating system. Furthermore, it outperformed the traditional signature-based search approach using strings and regular expressions. Expanding its forensic capabilities, *SliceSnap* successfully extracted all 581 settings used in G-code generation while retrieving detailed metadata, including software version information, printer specifications (e.g., device ID, name, IP address, connection mode, and firmware), and slicing timestamp (e.g., date and time). Further evaluation analyzed residual artifacts extracted at critical stages of Cura's lifecycle. Finally, *SliceSnap*, through *G-parser*, proved its effectiveness in detecting the disabled cooling fan during drone propeller printing and subtle setting modifications conducted by exploiting the IPC vulnerability. The impact of these manipulations was validated through mechanical testing of the printed samples. Based on these evaluation results, *SliceSnap* provides investigators with automated tools for efficient recovery and analysis of digital evidence in 3D printing-related investigations. We summarize the contributions of this paper as follows.

- Propose *SliceSnap*, a novel memory forensics framework that analyzes the Ultimaker Cura software to recover critical digital evidence of illegal printing activities.
- Implement novel Volatility 3 plugins to automate the extraction and analysis processes.
- Propose *G-parser*, a complementary analysis tool that detects malicious G-code manipulations to confirm whether defects are accidental or deliberate.
- Evaluate both the *SliceSnap*'s effectiveness with multiple criminal test scenarios, using diverse 3D models and the ability of *G-parser* to detect G-code manipulation conducted by our novel G-code attack vector.

2. Related work

Prior research in 3D printing forensics spans both physical and digital approaches. For physical forensics, Li et al. (2018) developed *PrinTracker*, extracting distinctive texture patterns via GLCM modeling to fingerprint source printers. Shim et al. (Shim et al., 2023; Shim and Hou, 2023) analyzed surface features using transformer-based models to identify printer models, materials, and slicing software using the SI3DP++ dataset (Shim et al., 2021). Digital forensics approaches have focused on object recognition and filesystem analysis to detect illegal prints and malicious activities. Ma et al. (2020) developed an object recognition system using G-code projections and images of the object during the printing process as input to a CNN model. Pham et al. (2018) focused on converting the mesh data of the 3D model into shape characteristic distributions to be analyzed by a CNN model. Garland et al. (2024b) investigated file-based artifacts from slicing software by analyzing system Registry, logs, and network captures, later developing Autopsy modules for G-code recovery (Garland et al., 2024a). Rais et al. (2023) proposed *FRoMEPP*, a forensic readiness framework integrating physical and digital artifact collection, including printer logs, slicing software logs, and network captures.

[1] This vulnerability has been assigned CVE-2024-51330.

Unlike existing digital forensics methods, *SliceSnap* analyzes the volatile memory of slicing software to recover evidence even when no disk traces exist. It provides comprehensive artifacts of the printing activities by extracting design files, G-code, slicing settings, printer metadata, and process logs. This memory-based framework offers several key advantages as it employs automated data structure analysis rather than simple string pattern matching, reconstructs complete forensic timelines of the slicing process, provides hardware attribution through detailed printer specifications, and detects deliberate G-code manipulations through G-parser that helps in establishing the criminal intent of the user.

3. SliceSnap framework

Fig. 1 illustrates the *SliceSnap* framework that begins with traversing the linked list of the active processes in kernel memory to identify the Cura process. Since Cura is implemented in Python, the first step toward extracting its objects is accessing the executable header of the Python process to locate the `_PyRuntime` global variable. For instance, in a Linux environment, this variable is located in the dynamic symbol table (`.dynsym` section) of the ELF header. `_PyRuntime` points to the Garbage Collector (GC) of Python, which represents the entry point to access the objects in memory. The garbage collector maintains three generations of objects based on their allocation time. The first generation contains recently created objects, while the third holds persistent objects. A `PyGC_Head` header structure precedes each object, containing bidirectional pointers to form a doubly linked list of objects for each generation. Moreover, Python implements distinct structures for different object types. For example, it uses the `PyDictObject` structure to represent the dictionaries, while the `PyListObject` and `PyModuleObject` structures to represent the lists and modules, respectively.

Upon accessing Cura objects, *SliceSnap* performs automated extraction and analysis through four specialized plugins. The *Mem_Design* plugin handles the extraction of mesh data and G-code. Moreover, it implements additional processing to reconstruct the design file from its mesh data. The *Mem_Setting* plugin extracts the slicing settings used to generate the G-code. The *Mem_Metadata* plugin retrieves information about the slicing software in addition to the specifications, capabilities, and network connectivity parameters of the 3D printer. Finally, the *Mem_Log* plugin recovers operational logs and timestamps. *SliceSnap* also includes the *G-parser* tool that analyzes recovered G-code to detect potential manipulations in slicing settings.

3.1. Ultimaker Cura architecture

We analyze the internal architecture of Ultimaker Cura software that consists of three main components, *Cura* (Ultimaker, a), *Uranium* (Ultimaker, c), and *CuraEngine* (Ultimaker, b), as illustrated in Fig. 2. Cura represents the front-end GUI of the application, where the 3D models are processed and visualized. Upon loading a 3D model (STL design file), Cura establishes a local socket connection with *CuraEngine* on port 49674. This connection is managed by the *libArcus* protocol, which is implemented in the `ArcusCommunication` class. Through both the `CuraEngineBackend` and `StartSliceJob` classes, Cura manages the slicing settings in a top-down approach across the global, extruder, and per-object stacks.

Uranium generates the mesh data of the 3D model, consisting of vertices (3D points marking the corners of each surface's triangle), normals (vectors perpendicular to the triangle surfaces), and indices (link the vertices to form triangles). It organizes the printing scene hierarchically, beginning with a `Scene` node that maintains the G-code. For each 3D model loaded by the user, *Uranium* creates two `SceneNode` nodes as children of the root node. Each node encapsulates mesh data, per-object slicing settings, and transformation matrices for translation, scaling, rotation, and shearing operations. The first node represents the original state of the model, while the second node stores a copy of the model with any applied modifications to its transformation matrices. It is important to note that the root node includes additional persistent child nodes, such as the camera (3D viewing) and the build volume, to maintain the global configurations of the scene. Fig. 3 demonstrates this hierarchy using a Beretta Prop Gun model. The second node contains the scaled version of the model, while the last node preserves its original state. Both maintain a per-object stack that encapsulates the slicing settings used with the model. The persistent nodes hold 3D models representing the printing view and contain the global container stack to store the global configurations.

During the slicing process, *Cura* transmits both the settings and mesh data associated with the final state of the model to the *CuraEngine* process. *CuraEngine* then generates the corresponding G-code through the `FffGcodeProcessor` and `FffGcodeWriter` classes and sends it back as a sequence of G-code layers encapsulated in `GCodeLayer` messages. The received G-code is then forwarded to *Uranium* to be stored as a

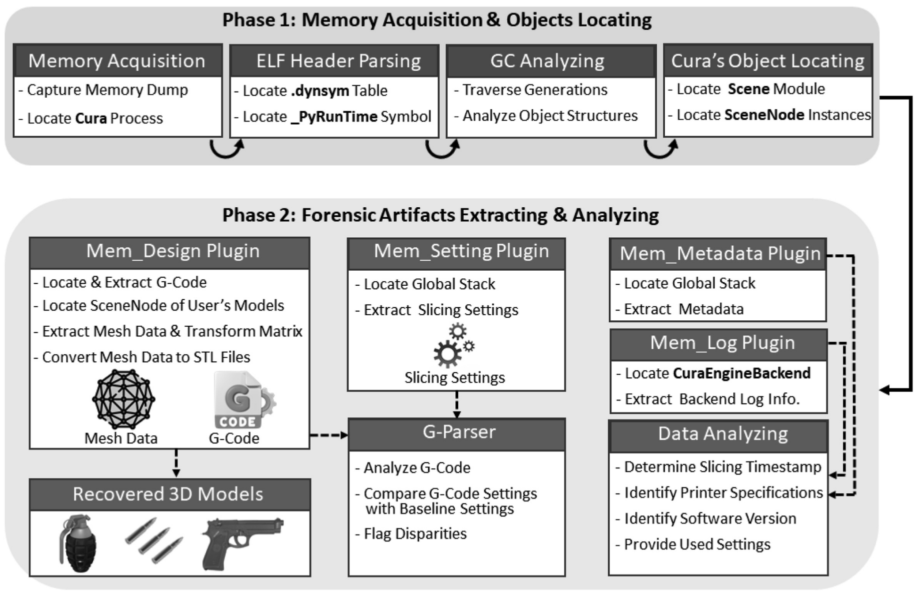

Phase 1: Memory Acquisition & Objects Locating
Memory Acquisition
- Capture Memory Dump
- Locate Cura Process
ELF Header Parsing
- Locate .dynsym Table
- Locate _PyRunTime Symbol
GC Analyzing
- Traverse Generations
- Analyze Object Structures
Cura's Object Locating
- Locate Scene Module
- Locate SceneNode Instances
Phase 2: Forensic Artifacts Extracting & Analyzing
Mem_Design Plugin
- Locate & Extract G-Code
- Locate SceneNode of User's Models
- Extract Mesh Data & Transform Matrix
- Convert Mesh Data to STL Files
Mesh Data
G-Code
Mem_Setting Plugin
- Locate Global Stack
- Extract Slicing Settings
Slicing Settings
Mem_Metadata Plugin
- Locate Global Stack
- Extract Metadata
Mem_Log Plugin
- Locate CuraEngineBackend
- Extract Backend Log Info.
G-Parser
- Analyze G-Code
- Compare G-Code Settings with Baseline Settings
- Flag Disparities
Data Analyzing
- Determine Slicing Timestamp
- Identify Printer Specifications
- Identify Software Version
- Provide Used Settings
Recovered 3D Models

Fig. 1. Overview of the proposed *SliceSnap* framework.

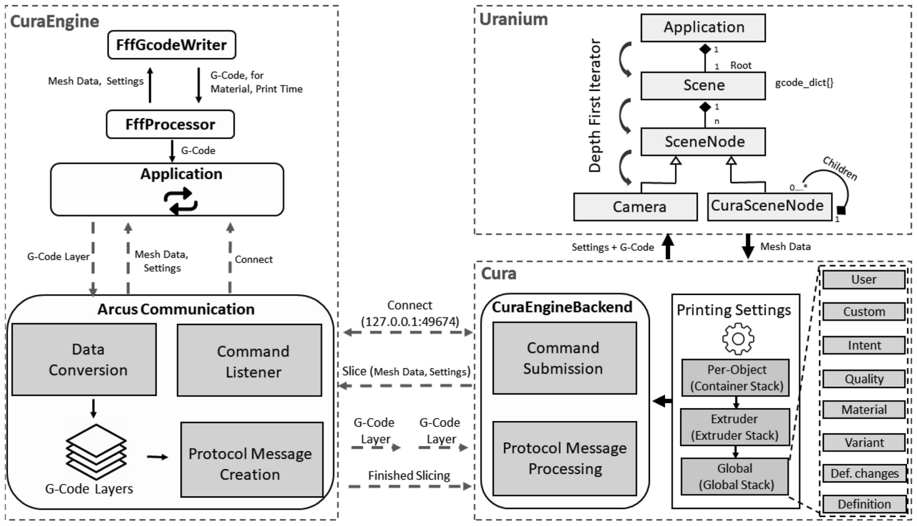


Fig. 2. Internal architecture of Ultimaker cura software.

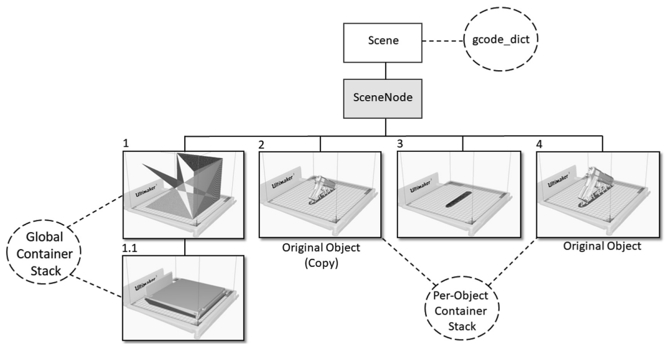


Fig. 3. Hierarchical storage of the 3D models, G-Code, and settings within the scene.

gcode_dict attribute for the entire scene.

3.2. *Design files reconstruction*

For G-code extraction, the *Mem_Design* plugin starts with locating the *Scene.py* class in memory. In Python, each class is represented as a module with an associated dictionary containing its components. Given that `gcode_dict` is an attribute of the `Scene.py` class, it is located within the class dictionary by its matching key name. This dictionary utilizes a single empty string key to store G-code layers as a list, where each element represents one layer with sequential ordering maintained through list indices. Upon extracting, the plugin stores the G-code data into an external file (.gcode).

Mem_Design then traverses the imported classes by `Scene.py` to identify all the `SceneNode.py` instances. For each node, it leverages the `_name` attribute that helps recognize the models loaded by the user while using the `_mesh_data` and `_world_transformation` attributes to locate the mesh data and transformation information, respectively. For each vertex (v_x, v_y, v_z) within the mesh data, the plugin applies a 4 × 4 transformation matrix incorporating scaling, rotation, and shearing. The translation vector derived from the fourth column of the matrix is then added to these transformed vertices to position the model on the build plate. The normals, on the other hand, are affected only by rotation and scaling. They are padded to 4D vectors (n_x, n_y, n_z, 0) to nullify translation effects. After transformation, normals are normalized to preserve direction and accuracy. At the final step, *Mem_Design* swaps Y and Z axes to convert from Cura's Y-up system (where Y indicates vertical direction and Z represents horizontal depth) to the standard Z-up system. Note that the STL format uses interconnected triangles with shared vertices to represent object surfaces. However, user-loaded models contain only vertices and normals. Consequently, the plugin processes vertices sequentially in groups of three as the application handles this internally. The final design files are then written in binary format (.stl) to align Cura's binary-encoded array structure for mesh data storage.

3.3. *Slicing settings extraction*

Ultimaker Cura maintains slicing settings through three distinct stacks: global, extruder, and per-object. It employs a top-down evaluation approach, where extruder and per-object settings override global settings. Therefore, the global stack contains the final setting values used for G-code generation. This stack comprises eight containers, the 'user' container for the current changes, the 'custom' container for user preferences, the 'intent' container for optimization goals, the 'quality' container for predefined setting groups, the 'material' container for material property settings, the 'variant' container for hardware settings, the 'definition changes' container for printer modifications, and finally, at the bottom, the 'definition' container for official setting definitions.

For settings retrieval, the *Mem_Setting* plugin accesses the global stack through the persistent `SceneNode` instance in the hierarchy. It accesses the `_global_container_stack` dictionary, which maintains references to all containers. The plugin specifically targets the 'user' container, extracting settings from the `_SettingDefinition__property_values` entry within the `_definition_cache` dictionary. Each setting encompasses multiple attributes, including its current and default values, valid range boundaries, unit, descriptive label, and detailed description. Current values are present only for modified settings; consequently, the plugin references default values for unmodified settings.

3.4. *Metadata extraction*

The metadata is located in the global stack. The stack itself maintains a top-level metadata dictionary about the entire stack. Meanwhile, each of its containers maintains its metadata dictionary with specific attributes relevant to its role. For metadata extraction, the *Mem_Metadata* plugin focuses on two key sources of information from the stack and its definition container. This plugin leverages the `_metadata` field within the stack structure that includes critical information about the used 3D printer. Moreover, it iterates the `_containers` field to access the definition container and extract its metadata.

The metadata at the stack level includes version information about Cura settings, unique identifiers such as the group_id (UUID) that identifies the printer group, specific printer ID, and display name. It also stores network configuration details, including the printer IP address settings and connection mode, which can help trace how the printer was connected and used. On the other hand, the definition container encapsulates general specifications about the printer model and operational capabilities. This encompasses material compatibility matrices, hardware interface specifications including USB and network connectivity capabilities, print head configurations including nozzle quantity and specifications, and firmware-related parameters. *Mem_Metadata* plugin combines information from both sources to generate comprehensive metadata documentation.

3.5. Printing log extraction

Ultimaker Cura logs all activities during the slicing process. The *Mem_Log* plugin retrieves this logging data by locating the `CuraEngineBackend.py` class to access its `_backend_log` attribute, which contains UTF-8 encoded strings. However, the log content varies based on the memory acquisition timing. For instance, after loading a 3D model, the log includes information on the connection with the *CuraEngine* process and the version of the slicing software. In contrast, deploying the plugin with a memory dump acquired after slicing the model will provide information about the slicing settings used to generate the G-code, the slicing software, the 3D printer, and the timestamp of the slicing process. This plugin provides crucial forensic evidence for analyzing potential illicit manufacturing activities, allowing investigators to reconstruct the exact timeline and parameters of printing operations.

3.6. G-parser

G-parser focuses on analyzing critical printing parameters known to compromise the quality of the printed objects, such as nozzle and bed temperatures (Cho et al., 2019), cooling fan speed (Moore et al., 2017; Gao et al., 2018), and layer height (Rais et al., 2022). This tool processes both the G-code file extracted by the *Mem_Design* plugin and the settings provided by the *Mem_Setting* plugin. Since the G-code file is organized as layers of commands, *G-Parser* determines the beginning of each layer through the 'Z' parameter of the movement commands. For each layer, *G-Parser* locates the M104/M109 commands that adjust the nozzle temperature, M140/M190 commands that change the bed temperature, and M106/M107 commands that control the fan speed. It then recovers the setting values through the 'S' parameter of these commands. Moreover, it calculates the layer height as the difference between the consecutive 'Z' values.

To establish a comparison baseline, this tool leverages the output of the *Mem_Setting* plugin. It recovers the values of the `material_print_temperature_layer_0`, `material_bed_temperature_layer_0`, and `cool_fan_speed_0` attributes as the nozzle temperature, bed temperature, and fan speed used with the first layer, respectively. While, for the subsequent layers, *G-Parser* extracts the values of the `material_print_temperature`, `material_bed_temperature`, and `cool_fan_speed` attributes. Furthermore, it determines the layer height through the value of the `layer_height` attribute. In the final step, *G-parser* compares the parsed G-code settings with these baseline settings to find the disparities that indicate potential malicious manipulations. This approach enables forensic investigators to identify unauthorized modifications to G-code through compromised slicing software.

4. Experimental evalaution

We conducted our experiments using two virtual machines (VMs) running on VMware Workstation 16 Pro (version 16.2.5), each configured with Ubuntu 20.04.6 LTS, 4 GB RAM, and Python 3.8.10 (Foundation). The target VM, simulating a criminal's device, has Ultimaker Cura 4.4.1 installed, while the investigation VM is equipped with the Volatility 3 framework (version 2.5.0) and required packages for design file reconstruction (`numpy 1.17.4` and `numpy-stl 3.1.2`). To facilitate memory analysis through Volatility 3, we used the *dwarf2json* tool to generate the debugging symbol table files for both the kernel and Python interpreter (Volatility Foundation).

The extraction accuracy of *Mem_Design* was evaluated across various 3D models that vary in geometry complexities, parts, and size. This includes weapons-related models (PX4, Beretta Prop Gun, Grenade, Magazine, Revolver, Bullets, AK_47 Part, and Revolver Cylinder) and security-critical items (TSA Master Key and Handcuff Key). Memory dumps were acquired after slincing each model in a separate session. The STL files of these models were downloaded from the popular *Thingiverse* repository (Thingiverse)[2].

To establish a reference baseline for each of the extracted STL and G-code files, we exported the 3D model as an STL file after completing the slicing process to get its final state. We also saved the generated G-code to an external file. For STL comparison, we focused specifically on vertices, as normal vectors solely indicate design directionality. With the G-code files, we considered the commands while excluding the comments to ensure a meaningful comparison. We employed multiple similarity metrics to evaluate extraction accuracy. The Jaccard similarity measures the overlap between recovered and baseline files. The metric is defined as $J(S1, S2) = \frac{|S1 \cap S2|}{|S1 \cup S2|}$, where S1 and S2 represent vertex or command sets. We extended this with sequential Jaccard similarity to consider the sequence importance. We also implemented Levenshtein distance analysis, converting vertex coordinates and G-code commands to strings for comparison (e.g., transforming coordinates (1.234, 5.678, 9.012) to "1.234 5.678 9.012"). This metric provides enhanced sensitivity for detecting subtle modifications. Table 1 demonstrates the ability of *Mem_Design* to extract STL and G-code files accurately in terms of similarity measures, considering the number of vertices and normals, along with G-code layers and commands. This table displays a representative subset of ten 3D models; the complete evaluation dataset is available in Appendix A.

While our framework achieved 100% extraction accuracy in controlled scenarios, forensic investigators may encounter transformed variations of the same model in real-world cases. Simple transformations produce completely different vertex values and G-code, causing low Jaccard similarity and large Levenshtein distance, even though the 3D model and functionality remain unchanged. Thus, in these scenarios, we used appropriate similarity metrics, such as D2 Shape Distribution, Volume/Surface Area comparisons, and Heat Kernel Signatures. We evaluated the effectiveness of *Mem_Design* using these metrics with the Beretta Prop Gun model after applying different modifications, such as extended barrel, rotation, and scaling down (See Appendix B). These metrics reveal the true nature of modifications while maintaining high shape similarity scores.

We also compared the performance of *Mem_Design* against a conventional forensic method combining Unix `strings` utility with Regular Expressions, named *Strings_RE*. This method has been proven effective in various forensic analyses, including data recovery from social media and messaging platforms (Thantilage and Le Khac, 2019; Barradas et al., 2019; Davis et al., 2022). However, Ultimaker Cura stores the STL files as binary-encoded mesh data within NumPy arrays, making *Strings_RE* ineffective for STL file extraction. On the other hand, this method remains viable for G-code extraction, given its plaintext storage format.

Table 2 presents a comparative analysis between the *Mem_Design*

[2] In this research, no actual illegal items were physically printed.

Table 1
Evaluation of *Mem_Design* plugin extraction accuracy across various STL and G-Code files.

File	Baseline STL		Extracted STL		Baseline G-Code		Extracted G-Code		Jaccard	Seq. Jaccard	Levenshtein
	Vertices	Normals	Vertices	Normals	Layers	Commands	Layers	Commands	Similarity	Similarity	Distance
PX4	11712	22887	11712	22887	417	203703	417	203703	100%	100%	0
Beretta Prop Gun	382704	127568	382704	127568	356	665032	356	665032	100%	100%	0
Grenade Container	84918	28306	84918	28306	284	436442	284	436442	100%	100%	0
Magazine	5886	1962	5886	1962	486	173947	486	173947	100%	100%	0
Revolver	48234	16078	48234	16078	275	89422	275	89422	100%	100%	0
Revolver Cylinder	86670	28890	86670	28890	122	138806	122	138806	100%	100%	0
Nato Bullets	74988	24996	74988	24996	71	65108	71	65108	100%	100%	0
AK_47 Part	4812	1604	4812	1604	398	139724	398	139724	100%	100%	0
TSA Master Key	6588	2196	6588	2196	9	14564	9	14564	100%	100%	0
Handcuff Key	1422	474	1422	474	15	15744	15	15744	100%	100%	0

Table 2
Comparative analysis of G-Code recovery: *Mem_Design* plugin versus *Strings_RE* method.

File	SliceSnap					Strings_RE				
	#Layers	#Commands	Jaccard Similarity	Seq. Jaccard Similarity	Levenshtein Distance	#Layers	#Commands	Jaccard Similarity	Seq. Jaccard Similarity	Levenshtein Distance
PX4	417	203703	100%	100%	0	425	184116	0.250%	0.206%	214033
Beretta Prop Gun	356	665032	100%	100%	0	365	609383	0.235%	0.208%	569588
Grenade Continer	284	436442	100%	100%	0	285	414765	0.250%	0.143%	295217
Magazine	486	173947	100%	100%	0	491	160419	0.235%	0.186%	146003

plugin and *Strings_RE* across four G-code files (PX4, Beretta prop gun, grenade container, and magazine). The performance limitations of *Strings_RE* arise from the complex memory architecture of Cura. The software's layer-based caching mechanism, while optimizing runtime performance, creates multiple memory references to identical G-code segments, leading to layer duplication and sequence disruption during extraction, and thus reducing the Jarracrd similarity. Additionally, memory analysis reveals that ASCII-encoded G-code commands are fragmented and interspersed with corrupted content, resulting in command sequence disruption and syntax errors, thus causing larger Levenshtein distances. Fig. 4 demonstrates these artifacts by visualizing the extracted G-code.

In some scenarios, Cura can be used to slice multiple 3D models together, as shown in Fig. 5. Therefore, we conducted an experiment that considered processing two models, the Beretta Prop Gun (A) and the Grenade Container (B). This experiment demonstrates the residual data

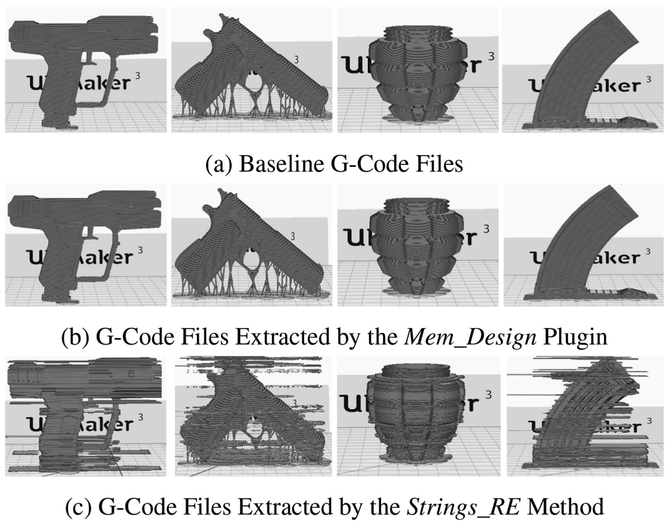

(a) Baseline G-Code Files
(b) G-Code Files Extracted by the *Mem_Design* Plugin
(c) G-Code Files Extracted by the *Strings_RE* Method

Fig. 4. Comparative visualization of G-Code recovery: *Mem_Design* plugin versus *Strings_RE* method.

of each model across critical stages of the software lifecycle. Initially, after loading model (A), *Mem_Design* extracts only the STL file as no G-code is present in memory. During slicing, G-code data accumulates incrementally in the `gcode_dict` attribute. Once the slicing process is complete, the full G-code is available in memory. However, for any simple change in the scene, such as scaling the model, removing the model, or even adding a new model, Cura, through its `CuraEngine-Backend.py` class, deallocates G-code references by clearing the `gcode_dict` attribute. Therefore, both loading model (B) and removing model (A) lead to deleting the G-code from memory.

To demonstrate the cross-version adaptability of our approach, we evaluated the effectiveness of the *Mem_Design* plugin by analyzing memory dumps of three versions of Ultimaker Cura, each running with a different Python version and operating system, as detailed in Table 3. We maintained identical slicing parameters and model dimensions across all tests to ensure experimental validity. Results show newer Cura versions generate G-code more efficiently due to the enhancement in slicing algorithms, with version 5.11.0-alpha.0 producing ∼ 31–46% fewer commands than 4.4.1 while preserving identical layer counts. This efficiency gain is attributed to the Arachne engine introduced in Cura 5.x, which implements variable line width technology and sophisticated path planning algorithms.

Figs. 6–8 demonstrate the artifacts extracted by the *Mem_Setting*, *Mem_Matadata*, and *Mem_Log* plugins, respectively, from a memory dump acquired after slicing the Beretta prop gun model. Fig. 6 presents a subset of the 581 settings extracted by the *Mem_Setting* plugin, including the G-code flavor which defines the machine-specific instruction set ('Griffin' for Ultimaker printers), nozzle and bed temperatures, fan speed, infill characteristics (i.e., pattern, density, and direction), and layer height, in addition to the number of top and bottom layers.

The determined section in the *Mem_Metadata*'s output, as shown in Fig. 7, represents the metadata extracted from the global stack, while the following entries are extracted from the definition container. The stack provides information about the connected 3D printer, including its ID ('Ultimaker-008b14 #2'), display name ('Ultimaker-008b14'), connection mode (2: network connection), IP address (172.22.123.205), and its configuration mode (e.g., manual). The definition container provides general descriptions of the hardware capabilities of the printer, such as

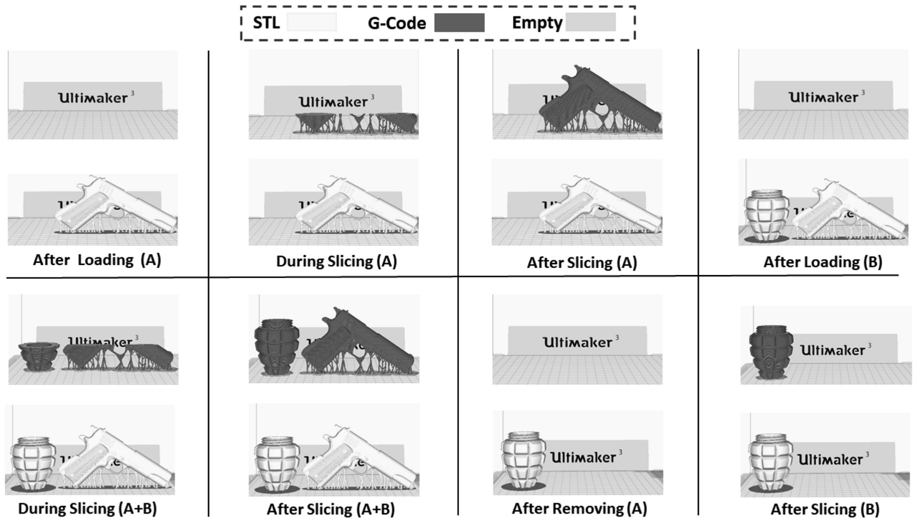


Fig. 5. Visualization of the extracted stl and G-code files by the *Mem_Design* plugin during multi-model processing.

Table 3
Evaluation of *Mem_Design* across differnt cura versions.

Env. & File		Cura (V.4.4.1)	Cura (V.4.13.0)	Cura (V.5.11.0-alpha.0)
Environment	Python	3.8.10	3.10.12	3.12.3
	Ubuntu	20.04.4 LTS	22.04.2 LTS	24.04.2 LTS
PX4	# Layers	417	417	417
	# Commands	203703	184471	123015
Beretta Prop Gun	# Layers	356	356	356
	# Commands	665032	578567	402091
Grenade Continer	# Layers	284	284	284
	# Commands	436442	425399	236641
Magazine	# Layers	486	486	486
	# Commands	173947	169700	120005

supported material types, as well as information about the firmware and its downloading URL.

Fig. 8 illustrates the UTF-8 decoded metadata that provides information about the connection between the *Cura* and *CuraEngine* processes (127.0.0.1:49674), software version, the date (17-01-2025) and time (20:43:12) of the slicing process, along with the printer name and the

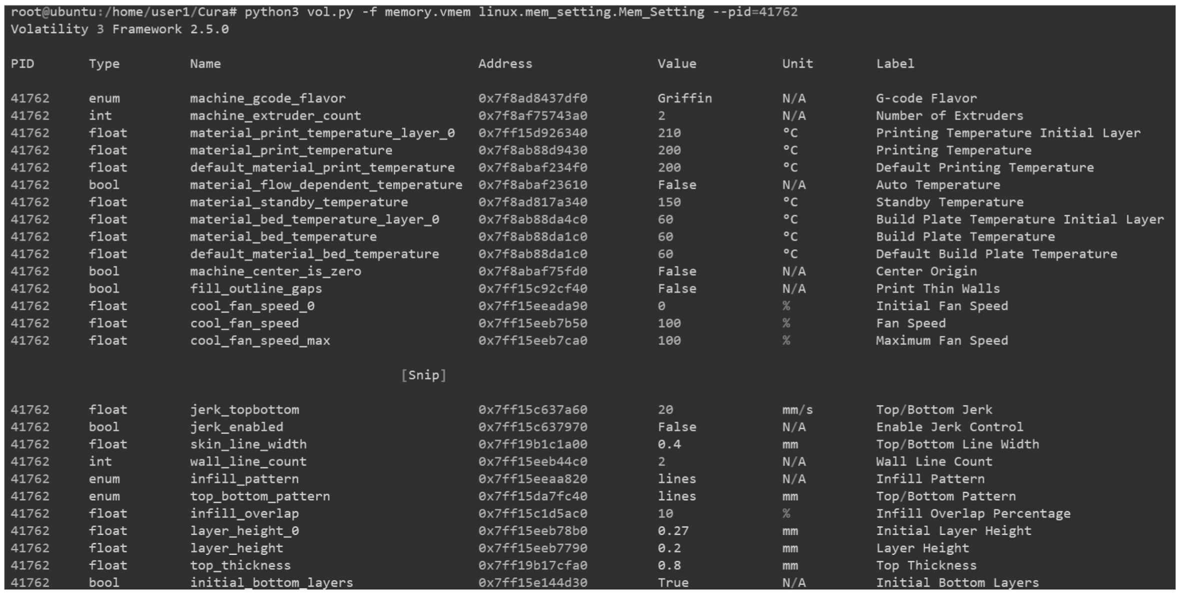
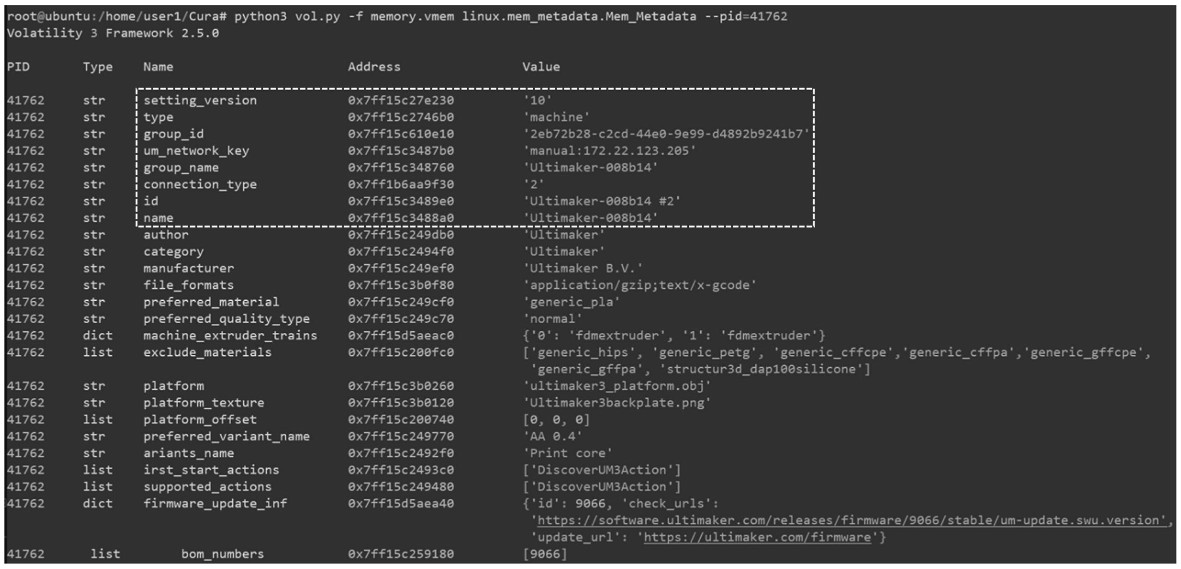
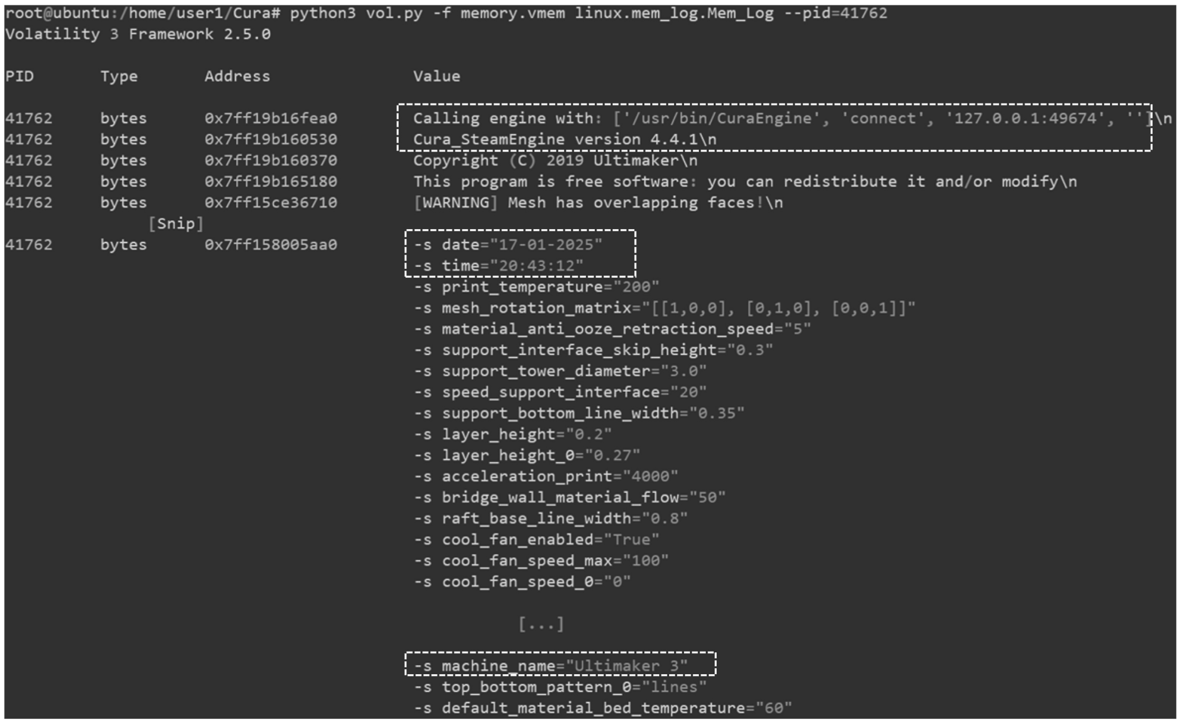

```
root@ubuntu:/home/user1/Cura# python3 vol.py -f memory.vmem linux.mem_setting.Mem_Setting --pid=41762
Volatility 3 Framework 2.5.0

PID      Type      Name                               Address          Value      Unit      Label

41762    enum      machine_gcode_flavor               0x7f8ad8437df0   Griffin    N/A       G-code Flavor
41762    int       machine_extruder_count             0x7f8af75743a0   2          N/A       Number of Extruders
41762    float     material_print_temperature_layer_0 0x7ff15d926340   210        °C        Printing Temperature Initial Layer
41762    float     material_print_temperature         0x7f8ab88d9430   200        °C        Printing Temperature
41762    float     default_material_print_temperature 0x7f8abaf234f0   200        °C        Default Printing Temperature
41762    bool      material_flow_dependent_temperature 0x7f8abaf23610  False      N/A       Auto Temperature
41762    float     material_standby_temperature       0x7f8ad817a340   150        °C        Standby Temperature
41762    float     material_bed_temperature_layer_0   0x7f8ab88da4c0   60         °C        Build Plate Temperature Initial Layer
41762    float     material_bed_temperature           0x7f8ab88da1c0   60         °C        Build Plate Temperature
41762    float     default_material_bed_temperature   0x7f8ab88da1c0   60         °C        Default Build Plate Temperature
41762    bool      machine_center_is_zero             0x7f8abaf75fd0   False      N/A       Center Origin
41762    bool      fill_outline_gaps                  0x7ff15c92cf40   False      N/A       Print Thin Walls
41762    float     cool_fan_speed_0                   0x7ff15eeada90   0          %         Initial Fan Speed
41762    float     cool_fan_speed                     0x7ff15eeb7b50   100        %         Fan Speed
41762    float     cool_fan_speed_max                 0x7ff15eeb7ca0   100        %         Maximum Fan Speed

                                               [Snip]

41762    float     jerk_topbottom                     0x7ff15c637a60   20         mm/s      Top/Bottom Jerk
41762    bool      jerk_enabled                       0x7ff15c637970   False      N/A       Enable Jerk Control
41762    float     skin_line_width                    0x7ff19b1c1a00   0.4        mm        Top/Bottom Line Width
41762    int       wall_line_count                    0x7ff15eeb44c0   2          N/A       Wall Line Count
41762    enum      infill_pattern                     0x7ff15eeaa820   lines      N/A       Infill Pattern
41762    enum      top_bottom_pattern                 0x7ff15da7fc40   lines      mm        Top/Bottom Pattern
41762    float     infill_overlap                     0x7ff15c1d5ac0   10         %         Infill Overlap Percentage
41762    float     layer_height_0                     0x7ff15eeb78b0   0.27       mm        Initial Layer Height
41762    float     layer_height                       0x7ff15eeb7790   0.2        mm        Layer Height
41762    float     top_thickness                      0x7ff19b17cfa0   0.8        mm        Top Thickness
41762    bool      initial_bottom_layers              0x7ff15e144d30   True       N/A       Initial Bottom Layers
```

Fig. 6. The output of the *Mem_Setting* plugin showing the settings used for G-Code generation.



```
root@ubuntu:/home/user1/Cura# python3 vol.py -f memory.vmem linux.mem_metadata.Mem_Metadata --pid=41762
Volatility 3 Framework 2.5.0

PID      Type    Name                       Address              Value

41762    str     setting_version            0x7ff15c27e230       '10'
41762    str     type                       0x7ff15c2746b0       'machine'
41762    str     group_id                   0x7ff15c610e10       '2eb72b28-c2cd-44e0-9e99-d4892b9241b7'
41762    str     um_network_key             0x7ff15c3487b0       'manual:172.22.123.205'
41762    str     group_name                 0x7ff15c348760       'Ultimaker-008b14'
41762    str     connection_type            0x7ff1b6aa9f30       '2'
41762    str     id                         0x7ff15c3489e0       'Ultimaker-008b14 #2'
41762    str     name                       0x7ff15c3488a0       'Ultimaker-008b14'
41762    str     author                     0x7ff15c249db0       'Ultimaker'
41762    str     category                   0x7ff15c2494f0       'Ultimaker'
41762    str     manufacturer               0x7ff15c249ef0       'Ultimaker B.V.'
41762    str     file_formats               0x7ff15c3b0f80       'application/gzip;text/x-gcode'
41762    str     preferred_material         0x7ff15c249cf0       'generic_pla'
41762    str     preferred_quality_type     0x7ff15c249c70       'normal'
41762    dict    machine_extruder_trains    0x7ff15d5aeac0       {'0': 'fdmextruder', '1': 'fdmextruder'}
41762    list    exclude_materials          0x7ff15c200fc0       ['generic_hips', 'generic_petg', 'generic_cffcpe','generic_cffpa','generic_gffcpe',
                                                                  'generic_gffpa', 'structur3d_dap100silicone']
41762    str     platform                   0x7ff15c3b0260       'ultimaker3_platform.obj'
41762    str     platform_texture           0x7ff15c3b0120       'Ultimaker3backplate.png'
41762    list    platform_offset            0x7ff15c200740       [0, 0, 0]
41762    str     preferred_variant_name     0x7ff15c249770       'AA 0.4'
41762    str     ariants_name               0x7ff15c2492f0       'Print core'
41762    list    irst_start_actions         0x7ff15c2493c0       ['DiscoverUM3Action']
41762    list    supported_actions          0x7ff15c249480       ['DiscoverUM3Action']
41762    dict    firmware_update_inf        0x7ff15d5aea40       {'id': 9066, 'check_urls':
                                                                  'https://software.ultimaker.com/releases/firmware/9066/stable/um-update.swu.version',
                                                                  'update_url': 'https://ultimaker.com/firmware'}
41762     list       bom_numbers            0x7ff15c259180       [9066]
```

Fig. 7. The output of the *Mem_Metadata* plugin showing the metadata (i.e., 3D printer characteristics).



```
root@ubuntu:/home/user1/Cura# python3 vol.py -f memory.vmem linux.mem_log.Mem_Log --pid=41762
Volatility 3 Framework 2.5.0

PID      Type      Address              Value

41762    bytes     0x7ff19b16fea0       Calling engine with: ['/usr/bin/CuraEngine', 'connect', '127.0.0.1:49674', '']\n
41762    bytes     0x7ff19b160530       Cura_SteamEngine version 4.4.1\n
41762    bytes     0x7ff19b160370       Copyright (C) 2019 Ultimaker\n
41762    bytes     0x7ff19b165180       This program is free software: you can redistribute it and/or modify\n
41762    bytes     0x7ff15ce36710       [WARNING] Mesh has overlapping faces!\n
                [Snip]
41762    bytes     0x7ff158005aa0       -s date="17-01-2025"
                                        -s time="20:43:12"
                                        -s print_temperature="200"
                                        -s mesh_rotation_matrix="[[1,0,0], [0,1,0], [0,0,1]]"
                                        -s material_anti_ooze_retraction_speed="5"
                                        -s support_interface_skip_height="0.3"
                                        -s support_tower_diameter="3.0"
                                        -s speed_support_interface="20"
                                        -s support_bottom_line_width="0.35"
                                        -s layer_height="0.2"
                                        -s layer_height_0="0.27"
                                        -s acceleration_print="4000"
                                        -s bridge_wall_material_flow="50"
                                        -s raft_base_line_width="0.8"
                                        -s cool_fan_enabled="True"
                                        -s cool_fan_speed_max="100"
                                        -s cool_fan_speed_0="0"

                                                   [...]

                                        -s machine_name="Ultimaker 3"
                                        -s top_bottom_pattern_0="lines"
                                        -s default_material_bed_temperature="60"
```

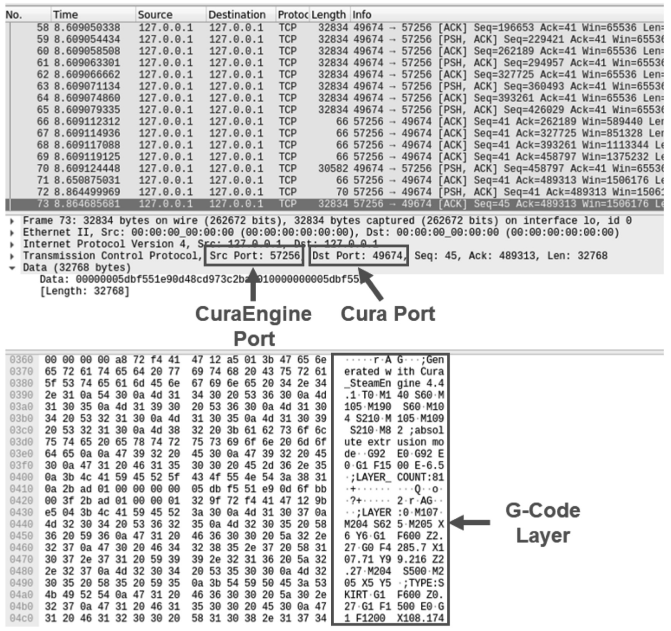
Fig. 8. The Output of the *Mem_Log* Plugin Showing the Log data after Slicing the 3D Model.

Table 4
Artifacts extracted by the plugins during Cura's lifecycle.

Memory Capture Point	Memory Artifacts (Cura)			
	Mesh Data	G-Code	Metadata	Settings
Post-Application Launch	×	×	×	×
After Printer Connection	×	×	✓	×
After 3D Model Loading	✓	×	✓	✓
During Slicing	✓	Partial	✓	✓
After Slicing Completion	✓	✓	✓	✓
After G-code Saving	✓	✓	✓	✓
After G-code Transmission	✓	✓	✓	✓
After 3D Model Removal	Original Model	×	✓	✓

settings that are represented as '-s name = "value" format.

Finally, Table 4 illustrates the persistence of the artifacts extracted by the plugins at critical points of Cura's lifecycle, including post-application launch, after establishing a network connection with the 3D printer, after loading the 3D model, during slicing the model, upon completion of slicing, after saving the G-code to disk, following the G-code transmission to the printer, and after removing the 3D model from the scene. Initially, only the setting definitions that are located in the 'definition' container of the global stack are present in memory. The printer specifications start to appear in the metadata after establishing the connection with the printer. Loading the 3D model leads to presenting its mesh data, transformation matrix, and settings that are stored in the 'user' container of the global stack. During the slicing process, the



```
No.  Time         Source     Destination Protoc Length Info
  58 8.609050338  127.0.0.1  127.0.0.1   TCP    32834 49674 → 57256 [ACK] Seq=196653 Ack=41 Win=65536 Len=
  59 8.609054434  127.0.0.1  127.0.0.1   TCP    32834 49674 → 57256 [PSH, ACK] Seq=229421 Ack=41 Win=65536
  60 8.609058508  127.0.0.1  127.0.0.1   TCP    32834 49674 → 57256 [ACK] Seq=262189 Ack=41 Win=65536 Len=
  61 8.609063301  127.0.0.1  127.0.0.1   TCP    32834 49674 → 57256 [PSH, ACK] Seq=294957 Ack=41 Win=65536
  62 8.609066662  127.0.0.1  127.0.0.1   TCP    32834 49674 → 57256 [ACK] Seq=327725 Ack=41 Win=65536 Len=
  63 8.609071134  127.0.0.1  127.0.0.1   TCP    32834 49674 → 57256 [PSH, ACK] Seq=360493 Ack=41 Win=65536
  64 8.609074860  127.0.0.1  127.0.0.1   TCP    32834 49674 → 57256 [ACK] Seq=393261 Ack=41 Win=65536 Len=
  65 8.609079335  127.0.0.1  127.0.0.1   TCP    32834 49674 → 57256 [PSH, ACK] Seq=426029 Ack=41 Win=65536
  66 8.609112312  127.0.0.1  127.0.0.1   TCP       66 57256 → 49674 [ACK] Seq=41 Ack=262189 Win=589440 Len
  67 8.609114936  127.0.0.1  127.0.0.1   TCP       66 57256 → 49674 [ACK] Seq=41 Ack=327725 Win=851328 Len
  68 8.609117088  127.0.0.1  127.0.0.1   TCP       66 57256 → 49674 [ACK] Seq=41 Ack=393261 Win=1113344 Le
  69 8.609119125  127.0.0.1  127.0.0.1   TCP       66 57256 → 49674 [ACK] Seq=41 Ack=458797 Win=1375232 Le
  70 8.609124448  127.0.0.1  127.0.0.1   TCP    30582 49674 → 57256 [PSH, ACK] Seq=458797 Ack=41 Win=65536
  71 8.650875031  127.0.0.1  127.0.0.1   TCP       66 57256 → 49674 [ACK] Seq=41 Ack=489313 Win=1506176 Le
  72 8.864499969  127.0.0.1  127.0.0.1   TCP       70 57256 → 49674 [PSH, ACK] Seq=41 Ack=489313 Win=15061
  73 8.864685681  127.0.0.1  127.0.0.1   TCP    32834 57256 → 49674 [ACK] Seq=45 Ack=489313 Win=1506176 Le
▸ Frame 73: 32834 bytes on wire (262672 bits), 32834 bytes captured (262672 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 57256, Dst Port: 49674, Seq: 45, Ack: 489313, Len: 32768
▾ Data (32768 bytes)
    Data: 00000005dbf551e90d48cd973c2ba010000000005dbf55
    [Length: 32768]

0360  00 00 00 00 a8 72 f4 41  47 12 a5 01 3b 47 65 6e   ·····r·A G···;Gen
0370  65 72 61 74 65 64 20 77  69 74 68 20 43 75 72 61   erated w ith Cura
0380  5f 53 74 65 61 6d 45 6e  67 69 6e 65 20 34 2e 34   _SteamEn gine 4.4
0390  2e 31 0a 54 30 0a 4d 31  34 30 20 53 36 30 0a 4d   .1·T0·M1 40 S60·M
03a0  31 30 35 0a 4d 31 39 30  20 53 36 30 0a 4d 31 30   105·M190  S60·M10
03b0  34 20 53 32 31 30 0a 4d  31 30 35 0a 4d 31 30 39   4 S210·M 105·M109
03c0  20 53 32 31 30 0a 4d 38  32 20 3b 61 62 73 6f 6c    S210·M8 2 ;absol
03d0  75 74 65 20 65 78 74 72  75 73 69 6f 6e 20 6d 6f   ute extr usion mo
03e0  64 65 0a 0a 47 39 32 20  45 30 0a 47 39 32 20 45   de··G92  E0·G92 E
03f0  30 0a 47 31 20 46 31 35  30 30 20 45 2d 36 2e 35   0·G1 F15 00 E-6.5
0400  0a 3b 4c 41 59 45 52 5f  43 4f 55 4e 54 3a 38 31   ·;LAYER_ COUNT:81
0410  0a 2b ad 01 00 00 00 00  05 db f5 51 e9 0d 6f bb   ·+······ ···Q··o·
0420  00 3f 2b ad 01 00 00 01  32 9f 72 f4 41 47 12 9b   ·?+····· 2·r·AG··
0430  e5 04 3b 4c 41 59 45 52  3a 30 0a 4d 31 30 37 0a   ··;LAYER :0·M107·
0440  4d 32 30 34 20 53 36 32  35 0a 4d 32 30 35 20 58   M204 S62 5·M205 X
0450  36 20 59 36 0a 47 31 20  46 36 30 30 20 5a 32 2e   6 Y6·G1  F600 Z2.
0460  32 37 0a 47 30 20 46 34  32 38 35 2e 37 20 58 31   27·G0 F4 285.7 X1
0470  30 37 2e 37 31 20 59 39  39 2e 32 31 36 20 5a 32   07.71 Y9 9.216 Z2
0480  2e 32 37 0a 4d 32 30 34  20 53 35 30 30 0a 4d 32   .27·M204  S500·M2
0490  30 35 20 58 35 20 59 35  0a 3b 54 59 50 45 3a 53   05 X5 Y5 ·;TYPE:S
04a0  4b 49 52 54 0a 47 31 20  46 36 30 30 20 5a 30 2e   KIRT·G1  F600 Z0.
04b0  32 37 0a 47 31 20 46 31  35 30 30 20 45 30 0a 47   27·G1 F1 500 E0·G
04c0  31 20 46 31 32 30 30 20  58 31 30 38 2e 31 37 34   1 F1200  X108.174
```

Fig. 9. Wireshark capture of unencrypted G-Code transmission between *Cura* and *CuraEngine* processes.

G-code is present partially, while it completely appears in memory after finishing the slicing and even after saving or sending it to the 3D printer. At the final stage, when the model is removed, only the original node data is present, whereas its copy node and G-code are deleted. Moreover, since the main persistent `SceneNode` nodes exist in memory as long as the software is active, the settings in both the definition and user containers are still available. With all the stages before the slicing, the logging data contains the slicing version and information about the connection between *Cura* and *CuraEngine* processes, while after slicing, it includes the slicing timestamp and setting values.

Fig. 9 illustrates the unencrypted G-code transmission between the *Cura* and *CuraEngine* when the user clicks on the 'Slice' Button in the Cura GUI. The security flaw resides in the local network communication between these processes, targeting traffic on localhost (127.0.0.1) to and from port 49674 (Cura's port). To demonstrate the severity of this vulnerability, we developed an attack script that runs with root privileges on Cura's local machine. The script leverages Linux's `iptables` and the `NetfilterQueue` library to intercept and modify network packets in real-time. These intercepted packets contain critical data, including printing settings and mesh data sent to *CuraEngine*, as well as G-code layers returned to Cura.

Fig. 10 demonstrates the impacts of the G-code manipulations

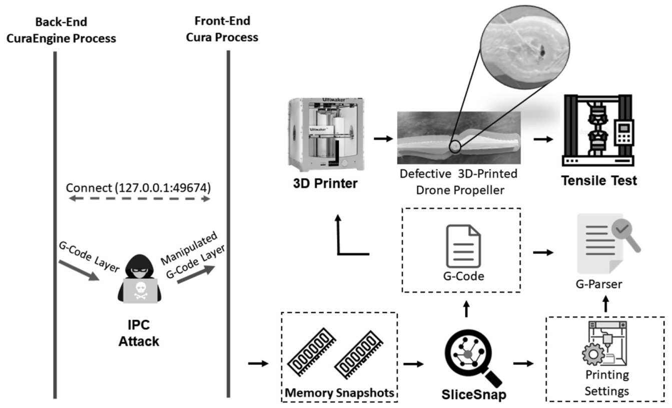
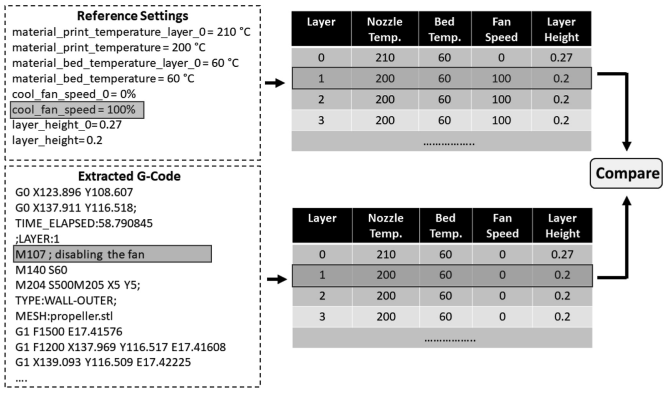


Fig. 10. Capability assessment of *G-Parser* in detecting G-Code manipulations.



Reference Settings
material_print_temperature_layer_0 = 210 °C
material_print_temperature = 200 °C
material_bed_temperature_layer_0 = 60 °C
material_bed_temperature = 60 °C
cool_fan_speed_0 = 0%
cool_fan_speed = 100%
layer_height_0 = 0.27
layer_height = 0.2

Layer	Nozzle Temp.	Bed Temp.	Fan Speed	Layer Height
0	210	60	0	0.27
1	200	60	100	0.2
2	200	60	100	0.2
3	200	60	100	0.2
....				

Extracted G-Code

```
G0 X123.896 Y108.607
G0 X137.911 Y116.518;
TIME_ELAPSED:58.790845
;LAYER:1
M107 ; disabling the fan
M140 S60
M204 S500M205 X5 Y5;
TYPE:WALL-OUTER;
MESH:propeller.stl
G1 F1500 E17.41576
G1 F1200 X137.969 Y116.517 E17.41608
G1 X139.093 Y116.509 E17.42225
....
```

Layer	Nozzle Temp.	Bed Temp.	Fan Speed	Layer Height
0	210	60	0	0.27
1	200	60	0	0.2
2	200	60	0	0.2
3	200	60	0	0.2
....				

Fig. 11. Fan speed manipulation detected by *G-Parser*.

conducted by our attack that exploit this IPC vulnerability during the slicing process. It shows how disabling the cooling fan through the injection of 'M107' commands at the beginning of each G-code layer leads to material melting in the drone propeller during printing. This manipulation is detected automatically by the *G-parser* tool, as shown in Fig. 11. It illustrates that while the nozzle temperature, bed temperature, and layer height of all the layers match their corresponding baselines, the fan speed differs from its baseline, indicating potential manipulations.

To evaluate the G-parser's ability to detect the minor setting adjustments that can affect the mechanical properties of the printed object, we sliced a rectangular bar (60 mm × 40 mm × 4 mm) multiple times using the slicing parameters mentioned in (Rais et al., 2021a). For each generated G-code, we manipulated specific settings by exploiting the found vulnerability. The nozzle temperature was changed by ± 12° at the beginning of each layer and reset to its normal value in the middle of the layers. Similarly, the fan speed was changed by ± 4%, while the layer height was decreased by 0.2 mm for even layers and reduced by the same magnitude for odd layers (Rais et al., 2022). We printed the manipulated G-code files and a benign variant, each five times, using an Ultimaker 3 3D printer. We then measured the average peak stress and strain of the five prints of each G-code file by conducting Tensile testing using the MTS Insight 30 machine.

Table 5 illustrates how these subtle changes can impact the mechanical properties of the printed objects. The benign specimens establish baseline properties with an average peak stress of 23.63 MPa and strain of 0.026 mm/mm. Raising the nozzle temperature increases the peak stress to 28.2 MPa due to enhanced layer fusion while maintaining a similar strain (0.0256 mm/mm). Conversely, reducing the temperature causes lower peak stress at 20.633 MPa with comparable strain (0.025 mm/mm). Fan speed modifications also produce notable changes. Increasing the fan speed results in 24.033 MPa peak stress with 0.029 mm/mm strain, while reducing it leads to increased stress of 28.1 MPa, attributable to slower cooling, enabling better layer fusion.

Table 5
Evaluation of G-Parser against the tensile test results.

Attack	Measure		
	Peak Stress (MPa)	Strain at Break (mm/mm)	G-Parser
Benign	23.630	0.0260	No Change
High Temp. (+12°)	28.200	0.0256	Temp. Changed
Low Temp. (−12°)	20.633	0.0250	Temp. Changed
High Fan (+4%)	24.033	0.0290	Fan Changed
Low Fan (−4%)	28.100	0.0250	Fan Changed
Layer Height (±0.2)	22.300	0.0506	Layer Changed

Finally, changing the layer height produces the most distinctive mechanical behavior. While peak stress decreases slightly to 22.3 MPa, strain increases substantially to 0.0506 mm/mm. Compared to these results, G-parser could flag all the manipulations and specify their type.

5. Conclusion and future work

This paper presented *SliceSnap*, a novel memory forensics framework to detect the production of illegal tools using 3D printing technology. *SliceSnap* focused on analyzing the volatile memory of slicing software, specifically Ultimaker Cura, the most popular Python-based slicing tool. Implemented as Volatility 3 plugins, *SliceSnap* provides automated forensic capabilities to reconstruct 3D models, extract printer-executable G-code and the entire set of settings used to generate it, provide detailed information about the used slicing software and 3D printer, and recover the logging data of the slicing software. This paper also presented the complementary analysis tool of the framework, *G-parser*, to detect malicious manipulations in the recovered G-code. The evaluation results demonstrated the 100% accuracy of *SliceSnap* in extracting the design and G-code files for diverse 3D models (e.g., firearms and keys), while successfully identifying the slicing software (e.g., version), 3D printer (e.g., name, IP), and slicing process timestamp. *G-parser* also proved its effectiveness in detecting the G-code setting changes, such as fan speed, and nozzle temperature, that were conducted by exploiting the IPC vulnerability of Cura. In the future, we plan to extend our analysis to other popular slicing software, such as PrusaSlicer and Simplify3D, and support memory samples from different operating systems.

A. Full Experimental Dataset for 3D Model Recovery

Table 6 presents the complete set of 3D models used to evaluate the extraction accuracy of the *Mem_Design* Plugin compared to their original STL and G-code files. This comprehensive dataset includes various firearm models with different geometric complexities, parts, and sizes, as well as security-critical items such as TSA Master Keys and Handcuff Keys.

Table 6
Evaluation of *Mem_Design* Plugin Extraction Accuracy across Various STL and G-Code Files

File	Baseline STL		Extracted STL		Baseline G-Code		Extracted G-Code		Jaccard	Seq. Jaccard	Levenshtein
	Vertices	Normals	Vertices	Normals	Layers	Commands	Layers	Commands	Similarity	Similarity	Distance
PX4	11712	22887	11712	22887	417	203703	417	203703	100%	100%	0
Beretta Prop Gun	382704	127568	382704	127568	356	665032	356	665032	100%	100%	0
Gun Frame	32064	10688	32064	10688	422	243433	422	243433	100%	100%	0
Lasgun	79146	26382	79146	26382	339	276048	339	276048	100%	100%	0
Colt Gun	7311	2437	7311	2437	283	233384	283	233384	100%	100%	0
XDM	61626	20542	61626	20542	731	573114	731	573114	100%	100%	0
M&P 22	1600020	533340	1600020	533340	653	1438145	653	1438145	100%	100%	0
Revolver	48234	16078	48234	16078	275	89422	275	89422	100%	100%	0
Revolver Cylinder	86670	28890	86670	28890	122	138806	122	138806	100%	100%	0
Grenade Container	84918	28306	84918	28306	284	436442	284	436442	100%	100%	0
M67 Grenade (All Parts)	45954	15318	45954	15318	336	629175	336	629175	100%	100%	0
Magazine	5886	1962	5886	1962	486	173947	486	173947	100%	100%	0
AK_47 Part	4812	1604	4812	1604	398	139724	398	139724	100%	100%	0
Glock Carbine Body (front)	48378	16126	48378	16126	397	711544	397	711544	100%	100%	0
Glock Carbine Body (Rear)	31272	10424	31272	10424	397	498383	397	498383	100%	100%	0
Single Shot Rifle (Barrel)	326424	108808	326424	108808	565	426165	565	426165	100%	100%	0
Westar Carbine (Body)	64974	21658	64974	21658	397	189256	397	189256	100%	100%	0
Westar Carbine (Muzzle)	43734	14578	43734	14578	394	123497	394	123497	100%	100%	0
Nato Bullets	74988	24996	74988	24996	71	65108	71	65108	100%	100%	0
BMG bullets	9126	3042	9126	3042	691	209876	391	209876	100%	100%	0
Bullet Belt	146556	48852	146556	48852	261	883393	261	883393	100%	100%	0
Airsoft Pistol Silencer	299256	99752	299256	99752	124	230902	124	230902	100%	100%	0
TSA Master Key	6588	2196	6588	2196	9	14564	9	14564	100%	100%	0
Handcuff Key	1422	474	1422	474	15	15744	15	15744	100%	100%	0
Lock Pick	1020	340	1020	340	9	16915	9	16915	100%	100%	0

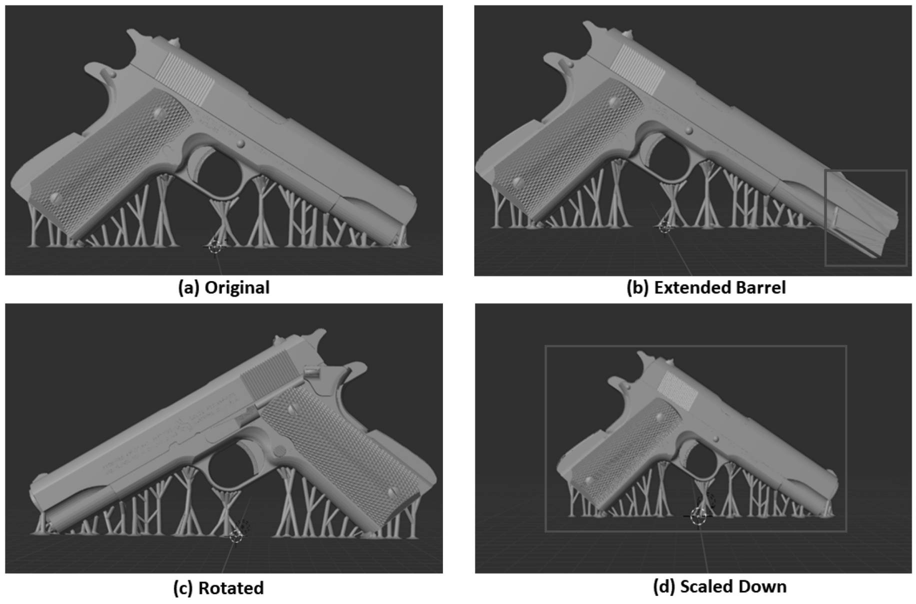


Fig. 12. Customized STL Files Derived from the Original File

B. Similarity Metrics for Transformed 3D Models

Fig. 12 shows the Beretta Prop Gun model with different modifications (extended barrel, rotation, scaling down). Our selected metrics (i.e., D2 Shape Distribution, Volume/Surface Area comparisons, and Heat Kernel Signatures) reveal the true nature of modifications while maintaining high shape similarity scores. Table 7 demonstrates how these metrics analyze the transformed models extracted by *Mem_Design*. The extended barrel model maintains high D2 Shape similarity (0.9903) with an 8.87% volume increase (ratio: 1.0887), while the Heat Kernel Signature (0.9992) confirms topological preservation. Similarly, the scaled-down model maintained shape characteristics despite significant volume reduction to 16.63% of the original size. These metrics enable investigators to identify functionally equivalent weapons despite transformations.

Table 7
Similarity Measures between Customized STL Files Extracted by the *Mem_Design* and the Original Files

File	Measure					
	D2 Shape	Volume Similarity	Volume Ratio	Surface Similarity	Surface Ratio	Heat Kernel
Original	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Extended Barrel	0.9903	0.9948	1.0052	0.9185	1.0887	0.9992
Rotated	0.9873	1.0000	1.0000	1.0000	1.0000	0.9998
Scaled Down	0.9883	0.1663	0.1663	0.3024	0.3024	0.9987

References

Ali, H., Cano, A., Ahmed, I., 2025. Machine learning-based early detection of malicious G-code manipulations in 3D printing. J. Manuf. Process. 145, 211–235.
Barradas, D., Brito, T., Duarte, D., Santos, N., Rodrigues, L., 2019. Forensic analysis of communication records of messaging applications from physical memory. Comput. Secur. 86, 484–497.
Belikovetsky, S., Yampolskiy, M., Toh, J., Gatlin, J., Elovici, Y., 2017. dr0wned–{Cyber-Physical} attack with additive manufacturing. In: 11th USENIX Workshop on Offensive Technologies (WOOT 17).
Bricknell, J., 2023. The best 3D printing slicer. https://www.cnet.com/tech/computing/the-best-3d-printing-slicer/, 2025-01-24.
Bricknell, James, 2025. Best 3D printer slicer: printing guides. https://www.ankermake.com/blogs/printing-guides/best-3d-printer-slicer, 2025-01-22.
Bryans, D., 2015. Unlocked and loaded: government censorship of 3d-printed firearms and a proposal for more reasonable regulation of 3d-printed goods. Ind. LJ 90, 901.
Budzik, G., Tomaszewski, K., Soboń, A., 2022. Opportunities for the application of 3d printing in the critical infrastructure system. Energies 15, 1656.
Carpenter, C., 2023. Bond set at $1m for Chavez HS student accused of bringing 3D-printed gun to School. https://abc13.com/student-brings-3d-gun-alexander-teran-chavez-hs-gun-incident-on-hisd-campus/13100045/, 2025-01-22.
Cho, E.E., Hein, H.H., Lynn, Z., Hla, S.J., Tran, T., 2019. Investigation on influence of infill pattern and layer thickness on mechanical strength of pla material in 3d printing technology. J. Eng. Sci. Res 3, 27–37.
Davis, M., McInnes, B., Ahmed, I., 2022. Forensic investigation of instant messaging services on linux os: discord and slack as case studies. Forensic Sci. Int.: Digit. Invest. 42, 301401.
Foundation, P.S., . Python 3.8 Documentation. https://docs.python.org/3.8/. Accessed: 2024-10-10.
Gao, Y., Li, B., Wang, W., Xu, W., Zhou, C., Jin, Z., 2018. Watching and safeguarding your 3d printer: online process monitoring against cyber-physical attacks. Proc. ACM Interact. Mob. Wearab. Ubiquit. Technol. 2, 1–27.
Garland, L., Kirui, P., An, M.K., 2024a. Enhancing autopsy with g-code file recovery: ingest module development. In: 2024 International Conference on Computer, Information and Telecommunication Systems (CITS). IEEE, pp. 1–7.
Garland, L., Neyaz, A., Varol, C., Shashidhar, N.K., 2024b. Investigating digital forensic artifacts generated from 3d printing slicing software: windows and linux analysis. Electronics 13, 2864.
Haldar, R., Mukhopadhyay, D., 2011. Levenshtein distance technique in dictionary lookup methods: an improved approach. arXiv preprint arXiv:1101.1232. https://arxiv.org/abs/1101.1232.
Hasanov, S., Alkunte, S., Rajeshirke, M., Gupta, A., Huseynov, O., Fidan, I., Alifui-Segbaya, F., Rennie, A., 2021. Review on additive manufacturing of multi-material parts: progress and challenges. J. Manufact. Mater. Proc. 6, 4.
Kurkowski, E., Van Stockum, A., Dawson, J., Taylor, C., Schulz, T., Shenoi, S., 2022. Manipulation of g-code toolpath files in 3d printers: attacks and mitigations. In: International Conference on Critical Infrastructure Protection. Springer, pp. 155–174.

Li, Z., Rathore, A.S., Song, C., Wei, S., Wang, Y., Xu, W., 2018. Printracker: Fingerprinting 3d printers using commodity scanners. In: Proceedings of the 2018 ACM Sigsac Conference on Computer and Communications Security, pp. 1306–1323.

Ma, X., Li, Z., Li, H., An, Q., Qiu, Q., Xu, W., Wang, Y., 2020. Database and benchmark for early-stage malicious activity detection in 3d printing. In: 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, pp. 494–499.

Miller, M., 2023. Ghost guns, 3D printer found in unlocked room during bust at East Harlem day care. https://www.nbcnewyork.com/news/local/crime-and-courts/ghost-guns-3d-printer-found-in-unlocked-room-during-bust-at-east-harlem-day-care/4717259/, 2025-01-22.

Moon, M., 2016. Anyone can now print out all TSA master keys. https://www.engadget.com/2016-07-28-tsa-master-key-3d-models.html, 2025-01-22.

Moore, S., Armstrong, P., McDonald, T., Yampolskiy, M., 2016. Vulnerability analysis of desktop 3d printer software. In: 2016 Resilience Week (RWS). IEEE, pp. 46–51.

Moore, S.B., Glisson, W.B., Yampolskiy, M., 2017. Implications of malicious 3d printer firmware. In: Proceedings of the 50th Hawaii International Conference on System Sciences.

Pham, G.N., Lee, S.H., Kwon, O.H., Kwon, K.R., 2018. Anti-3d weapon model detection for safe 3d printing based on convolutional neural networks and d2 shape distribution. Symmetry 10, 90.

Piltch, Avram, 2023. NY bill would require background checks to buy 3D printers, attempts to target ghost guns. https://www.tomshardware.com/news/ny-bill-bans-3d-printers, 2025-01-27.

Rais, M.H., Li, Y., Ahmed, I., 2021a. Dynamic-thermal and localized filament-kinetic attacks on fused filament fabrication based 3d printing process. Addit. Manuf. 46, 102200.

Rais, M.H., Li, Y., Ahmed, I., 2021b. Spatiotemporal g-code modeling for secure fdm-based 3d printing. In: Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems, pp. 177–186.

Rais, M., Ahsan, M., Sharma, V., Barua, R., Prins, R., Ahmed, I., 2022. Low-magnitude infill structure manipulation attacks on fff-based 3d printersa. In: Proceedings of the A16th IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection.

Rais, M.H., Ahsan, M., Ahmed, I., 2023. Fromepp: digital forensic readiness framework for material extrusion based 3d printing process. Forensic Sci. Int.: Digit. Invest. 44, 301510.

Shim, B.S., Hou, J.U., 2023. Improving estimation of layer thickness and identification of slicer for 3d printing forensics. Sensors 23, 8250.

Shim, B.S., Shin, Y.S., Park, S.W., Hou, J.U., 2021. Si3dp: source identification challenges and benchmark for consumer-level 3d printer forensics. In: Proceedings of the 29th ACM International Conference on Multimedia, pp. 1721–1729.

Shim, B.S., Choe, J.H., Hou, J.U., 2023. Source identification of 3d printer based on layered texture encoders. IEEE Trans. Multimed. 25, 8240–8252.

Stilgherrian, 2017. Fear of downloadable guns becoming a reality. https://www.zdnet.com/article/fear-of-downloadable-guns-becoming-a-reality/, 2025-01-22.

Sundby, Alex, Doyle, John, Ferris, Layla, Doan, Laura, Li, Emma, Breen, Kerry, 2024. What we know about Luigi Mangione, suspect charged in UnitedHealthcare CEO's killing. https://www.cbsnews.com/news/luigi-mangione-healthcare-ceo-shooting-what-we-know/, 2025-01-28.

Thantilage, R.D., Le Khac, N.A., 2019. Framework for the retrieval of social media and instant messaging evidence from volatile memory. In: 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE). IEEE, pp. 476–482.

Thingiverse, . Thingiverse: A Universe of Things. https://www.thingiverse.com/. Accessed: 2024-11-5.

Ultimaker, a. Cura Documentation: Wiki. https://github.com/Ultimaker/Cura/wiki. Accessed: 2024-10-22.

Ultimaker, b. CuraEngine Repository. https://github.com/Ultimaker/CuraEngine. Accessed: 2024-10-22.

Ultimaker, c. Uranium Repository. https://github.com/Ultimaker/Uranium. Accessed: 2024-10-22.

Volatility Foundation, . dwarf2json Repository. https://github.com/volatilityfoundation/dwarf2json. Accessed: 2025-01-22.

The Volatility Foundation, 2017. The Volatility framework: volatile memory artifact extraction utility framework. https://github.com/volatilityfoundation/volatility, 2025-01-24.