USENIX'23 Artifact Appendix: XCheck: Verifying Integrity of 3D Printed Patient-Specific Devices via Computing Tomography

Zhiyuan Yu[†], Yuanhaur Chang[†], Shixuan Zhai[†], Nicholas Deily[†],
Tao Ju[†], XiaoFeng Wang[§], Uday Jammalamadaka[‡], Ning Zhang[†]

[†] Washington University in St. Louis
[§] Indiana University Bloomington
[‡] Rice University

A Artifact Appendix

A.1 Abstract

XCheck is a defense system designed to verify the integrity of 3D-printed medical devices, by crosschecking their CT scans against the original designs with shape comparison techniques. Our artifact comprises the source code, CT scans (in the DICOM format), and designed model files (in the STL format). To operate the system, the user needs to run the program in the command line, providing input file paths and specifying acceptable thresholds adapted to different types of medical devices. The expected output includes interactive visualization for identifying malicious areas, and quantitative scores indicating whether the print is benign or malicious.

Given the complex computation and geometry rendering involved in XCheck, a machine with a moderate CPU and GPU, as well as memory of at least 16 GB is recommended. Please note that run-time may vary depending on the user's hardware. We have compiled a list of required dependencies into a YML configuration file.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact provided does not contain any harmful materials that may compromise machine security or pose threats to human health. The models and CT scans used by the system were not derived from real human subjects, and there is no privacy concern associated with the use of artifacts. We have taken the utmost care to ensure that the artifact meets all necessary safety standards and guidelines.

A.2.2 How to access

We have made the code, models, and CT scans publicly available on GitHub. The stable URL link pointing to the commit is `https://github.com/WUSTL-CSPL/XCheck/commits/5ee4b4820671fc215795ccb09daa70670a29e4f3`.

A.2.3 Hardware dependencies

The system can run on a machine with a moderate CPU and at least 16GB of available RAM. The system was tested stable on a desktop computer with AMD Ryzen 9 3900X 12-Core Processor, NVIDIA GeForce RTX 3070 Ti, and 32GB memory; and a laptop with Intel i9-9880H Processor, AMD Radeon Pro 5500M, and 16GB memory. Both setups are equipped with OpenGL of version 4.1. No other specific hardware is required, but the variance in hardware can lead to differences in run-time.

A.2.4 Software dependencies

XCheck was implemented in Python, and the system's environment was set up using Miniconda 4.12.0 on Ubuntu 22.0.4. The required packages include vedo, vtkplotter, open3d, point-cloud-utils, numpy, matplotlib, pydicom, seaborn, and scipy. For the detailed installation process please see Section A.3. The models and CT scans for testing are included in the artifact and do not need to be downloaded from external sources.

A.2.5 Benchmarks

The data required by the experiments are the design models (in the STL format) and CT scans (in the DICOM format) of the printed medical devices. Nine malicious models underwent geometry or material modifications provided; three malicious bone scaffolds where certain internal regions were filled in solid, a lung-on-chip with solid internal bulges, a dental guide with added sphere volume, two bone screws with enlarged thread distance and shortened non-threaded shank, a bone screw and a dental guide printed with a different material. The CT scans and design models can be found in the directory *./Geometry*.

A.3 Set-up

A.3.1 Installation

Conda or Miniconda is recommended for setting up the environment. It can be installed via the official link https://docs.conda.io/en/latest/miniconda.html and the process can differ based on the user's OS. The commands for setting up the environment with the *xcheck.yml* file are:

```
$ cd <the_path_to_the_folder>
$ conda env create -f xcheck.yml
$ conda activate xcheck
```

A.3.2 Basic Test

The basic functionality can be tested with an additional flag *-basic* in the command line. For testing, please run the following command:

```
$ python3 run.py -basic Bone_12
```

The expected output is an interactive visualization window, with an example shown in Figure 6 in the manuscript. The boxes in the top right corner are clickable, each corresponding to an analysis method. When clicking on *Registration*, the CT-reconstructed model should overlap with the design model, indicating that they are properly aligned after registration. When clicking on *Added Voxel* or *Missing Voxel*, an additional box *Colormap* becomes clickable. The colormap enables filtering out areas with lower distances, therefore highlighting the areas where geometric discrepancies are high. When clicking on *Ray-based*, it shows results where malicious regions are highlighted in red in the 3D space.

A.4 Evaluation workflow

A.4.1 Major Claims

In summary, our major claim is that XCheck can verify the integrity of printed devices in terms of geometry and material, by providing both interactive visualization applications and quantitative risk scores.

(C1): XCheck can validate the geometry integrity by measuring and visualizing discrepancies in geometric features. This is proven by the experiment (E1) described in Section 6 whose results are illustrated/reported in Figure 6 and Figure 11.

(C2): XCheck can validate the material of printed objects by comparing them to prints with similar geometry but different materials. The material verification result is reflected in the corresponding *Gamma_m* value. This is proven by the experiments (E2) whose results are illustrated in Table 3 in the appendix.

(C3): XCheck provides a quantitative risk score with gamma analysis, which aggregates geometry and material deviations. This is proven by the experiments (E3) whose results are illustrated in Table 3 in the appendix.

A.4.2 Experiments

(E1): *[Geometry Verification] [16 compute minutes + 1GB disk]:*

Preparation: Detailed instructions regarding environment setup and activation are included in Section A.3.1.

Execution: The experiments are conducted using command lines. For each of the included CT scans and models under *./Geometry*, run the command by replacing the "-f1" and "-f2" arguments for the files to be compared, and the "-etdist", "-ets", and "-etg" command to set the appropriate thresholds. For instance, to compare the CT scans of a 3D printed bone scaffold (*./Geometry/Bone_12*), that is manipulated by adding various internal solid regions, to its original model (*./Geometry/Bone.stl*), use the following command:

```
$ python3 run.py \
     -f1 "Geometry/Bone_12" \
     -f2 "Geometry/Bone.stl"
     -o Bone_12 -etdist 1.7 \
     -ets 0.05 -etg 0.005 -etm 1
```

Results: XCheck begins to first reconstruct the CT scanned DICOM images, then aligns them to the original model. Due to the complexity of models, the registration process of XCheck adopts a non-deterministic design to significantly reduce the runtime. Therefore, the iterative registration will prompt the user to visualize the aligned shapes to easily verify the registration. After the user confirms and closes the visualization window, the user can either press "enter" to proceed, or press "r" to restart the registration.

When the program run to its termination, an interactive visualization window appears, through which the user can navigate to visualize the difference between the original model and the CT-scanned reconstruction. The user can click on *Added Voxel* or *Missing Voxel* coupled with selecting *Colormap* to visualize the added/missing voxel of the printed model. During this, the user can also control different aspects of the visualization using the following sliders. (1) *Opacity-Divergence Isolation*: filters out voxels below a certain distance threshold; (2) *Colormap-Upper Bound*: paints all voxels with distance above the upper bound red, then normalizes and assigns colors with the new bounds; (3) *Colormap-Lower Bound*: paints all voxels with distance below the lower bound blue, then normalize and assign colors with the new bounds; and (4) *Point Size*: adjust voxel size ranging from 1 to 10. By customizing these parameters, users can identify whether a region of interest has been manipulated. For instance, in the verification of the model *./Geometry/Screw_5*, adjusting *Opacity* can effectively isolate part of voxels on threads and render in red, because the enlarged pitch size results in threads shifting to shank regions. Such visualization enables a more flexible and intuitive presentation of malicious regions.

Ray-based analysis leverages the principle that all geometry attacks have to alter the volume of devices. It will identify and visualize such volume in the interactive window. An example is given in *./Geometry/Bone_14* model, where the internal solid region is too small to be captured by voxel analysis but can be visualized by ray-based analysis when clicking the *Ray-based* box.

(E2): *[Material Validation] [1 compute minute]:*

Preparation: Material validation follows the geometry verification automatically.

Execution: Material validation should run automatically following the geometry verification.

Results: Material validation extracts the HU values from the CT scans, using kernel density estimation (KDE) to find the features describing its shape. This information is compared with known benign values of the same type of 3D-printed devices, to decide whether the material is tampered with. The expected result is reflected in the *Gamma_m* value, which is produced by gamma analysis and will be printed out in the terminal. The material will be considered malicious if the value is higher than 1, otherwise it is considered benign.

(E3): *[Gamma Analysis] [Less than 1 compute minute]:*

Preparation: Gamma analysis follows the geometry verification and material validation automatically.

Execution: Gamma analysis should run automatically following the previous analysis.

Results: The gamma analysis is automatically calculated at each run. If any of the four Gamma terms is less than 1, it is set to 1 to avoid value compensation when aggregating different terms (Section 5.6 in the manuscript). Each individual term of Gamma is squared and aggregated to output a total Gamma value. If the total Gamma value is no greater than 2, it is deemed benign, otherwise, it is deemed malicious. Besides, each term can be analyzed independently to infer the exact attack type; for instance, a material term *Gamma_m* larger than 1 indicates the existence of material attacks in the examined device. XCheck then proceeds to run to its termination, with the expected output of an interactive visualization window described in Section A.3.2.

```
[Computed Gamma_s: 5.130472175211942]
[Computed Gamma_d: 6.949988175539468]
[Computed Gamma_v: 1.8414619560913288]
[Computed Gamma_m: 0.43136242250997886]
[Total Computed Gamma: 8.889041709683562]
[Final Decision: Malicious]
```

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2023/`.