FLAW3D: A Trojan-Based Cyber Attack on the Physical Outcomes of Additive Manufacturing

Hammond Pearce, *Member, IEEE*, Kaushik Yanamandra, Nikhil Gupta, *Senior Member, IEEE*, and Ramesh Karri, *Fellow, IEEE*

***Abstract*—Additive manufacturing (AM) systems such as 3-D printers use inexpensive microcontrollers that rarely feature cybersecurity defenses. This is a risk, especially given the rising threat landscape within the larger digital manufacturing domain. In this work, we demonstrate this risk by presenting the design and study of a malicious Trojan (the FLAW3D bootloader) for AVR-based Marlin-compatible 3-D printers (>100 commercial models). We show that the Trojan can hide from programming tools, and even within tight design constraints (less than 1.7 KB in size), it can compromise the quality of additively manufactured prints and reduce tensile strengths by up to 50%.**

***Index Terms*—Additive manufacturing (AM), bootloader trojan, cybersecurity, cyber-physical systems (CPSs), 3D printing, firmware trojan.**

I. Introduction

ADDITIVE manufacturing (AM), also known as *3D printing*, is a technique whereby materials are deposited and fused to produce volumetric parts. In recent years, there have been considerable advances in the field, and AM is increasingly being adopted across a range of industrial and mechatronic domains (e.g., within construction [1], robotic components [2], [3], aerospace [4], and others). The advantages are numerous: Additive Manufacturing (AM) allows for the creation of complex and bespoke products without complex tooling, allows for pull-based manufacturing of products on demand rather than in advance, and rapid prototyping to iterate over product designs.

With the increasing attention on AM cyber-physical systems (CPSs), there has been an increased scrutiny on the cybersecurity of the production process. Potential vulnerabilities have been highlighted at every step of the digital and physical supply chains [5]–[8]. The impacts and implications of successful AM cyber attacks have been explored, with demonstrations showing that parts can be modified at print using malicious firmware [9]. Even subtle modifications can have insidious consequences [10] (e.g., defects being introduced in drone propellers causing them to fail prematurely in flight [11]). However, while these works highlight the potential for exploitation, they do not examine the actual pathways for doing so within an AM CPS—yet in order to craft suitable defenses for attacks on AM CPS, we must have an idea of how they might be performed. As such, in this article, we examine how a malicious modification can be introduced in a 3-D printer firmware so as to compromise print quality. This is a realistic threat due to the hidden complexity of firmware in printers from the "hobbyist" to the "commercial grade," with potentially malicious and/or insecure code already highlighted as a likely attack vector [7], [9]. While initial work studied common printer firmware (such as in Marlin and Repetier [6]) for vulnerabilities, these analyses overlook the elemental piece of the firmware/software stack, the *bootloader*.

Within the AM context, bootloaders are small firmware components fundamental to the operation of the printer software. Typically, they are not replaced/updated during the product lifecycle. Bootloaders do two tasks: 1) install the higher-level firmware into the controller memory when requested. 2) launch the installed firmware after a normal power up sequence. Crucially, bootloaders are generic, and often used across different products—especially within 3-D printer implementations, which often share hardware and software designs (e.g., those popularized via the open-source RepRap project [12]). This means a single bootloader Trojan may be utilized to target a large number of AM machines, making it an attractive attack mechanism.

In this article, we consider the bootloaders installed into the low-level controllers within commercial 3-D printers (where they might be one part of the control system) and in hobbyist 3-D printers (where they can be the only controller). We focus our study on the popular low-cost Arduino-compatible 8-bit AVR microcontrollers as these are extremely commonly utilized within 3-D printer designs to execute the on-board firmware/software (also likely due to their historical usage within RepRap [12]). However, this class of attack is not restricted to AVR-type devices, and though this article frames and demonstrates an attack around low-end desktop 3-D printers,

Manuscript received November 15, 2021; accepted May 20, 2022. Recommended by Technical Editor Jiafu Wan and Senior Editor Hong Qiao. This work was supported in part by National Science Foundation SaTC-EDU under Grant DGE-1931724. *(Corresponding author: Hammond Pearce.)*

Hammond Pearce and Ramesh Karri are with the Department of Electrical and Computer Engineering and Center for Cybersecurity, New York University, New York, NY 11201 USA (e-mail: hammond.pearce@nyu.edu; rkarri@nyu.edu).

Kaushik Yanamandra is with the Department of Mechanical and Aerospace Engineering, New York University, New York, NY 11201 USA (e-mail: vsy212@nyu.edu).

Nikhil Gupta is with the Department of Mechanical and Aerospace Engineering and Center for Cybersecurity, New York University, New York, NY 11201 USA (e-mail: ngupta@nyu.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TMECH.2022.3179713.

Digital Object Identifier 10.1109/TMECH.2022.3179713

TABLE I
SURVEY OF LITERATURE DISCUSSING ATTACKS ON AM PRINTERS

Reference	Type	Summary
Moore et al. [6]	Vulnerability analysis	Automatic checking of 3D printer firmware for vulnerabilities.
Moore et al. [9]	Specific compromise	Demonstrates the potential for malicious 3D printer firmware to introduce defects into 3D printed parts.
Belikovetsky et al. [11]	Specific compromise	Automatically adding defects to 3D printed drone blades.
ESET [13]	Specific compromise	ACAD/Medre.A worm in AutoLISP steals engineering CAD files.
Slaughter et al. [14]	Specific compromise	Temperature manipulation reliably introduces defects in metal AM.
Zeltmann et al. [10]	Specific compromise	Discusses how subtle defects can be deliberately introduced to 3D prints that reduce print quality.
Mahesh et al. [8]	Survey	Survey of cybersecurity state of the art for AM. Attack / defense modelling, taxonomies, case studies.
Prinsloo et al. [15]	Survey	Security risks within the Industry 4.0 manufacturing domain.
Graves et al. [16]	Survey	Risk survey and attack / defense modelling for AM, noting differences with traditional manufacturing.
Yampolskiy et al. [17]	Survey	Survey and analysis of potential avenues for 3d printers to be 'weaponized'.
Yampolskiy et al. [18]	Taxonomy	Specifying manipulations as tuples of {influenced elements, influences} giving {affected elements, impacts}.
Gupta et al. [5]	Survey / Taxonomy	Examining the supply chain and build process for AM cybersecurity vulnerabilities.

these mechanisms could be used to target any CPS with embedded firmware running on insecure hardware (e.g., PCB printers, IC fabrication, and test).

A. Contributions

This is the first comprehensive study of a bootloader-based attack on AM CPS. Contributions are four-fold, and organized in this article as follows: Section II presents a study of the related work and attack surface for 3-D printers given their underlying implementations. Section III presents a design space exploration of a proof-of-concept firmware Trojan FLAW3D (pronounced "flawed"), which targets Marlin-compatible AVR controllers in 3-D printers. Section IV performs a qualitative and quantitative evaluation of the Trojan by examining two different mechanisms that can compromise print quality, and in Section IV-E, we provide a discussion and walkthrough of how the Trojan could be detected and prevented. Finally, Section V concludes this article.

II. BACKGROUND AND RELATED WORK

A. Attacker Motivation and Attack Taxonomies

The two major motivations for an attack on an AM CPS are [5]: 1) IP theft via product reverse engineering and/or counterfeiting, and 2) sabotage of either the printed part or the 3-D printer producing them. Both result in financial outcomes, either in the attacker gaining proprietary (valuable) knowledge, or in reducing the reputation or value of the attacked system. Overall in the literature, a number of potential and realized attack strategies have been published, with a summary of these presented in Table I. Many of the works focus on the motivations, taxonomy, and theoretical basis for attacks, rather than the specific technical steps required to achieve them. In addition, where specific attack methodologies are detailed, attack implementation is either considered out of scope or implemented within the higher-level cyber realm (such as within [11] and [13]). Importantly, when considering attacks in this higher-level area, many defenses already exist via standard information and cybersecurity best practices (e.g., ensuring trusted software updates, firewalls, virus scanning, etc.). The state of the art is less detailed when considering attacks on the *low-level hardware* of 3-D printers. The two major works in this space come from [6], which detailed an exploration of 3-D printer firmware vulnerabilities (e.g., to denial of service and data corruption attacks), and in [9], which augments 3-D printer firmware directly to add malicious code. However, while effective, their implementation strategy of their attacks on the 3-D printer firmware has several technical shortcomings—specifically, it relies on simply changing firmware codes directly, recompiling, and redownloading. If a program's source code can be changed in your attack model (e.g., by a malafide insider), then any behavior change is possible. As a result, it is a primary focus of designers to audit their firmware (as in [6]) and detect these changes before deployment.

In this article, we wish to expand the vision for low-level attack strategies and motivate the need for more defensive mechanisms within 3-D printers. For this, we present an attack strategy which does not rely on changing the 3-D printer firmware directly. We present this through an examination of a complete attack life-cycle, performing a deep dive into the technical details of how common 3-D printers function, and include installation mechanisms, exploitation strategies and triggers, and mechanisms to avoid detection and removal.

B. Software and Hardware Trojans

In the cybersecurity domain, "Trojan Horses" refer to deliberate fault-like modifications made to a design for malicious aims, either to steal information or to cause system failure [19] (aligning them closely with the aforementioned CPS attack motivations!). Trojans, which may be created by malicious actors working in a product's design team or from compromised CAD tools used during product creation, have three essential characteristics: malicious intent, evasion detection, and activation rarity [20]. A Trojan may seek to leak cryptographic information/design files, cause digital/real-world damage, reduce operational reliability/product life-time. While typically software-based [21], there has been recent attention on hardware Trojans embedded in systems, for instance within integrated circuits [20] or encoded into PCBs [22]. Although out of scope for this work, this does raise interesting recursive possibilities within AM for PCBs (i.e., PCB printers [23]). Here, a firmware Trojan in the printer could insert hardware Trojans into produced PCBs.

C. 3-D Printer Attack Surface

Thanks largely to the quality and success of the open source 3-D printer projects, many 3-D printer implementations (hardware and software) are closely related. The RepRap 3-D

TABLE II
FEATURES OF SELECT COMMERCIAL/HOBBYIST 3-D PRINTERS

Printer	Commercial grade?	"Open"?	Marlin compatible?	Native networking?	Support F/W updates?
Stratasys Elite	Y	N	N	Y	Y
Ultimaker S5 Pro	Y	N	N	Y	Y
Ultimaker 2+	Y	Y	Y	N	Y
Makerbot Replicator Z18	Y	N	N	Y	Y
Makerbot Replicator (orig.)	N	Y	Y	N	Y
Anet A8	N	Y	Y	N	Y
Printrbot Simple	N	Y	Y	N	Y

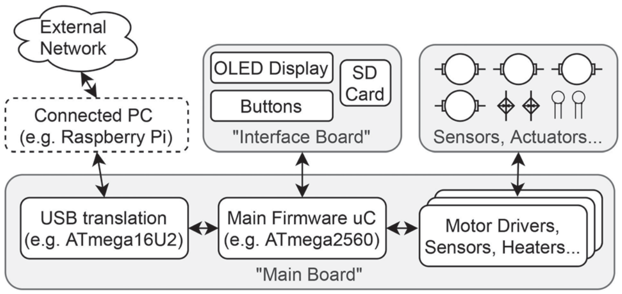


Fig. 1. Generic DM architecture for a 3-D printer.

printer project [12], itself based on early industrial fused deposition modeling (FDM) printers, has inspired at least 81 models of 3-D printers directly [24] (with the derivatives often going on to inspire further designs). Table II summarizes some examples. 3-D printers are a mix of "open" (i.e., open schematics, hardware layouts, and software) and "closed" (more likely the commercial printers with trade secrets). The (listed) printers all have some method for updating the installed firmware. In general, "open" printers are often compatible with third-party firmware—the most popular being the Arduino-based Marlin [25], which supports over 100 printer models [26]. While formerly a premium feature, increasing numbers of 3-D printers are also beginning to include native support for networking. Where this is not included, 3-D printers may be networked by wiring connections to external print servers, e.g., Octoprint [27]. A "networked" 3-D printer may just include an Octoprint server internally! Overall, the networking of 3-D printers is becoming a concern, especially when they are exposed to the wider internet (e.g., IoT scanning website Censys estimates >2500 Octoprint servers exposed globally[1]).

All 3-D printers have their low-level hardware control (e.g., control of the sensors and actuators) managed by time-predictable microcontrollers. Computationally intensive functionality such as GUIs and networking (if present) are managed by more powerful embedded or general purpose devices (e.g., as in the Ultimaker S5 Pro) or by external connected devices. An example of this kind of hierarchy can be seen in a generalized 3-D printer architecture, detailed in Fig. 1. Here, the high-level functionality (networking, slicing, etc) is provided by an external general purpose computer—for example, a Raspberry Pi, which could be running software such as Octoprint or Ultimaker Cura. This is interfaced with the printer control firmware, which is

[1]Search using [Online]. Available: https://search.censys.io/search?resource=hosts&q=octoprint

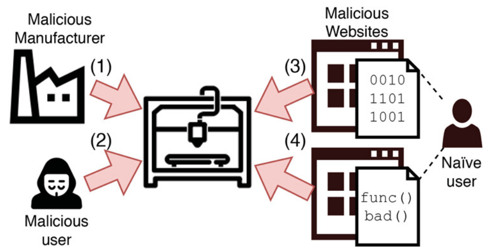


Fig. 2. Attack surface for FLAW3D.

distributed across a network of microcontrollers. Crucially, unlike in the Raspberry Pi (and other general purpose computers), where mechanisms such as "Verified Boot" [28] or "Secure Boot" [29]) can be used to ensure the security of the low-level firmware, there is no specific functionality to perform this in low-end microcontrollers. Security instead becomes the responsibility of the installed bootloader (if deemed necessary)—which may include features for cryptographically checking the firmware updates before installation [30]. Of course, if the bootloader itself is replaced (either via external hardware circuitry or via the network-connected high-end embedded systems), nothing may verify that the replacement bootloader is free of malice. Further, as the bootloader is not part of the firmware (e.g., Marlin), *even if the firmware is audited for security risks, the bootloader may be excluded from this analysis*. This is especially a concern in desktop/hobbyist AM, where bootloaders are often provided in binary form (preventing adequate auditing of their source codes) and reused across many compatible devices. We note that bootloader exploits have previously been examined for other devices (e.g., voting machines [31]). However, although bootloaders have also been highlighted as an area of concern within digital manufacturing [7], no exploit built upon them has previously been demonstrated. If an adversary has (or has had) physical access to a printer-under-attack, or is otherwise able to trick someone with physical access into installing the malicious bootloader, then four attack vectors are possible, as depicted in Fig. 2: 1) the original manufacturer of the printer or a malicious insider, 2) a malicious user with access to the printer, 3) a third-party (e.g., a website) provides a precompiled malicious bootloader or 4) a third-party provides malicious bootloader source code and a naïve user installs it without adequate auditing.

This risk is expanded by the presence of the *network* of microcontrollers. As can be seen in Fig. 1, which represents a common architecture used within 3-D printers, there are two different microcontrollers which could each individually interfere with the correct operation of the device. This only becomes more challenging given commercial and industrial-grade additive manufacturing machines, which can feature tens to hundreds of microcontrollers all running different code.

III. FLAW3D BOOTLOADER

A. Goals

In this section, we discuss the design space exploration for the creation of a new firmware Trojan called FLAW3D. It will target

AVR-based desktop 3-D printers which run the Marlin printer firmware. We note that the attack will not target any specific feature of any printer, rather, it aims to misuse features from the underlying AVR microcontrollers and the operating Marlin firmware. Though we specifically target AVRs, we also note that the general methodology in this section can be used to target other printers using other microcontrollers, including at the "commercial grade" by malafide insiders—the only requirement being the usage of unsecured bootloaders within their designs.

Overall, we will consider an adversary that aims to sabotage a design firm by reducing the quality of printed designs. In order to achieve this, we seek to give the Trojan the ability to both *relocate* and *remove* printed material.

We must also work within constraints: specifically, the Trojan must not increase the compiled size of the bootloader beyond the boot flash size limit. For example, in the ATmega2560, a microcontroller commonly used in 3-D printers, this is just 8192 Bytes, of which 5786 B are already taken by the existing bootloader code, leaving approx. 2406 B for the Trojan).

B. Arduino-Compatible Bootloaders for AVR

Marlin is installed on AVR-based 3-D printers via compatible software (e.g., the Arduino IDE, the Ultimaker Cura slicer) running on a secondary machine (e.g., a Raspberrry Pi, or a general purpose Windows or Linux computer). The gatekeeper for this process is the AVR-based bootloader, which resides on the target microcontroller. Upon power-up it executes a simple state machine, which initializes a UART communication peripheral and awaits valid bootloader commands. If no commands arrive before a timeout, the existing main firmware is initialized if available. If a command does arrive, the bootloader will execute it. These commands, which are based on a subset of the STK500 standard [32], include instructions to read and write flash and EEPROM memory. Three observations are important.

Observation One: though both bootloaders and applications are installed into the microcontroller flash memory, they do not run simultaneously. Bootloaders run first, eventually loading the main firmware. When this happens, bootloaders are entirely unloaded, with the stack and global memory reset and reconfigured for the main application.

Observation Two: for bootloaders to install the main firmware, all memory values for the binary must pass through them both during upload (installation) and download (verification). A secure bootloader could perform cryptographic and data integrity checks, but regular Arduino-compatible bootloaders do not. Instead, data verification is managed by the off-chip toolchain reading all memory addresses after installation and ensuring they match the expected.

Observation Three: though Arduino-compatible bootloaders do not tend to utilize interrupts, the underlying hardware supports this. Interrupts function by preempting control flow to specific locations in memory (known as interrupt vector tables). As the bootloader and the main firmware are distinct applications, they must have different vector tables. Hence, the AVR architecture supports changing the address of the vector table using a special control register `IVSEL`.

```
//inject code in this vector
ISR(..._vect) {

  ... // prologue injection here

  //call application's vector address then run CLI
  // this is important as the app. ISR will return
  // using RETI, which re-enables interrupts after
  // the next instruction - a CLI prevents this
  asm volatile("call [..._vect_addr]\n\t"
               "cli\n\t");

  ... // epilogue injection here

} // our ISR will return with another RETI
```

Listing 1. Structuring an AVR bootloader ISR to 'inject' code.

```
void main(void) {
  //create a copy of MCUCR
  char temp;
  temp = MCUCR;
  // Enable change of Interrupt Vector location
  MCUCR = temp | (1<<IVCE);
  // Point interrupts at Bootloader Flash section
  MCUCR = temp | (1<<IVSEL);

  ... //rest of the bootloader
}
```

Listing 2. Two-step process changes `IVSEL` in `main()`.

C. Design of a Generic Arduino-Compatible Trojan for AVR

Using the bootloader source code provided by Arduino at [33] as a start point, we now craft a Trojan for an ATmega2560, noting that the steps for other common AVR microcontrollers are largely the same. First, in order for the Trojan to function it needs to be able to inject instructions into the program executed by the main firmware. Based on *Observation Three*, this is achieved via the interrupt vector table select register `IVSEL`. By default this register is configured to make interrupts jump to the vector table associated with the main firmware. If changed, the machine will instead jump to *bootloader* program space upon an interrupt occurring. Crucially, the startup code (i.e., the code that runs "before `main()`") generated by the AVR compiler `avr-gcc` *does not check or set the* `IVSEL` *register* — nor does the firmware we are interested in hijacking (Marlin). This means that if we define our own bootloader interrupt service routines (ISRs), and set `IVSEL` before booting the main application, *the bootloader ISRs will replace the main application's*.

However, if the main application defines ISRs, and those ISRs are never called (because the hardware is calling the wrong interrupt vectors) then the presence of the Trojan will be easily noticed. Thus, the Trojan must embed within its ISRs calls to the main application ISRs (using the addresses of the original vector table). In this way, it *wraps* the application ISRs—allowing injection of both *prologue* and *epilogue* instructions to each routine. Code to perform these injections, using the `avr-gcc` compiler, is presented in Listings 1 and 2.

While this structure allows for the injection of instructions, declaring state (variables) that will persist outside of the ISRs is a separate, more difficult issue, as the processor memory is reinitialized by the main application (*Observation One*). In other

```
1  00000000 <__vectors>:
2    0: 0c 94 e7 10  jmp 0x39dc ; jump <__dtors_end>
3    ... ; ... etc
4
5  000039dc <__dtors_end>:
6    ; clear SREG
7    39dc: 11 24   eor  r1, r1
8    39de: 1f be   out  0x3f, r1
9    ; code to set SPH/SPL
10   39e0: cf ef   ldi r28, 0xFF ; load 0xFF to R28
11   39e2: d1 e2   ldi r29, 0x21 ; load 0x21 to R29
12   39e4: de bf   out 0x3e, r29 ; set SPH to R29 (0x21)
13   39e6: cd bf   out 0x3d, r28 ; set SPL to R28 (0xFF)
14   ... ; ... etc
```

Listing 3. Disassembled Marlin ATmega2560 startup code.

words, the Trojan has no safe way of storing global or static variables outside of ISR invocations.

To resolve this, we consider the implementation of the AVR's Harvard-style memory architecture. The data memory, which is separated from the instruction memory, is partitioned into register space (in the first 256 Bytes) and general RAM. In the general RAM, static objects and global variables are placed in the low addresses by the C compiler, and the stack (which stores local variables and function return addresses) grows from the highest address *downwards*.

To keep track of the stack's position, two registers are provided—`SPH` and `SPL` (for the high and low byte of the 16-bit address, respectively). During the startup code of an AVR application one of the first tasks that is performed is reinitialization of these two registers. An example which sets `SPH/SPL` to `0x21/0xFF` can be seen in the ATmega2560 startup disassembly in Listing 3. Note that, given the stack grows downwards, if the values loaded into `SPH` and `SPL` are decreased, then the addresses above their new value are excluded from the stack. This would free them for the Trojan.

Now, recall *Observation Two:* all instructions making up the main application are passed through the bootloader during installation. That is, the bootloader is responsible for receiving the compiled application binary over UART and saving it into flash memory. In addition, though the startup code, which initializes `SPH` and `SPL` may be located unpredictably within the binary, the *specific* instructions that make it up *do not change from application to application*, and further, are usually located at the first jump from the vector at program address `0000` (the reset vector). This means that during the installation loop, the bootloader can scan for the pattern of 8 Bytes, which sets `SPH/SPL` (Lines 10–13 in Listing 3) and then *alter those bytes that represent the data address before saving the program into application flash*. To minimize detection, the Trojan should change the value only slightly (otherwise the application has a higher chance of running out of memory unexpectedly during operation). While the exact amount is adjustable, in this work, we choose to exclude 15 Bytes from the stack, reserving two bytes for use as *canary values* (to ensure that if the running application overwrites the memory it can be detected), and presenting 13 Bytes to the Trojan payload for use in storing global state. The bootloader thus alters the instruction `ldi r28, 0xFF` to `ldi r28, 0xF0` (subtracting 15) prior to saving it to the flash memory.

While the edit to the program binary can be detected in the normal case, now recall again *Observation Two*: specifically

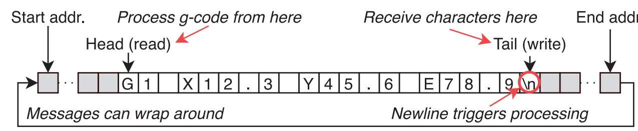


Fig. 3. Ring Buffer implementation in Marlin.

that the verification of the binary is also done via reading out the saved binary using the same bootloader. This means that by simply performing the edit step in reverse, the bootloader can change the binary values back to the expected when the programming tool attempts to ensure that the program has been uploaded correctly. This means that based on the described Trojan design, *programming tools which rely on the bootloader to upload, download, and verify the installed program cannot detect the malicious modifications*.

D. Design of FLAW3D

Returning to the original goal of subverting printing quality, consider now the flow of information in a 3-D printer. Print commands, which detail control sequences for the motors, extruders, and heaters of the printer, are specified in textual `g-code` language. These originate from a computer running a slicer program that converts 3-D computer aided design models into the `g-code`. From the point of view of the controller running the printer, the commands arrive character-by-character via the UART peripheral.

It is thus the goal of FLAW3D to edit this incoming `g-code`, using the Trojan framework from Section III-C as the starting point. Using the ISR code injection mechanism, the reception of valid `g-code` can be compromised and intercepted "in-flight." The Trojan can then edit received commands before they are processed by the main application.

While it might appear that this can be done by interfering with the UART peripheral, (e.g., editing the received character during the injection), two hardware constraints prevent this: 1) the UART RX register is *read-only*, and 2) reading from the UART RX register has side effects; once read, it unlocks the hardware for further reception. We thus instead consider how the bootloader can interfere with the higher level firmware (Marlin). Marlin, prior to processing the received `g-code` with the main process loop, uses its UART RX ISR to store received characters in a *ring buffer* (depicted in Fig. 3).

As the AVR has no memory protection, the Trojan can access the entire memory space from the compromised ISRs. If the location of the ring buffer can be deduced, the Trojan can read and edit the `g-code` commands prior to their processing by the main application. To accomplish this, consider *Observation Two*. As the Trojan can access the flash memory of the microcontroller, it can scan the UART ISR of Marlin, revealing two distinctive `lds` commands near the start (Listing 4).

The addresses in these two instructions correspond to the location in memory of the *head* and *tail* pointers of the ring buffer. By default the compiled Marlin firmware's data structure layout will place these pointers 128 Bytes after the ring buffer

```
00000000 <__vectors>:
  ... ; ... etc
  50: 0c 94 b7 83  jmp 0x1076e ; jump to UART RX ISR
  ... ; ... etc

0001076e <__vector_20>:
  ... ; ... etc
  10786: 20 91 24 03   lds r18, 0x0324
  1078a: e0 91 23 03   lds r30, 0x0323
  ... ; ... etc
```

Listing 4. Disassembly of UART RX ISR in Marlin on AVR.

itself. This means that the smaller of the two addresses (e.g. 0x0323) minus 128 gives the root address in memory of the ring buffer, where the incoming `g-code` is stored. FLAW3D thus encodes this behavior as a function `find_ring_buffer()` to do this task automatically prior to launching the main application firmware. The function performs this by traversing the binary of the main application, starting from the (constant) UART RX ISR vector location, and following the program jumps and the linear path of execution until it finds these back-to-back `lds` commands. If it does not identify them within 256 instructions, it aborts, and the Trojan is rendered dormant. If it succeeds, it stores the head pointer, tail pointer, and root address in the global state variables that we established earlier (in the top 15 B of the AVR memory). FLAW3D can then use these pointers with string manipulation code injected as an epilogue of the main application UART RX ISR: and now, incoming `g-code` can be edited.

As standard string manipulation in C (performed by functions such as `sscanf` to read out variables and `sprintf` to rewrite them) are too large to use within the context of a bootloader, FLAW3D relies on a simple embedded state machine. This examines incoming `g-code` strings character by character, and can internally convert received ASCII-encoded floating point values into integer-type fixed-point notation. If the bootloader detects that a target value to edit is arriving, it suppresses the Marlin firmware's normal behavior by editing the ring buffer head pointer addresses to hide the incoming characters. Then, once the target value has been entirely received, the bootloader can process and edit it before restoring the correct pointer value and allowing Marlin to detect and process the command.

IV. Inducing Defects With FLAW3D

A. Overview

FLAW3D scans and alters incoming `g-code` before processing by Marlin. Given the restricted space for Trojan code (e.g., ~2406 Bytes on ATmega2560), the edits need to be simple. Complex edits may also cause noticeable delays. Given these constraints, we present two Trojan methodologies, with code compiled using `avr-gcc` version 5.4.0 with optimization `-Os`. To measure the impact of the Trojans on print quality, strength tests were performed using the tensile test specimen design E8 from ASTM A370-20 [34] (see Fig. 4). This is performed destructively using an Instron 4467 universal test system. Samples were printed in PLA with two different sets of common slicing parameters using two different anonymized commercially available AVR-based 3-D printers, which use Marlin internally

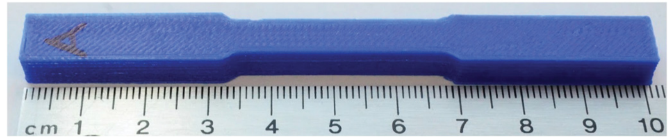


Fig. 4. Example control group test specimen.

TABLE III
Printers and Print Settings

Printer info.		(Print) Slicing settings			
ID	Cost (New) (USD)	Layer height (mm)	Line width (mm)	Infill strategy	Infill line distance (mm)
A	2499	0.15	0.35	Cubic (18%)	5.83
B	599	0.15	0.4	Grid (10%)	8.0

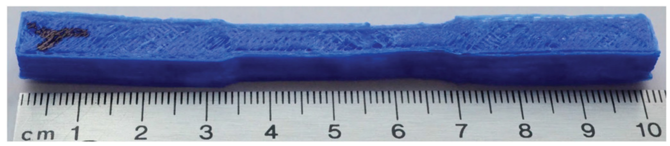


Fig. 5. Example 50% material reduction.

(see Table III). Design E8 was chosen for two reasons: 1) it is relatively small (minimizing the change for random deviations within the prints that effect print quality [35]), and 2) given the geometry and PLA print material the specimens do not slip within the tensile test machine grips.

While it is customary to report strength in the form of maximum load sustained divided by the cross-section area, as the Trojan modifies the printer `g-code` the exact cross-sectional area may vary from test to test. Hence, in this article, the normalized maximum load sustained by the specimens before failure is compared, as all specimens have the same origin test geometry. Though, it is deterministic, to minimize printer "noise" each sample is printed five times and test results averaged.

B. Trojan Attack 1: Material Reduction

This Trojan attack reduces the amount of printed material. FLAW3D uniformly scans for the `G1` commands (linear move) in the incoming `g-code`, which include the extrusion command (character `E`). Then, the extrusion value is decreased by some percentage. For instance, a command `G1 X2 Y3 E4`, which moves to (X,Y) (2,3) and extrudes 4 mm of filament can be edited to `G1 X2 Y3 E2` to reduce the material by 50%. While this reduces the maximum tensile strength of the specimen, the attack is easily detectable, both by weight tests and by visual inspection in severe cases (see Fig. 5).

The normalized test results are given in Figs. 6 and 7 for extrusion reduction between 0 and 50% (step size 10%). As can be seen, material reduction reduces the mass and maximum tensile load fairly linearly, i.e., as you increase the effect of the Trojan by reducing the amount of extruded filament, the mass and the supported maximum load of each specimen both decrease. The attack increases the bootloader size from 5786 to 7422 B, an increase of 1636 B.

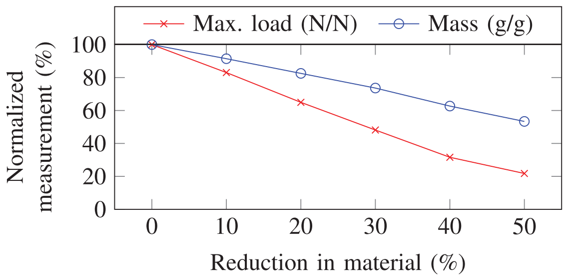


Fig. 6. *Printer A* - Max. tensile load versus material reduction.

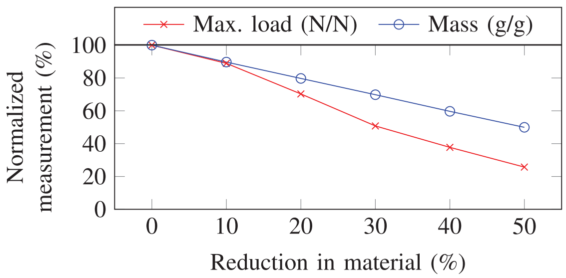


Fig. 7. *Printer B* - Max. tensile load versus material reduction.

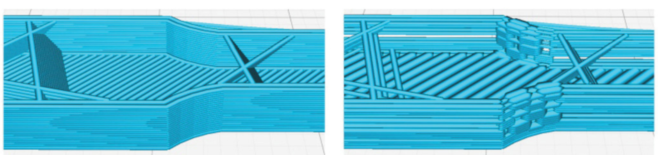


Fig. 8. Simulation of (L) original and (R) 1-in-2 relocation.

C. Trojan Attack 2: Material Relocation

In the `g-code`, the extrusion values are presented within an *absolute* frame of reference. This means that if one extrusion is removed, the next extrusion will deposit extra material to keep the values consistent. Consider three back-to-back commands `(G1 X1 Y2 E3)`, `(G1 X2 Y3 E4)`, and `(G1 X3 Y4 E5)`. The total extruded material is 3 mm after the first command, 4 mm after the second (it deposits 1 mm), and 5 mm after the third. If the Trojan alters the second command to `(G0 X2 Y3)`, no material is extruded during its execution, although the head continues along the same route. Crucially, the third command now deposits 2 mm and the total material used remains 5 mm.

This is the basis of the second attack, which scans for `G1` linear movement commands with extrusions, and converts a subset (either 1-in-4, 1-in-3, or 1-in-2) into `G0` linear movement commands with no extrusions. To further reduce the visibility of the attack, we also add a new activation trigger: we preclude the Trojan from activating until 25% of the part is printed, and deactivate it after 75% is printed. For this, we track the `M73` commands which are used to update the percentage remaining on printer displays. Fig. 8 shows the consequences on a specimen cross section.

The results of this attack are depicted in Fig. 9. All printed objects remain within 1–2% of the control masses, and despite

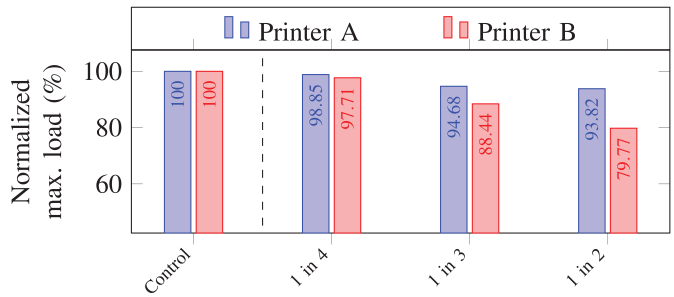


Fig. 9. Material relocation Trojan.

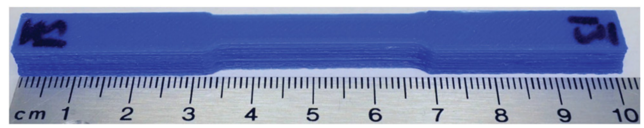


Fig. 10. Example 1-in-2 material relocation. No obvious changes are visible w.r.t. specimen quality.

the edits, the parts are visually similar (example: Fig. 10). Here, we observe that as we increase the frequency of the compromised lines, the strength of the printed parts reduce. While all edits weakened the parts, the attack on *Printer B* was more effective than the attack on *Printer A*, which is likely due to the different slicing infill strategies (10% compared to 18%). This attack increases the bootloader size to 6986 B, a 1200 B increase, smaller than the previous attack due to the simpler string manipulations even with `M73` tracking.

D. Discussion

Different attacks can change the strength profiles of printed parts in different ways. Reducing the printed material is simple and reliable, but noticeable (e.g., via weight). Material relocation is harder to detect, but less consistent, as evident from the differing effectiveness on *Printer A* compared to *Printer B*. Given its small size, (<1.4KB) the attack surface for such Trojans within AM CPS is enormous, since a large number of microcontrollers are present in complex "commercial-grade" systems, which have large feature sets. In addition, the Trojan could be made elusive by altering its trigger. While in this study the attack activated in every print, it would be insidious to activate rarely, with the intent of lowering the average quality of service/institutional reputation. Parts that are unpredictably faulty can get through certain quality controls, and if targeted at critical products could have catastrophic consequences (for instance consider a Trojan within 3-D printers for aerospace components [4] or bespoke medical parts [36]). Finally, while in this work, we limited the scope to only Printer A and Printer B, other Marlin-compatible AVR-based printers may also be vulnerable. As previously noted in Section II-C, this is over 100 known models of printer [26]. To further generalize the study, work could also be performed examining further possible defect mechanisms (i.e., different `g-code` edit strategies), including an examination of how different geometries react to different

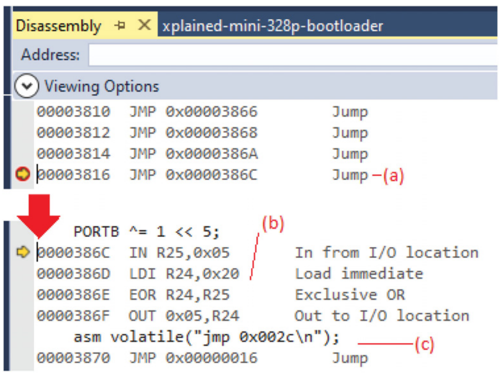


Fig. 11. Debugger observing code injection ISR.

edits. Though, we focused on AM (3-D printers), attacks could also be studied on devices in other manufacturing domains, e.g., CNC devices (which also utilize `g-code`) or machines for IC fabrication and test, as many of these will also utilize low-cost and vulnerable microcontrollers in their designs.

E. Detecting FLAW3D

Since this is a bootloader Trojan, it is difficult to detect at the software level—especially since it has the ability to alter high-level firmware prior to its installation. In other words, even if the defenses are encoded within user applications, they could be detected and disabled prior to activation.

However, as FLAW3D alters the printer's behavior, it is not impossible to detect. Side channels can be monitored with external devices such as cameras and microphones from an arms length—though this would be intricate as the Trojan effects extrusions and not the head movement. Careful measurement of the ISR delays could reveal the presence of injected instructions. External programming tools could be utilised—if monitored with a debugger, the alternate ISR jumps can be noticed. An example of this is presented in Fig. 11, which shows how the Atmel Studio debugger for an ATmega328P can be used to deduce a bootloader performing a code injection using the methodology in Section III-C. Here, the Trojan is designed to flip the fifth bit of `PORTB` whenever the target interrupt occurs. Within the debugger, we can observe that the assembly at (a) is the interrupt target, which then jumps to the injection exploit at (b), flipping a bit in `PORTB` before the correct ISR is jumped to at (c).

That said, in order to perform this test the embedded system must be designed to allow access to the AVR's debugging (JTAG) port, and toolchains must be utilized that support first-class debugging (e.g., not the original Arduino IDE, which offers no live debugging facilities). This access is not guaranteed, especially given the design constraints of the chosen AVR device and embedded system. For instance, on the ATmega2560, the JTAG pins are multiplexed with four of the ADC inputs. This means that when the ADCs are in use then the JTAG port is rendered unavailable. This is the case in *Printer A*. An alternative that does not require the debugging port is to export the bootloader itself from the flash memory of the AVR. As this is performed using the programming hardware of the AVR chip itself, there is no way for the bootloader to edit itself before download. Then, the bootloader may be disassembled/decompiled and audited for malicious behavior, though this will require specialist tools and knowledge.

As low-end microcontrollers do not support features such as secure boot, further prevention within the hardware can come if they are exchanged for something with more features. Policies surrounding firmware installation/inspection (including bootloaders) could be introduced to mitigate the attack vectors in Section II-C. A simple low-cost solution, although it would limit product flexibility, would be to embed bootloaders as nonreprogrammble ROMs, or to utilize *fuse* bits to disable reprogramming the bootloader code, such that the bootloader could not be replaced after product creation. Overall, we believe that this work motivates the inclusion of a trusted execution environment [37] within AM printers.

V. Conclusion

The cybersecurity of AM CPS is important. We examine a case study in detail, presenting the design space exploration of a bootloader Trojan, including (a) methodology for inclusion, (b) evasion of detection, (c) trigger customization, and (d) malicious payload. Even within tight constraints (both attacks less than 1.7 KB in size!), we were able to craft attacks, which lowered the strength of printed parts by up to 50%. Though, we frame the issue around desktop AM devices, we stress that the issues we highlight in this article are not restricted to these models. Indeed, the more complex an AM CPS is, the greater the attack surface for embedded Trojans, and "commercial grade"/"industrial scale" 3-D printers have complex internal networks of microcontrollers and embedded systems. This work serves as a reminder that these components can hide malicious surprises, especially when they support complex and powerful configuration options that can be misused. It takes only one component to be infected by a malafide insider or malicious third party with access to cause insidious and catastrophic consequences. We believe that procedures for bootloader and firmware verification should be introduced across the AM CPS space, alongside potential automatic monitoring (e.g., via side channels) which could be developed to detect and flag anomalous behavior.

Acknowledgment

The authors would like to thank G. Mac for his help with the CAD modeling.

Open Source Access: A simplified version of FLAW3D (to preserve the anonymity of the target devices) is available online at [38].

References

[1] O. Lakhal, T. Chettibi, A. Belarouci, G. Dherbomez, and R. Merzouki, "Robotized additive manufacturing of funicular architectural geometries based on building materials," *IEEE/ASME Mechatronics*, vol. 25, no. 5, pp. 2387–2397, Oct. 2020.

[2] Y. Wei, Y. Chen, Y. Yang, and Y. Li, "Novel design and 3-D printing of nonassembly controllable pneumatic robots," *IEEE/ASME Mechatronics*, vol. 21, no. 2, pp. 649–659, Apr. 2016.

[3] C. Tawk, H. Zhou, E. Sariyildiz, M. in het Panhuis, G. M. Spinks, and G. Alici, "Design, modeling, and control of a 3D printed monolithic soft robotic finger with embedded pneumatic sensing chambers," *IEEE/ASME Mechatronics*, vol. 26, no. 2, pp. 876–887, Apr. 2021.

[4] L. Nickels, "AM and aerospace: An ideal combination," *Metal Powder*, vol. 70, no. 6, pp. 300–303, Nov. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026065715003446

[5] 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. 47322–47333, 2020.

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

[7] D. Wu *et al.*, "Cybersecurity for digital manufacturing," *J. Manuf. Syst.*, vol. 48, pp. 3–12, Jul. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612518300396

[8] P. Mahesh *et al.*, "A survey of cybersecurity of digital manufacturing," *Proc. IEEE*, vol. 109, no. 4, pp. 495–516, Apr. 2021.

[9] S. B. Moore, W. B. Glisson, and M. Yampolskiy, "Implications of malicious 3D printer firmware," in *Proc. Hawaii Int. Conf. Syst. Sci. (HICSS-50)*, pp. 6089–6098, Jan. 2017. [Online]. Available: https://aisel.aisnet.org/hicss-50/st/digital_forensics/5

[10] S. E. Zeltmann, N. Gupta, N. G. Tsoutsos, M. Maniatakos, J. Rajendran, and R. Karri, "Manufacturing and security challenges in 3D printing," *JOM*, vol. 68, no. 7, pp. 1872–1881, Jul. 2016. [Online]. Available: https://doi.org/10.1007/s11837-016-1937-7

[11] S. Belikovetsky, M. Yampolskiy, J. Toh, J. Gatlin, and Y. Elovici, "dr0wned cyber-physical attack with additive manufacturing," in *Proc. 11th USENIX Workshop Offensive*, 2017. [Online]. Available: https://www.usenix.org/conference/woot17/workshop-program/presentation/belikovetsky

[12] R. Jones *et al.*, "RepRap the replicating rapid prototyper," *Robotica*, vol. 29, no. 1, pp. 177–191, Jan. 2011.

[13] ESET, "ACAD/Medre.A," Jun. 2012. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/200x/white-papers/ESET_ACAD_Medre_A_whitepaper.pdf

[14] A. Slaughter, M. Yampolskiy, M. Matthews, W. E. King, G. Guss, and Y. Elovici, "How to ensure bad quality in metal additive manufacturing: In-situ infrared thermography from the security perspective," in *Proc. 12th Int. Conf. Availability, Rel. Secur.*, New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 1–10. [Online]. Available: https://doi.org/10.1145/3098954.3107011

[15] J. Prinsloo, S. Sinha, and B. von Solms, "A review of industry 4.0 manufacturing process security risks," *Appl. Sci.*, vol. 9, no. 23, Jan. 2019, Art. no. 5105, number: 23 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2076-3417/9/23/5105

[16] 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. 103833–103853, 2019.

[17] M. Yampolskiy, A. Skjellum, M. Kretzschmar, R. A. Overfelt, K. R. Sloan, and A. Yasinsac, "Using 3D printers as weapons," *Int. J. Crit. Infrastruct. Protection*, vol. 14, pp. 58–71, Sep. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874548215300330

[18] M. Yampolskiy, P. Horváth, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits, "A language for describing attacks on cyber-physical systems," *Int. J. Crit. Infrastruct. Protection*, vol. 8, pp. 40–52, Jan. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874548214000602

[19] J. Vosatka, "Introduction to hardware trojans," in *The Hardware Trojan War: Attacks, Myths, and Defenses*, S. Bhunia and M. M. Tehranipoor, Eds. Cham:Springer International Publishing, 2018, pp. 15–51. [Online]. Available: https://doi.org/10.1007/978-3-319-68511-3_2

[20] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[21] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi, "A taxonomy of computer program security flaws," *ACM Comput. Surv.*, vol. 26, no. 3, pp. 211–254, Sep. 1994. [Online]. Available: https://doi.org/10.1145/185403.185412

[22] S. Ghosh, A. Basak, and S. Bhunia, "How secure are printed circuit boards against trojan attacks?," *IEEE Des. Test*, vol. 32, no. 2, pp. 7–16, Apr. 2015.

[23] BotFactory Inc., "BotFactory SV2 PCB printer," 2021. [Online]. Available: https://www.botfactory.co/page/botfactory-sv2-pcb-printer

[24] RepRap, "RepRap options - RepRap," 2021. [Online]. Available: https://reprap.org/wiki/RepRap_Options

[25] S. Lahteine, R. Neufeld, C. Pepper, B. Kuhn, and E. V. D. Zalm, "Home | Marlin Firmware," 2021. [Online]. Available: https://marlinfw.org/

[26] MarlinFirmware, "Release 2.0.5.3 MarlinFirmware/Configurations," Mar. 2020. [Online]. Available: https://github.com/MarlinFirmware/Configurations/releases/tag/2.0.5.3

[27] G. Häußge, "OctoPrint.org," 2021. [Online]. Available: https://octoprint.org/

[28] T. Rigas, "Enabling verified boot on raspberry Pi 3," Apr. 2019. [Online]. Available: https://blog.nviso.eu/2019/04/01/enabling-verified-boot-on-raspberry-pi-3/

[29] R. Wilkins and B. Richardson, "UEFI secure boot in modern computer security solutions," UEFI Forum, Tech. Rep., Sep. 2013. [Online]. Available: https://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf

[30] D. Lau, "Secure bootloader implementation," Freescale Semicond., Tech. Rep. AN4605, Oct. 2012. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN4605.pdf

[31] A. J. Feldman, J. A. Halderman, and E. W. Felten, "Security analysis of the diebold AccuVote-TS voting machine," in *Proc. 2006 USENIX/ACCURATE Electron. Voting Technol. Workshop*, Vancouver, B.C., Canada, Aug. 2006. [Online]. Available: https://www.usenix.org/legacy/event/evt07/tech/full_papers/feldman/feldman_html/

[32] "STK500 communication protocol," Atmel (Microchip), Tech. Rep. AN2591 / AVR068, Jun. 2006. [Online]. Available: http://ww1.microchip.com/downloads/en/Appnotes/doc2591.pdf

[33] Arduino, "arduino/Arduino-stk500v2-bootloader," Apr. 2021. [Online]. Available: https://github.com/arduino/Arduino-stk500v2-bootloader

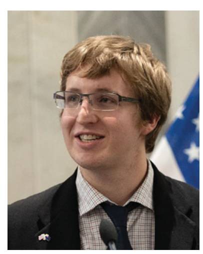
[34] *ASTM A370-20, Standard Test Methods and Definitions for Mechanical Testing of Steel Products*, ASTM Int., West Conshohocken, PA, Tech. Rep.ASTM A370-20, 2020.

[35] G. Mac, H. Pearce, R. Karri, and N. Gupta, "Uncertainty quantification in dimensions dataset of additive manufactured NIST standard test artifact," *Data Brief*, vol. 38, Oct. 2021, Art. no. 107286. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340921005709

[36] J. K. Placone and A. J. Engler, "Recent advances in extrusion-based 3D printing for biomedical applications," *Adv. Healthcare Mater.*, vol. 7, no. 8, 2018, Art. no. 1701161.

[37] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2015, vol. 1, pp. 57–64.

[38] H. Pearce, "kiwih/328p-flaw3d-bootloader," Apr. 2021. [Online]. Available: https://github.com/kiwih/328p-flaw3d-bootloader



Hammond Pearce (Member, IEEE) received the B.E. (Hons.) degree in computer systems engineering and the Ph.D. degree in computer systems engineering both from the University of Auckland, Auckland, New Zealand.

From 2020, he has worked as a Post-doctoral Research Associate with New York University, Brooklyn, NY, USA, with the Department of Electrical and Computer Engineering and with the NYU Center for Cybersecurity. His research focus is in industrial cybersecurity, including in additive manufacturing and in industrial informatics. In 2019, he took part in the NASA International Internship Programme and worked at NASA Ames in California. Other research interests include IoT, CPS, compilers, and AI/ML.

Kaushik Yanamandra received the master's degree in mechanical engineering specializing in mechanics and structural systems with New York University, Brooklyn, NY, USA, in 2018. He is currently toward the Ph.D. degree in mechanical engineering with New York University.

His research work is focused on development of lightweight advanced composites of metals for dynamic loading conditions. His research work is trying address shortcoming of lead acid battery through developing novel lead electrode for use in lead acid battery. Also, through his research, he is implementing machine learning models for material characterization.

Nikhil Gupta (Senior Member, IEEE) received the Ph.D. degree in engineering science from Louisiana State University, Baton Rouge, LA, USA, in 2003, specializing in lightweight advanced composite materials.

He is currently a Professor with the Department of Mechanical and Aerospace Engineering, New York University, Brooklyn, NY, USA, where he is also affiliated with the Center for Cybersecurity and the Department of Civil and Urban Engineering. He has four issued and six pending patents. He has authored or coauthored more than 195 journal articles and book chapters. His current research projects are focused on cybersecurity in additive manufacturing and additive manufacturing security education and use of machine learning methods in materials characterization. As a materials scientist, he has been interested in developing lightweight advanced composites of metals and polymers for dynamic loading conditions. His research has been supported by the National Science Foundation, the Office of Naval Research, the Army Research Laboratory, and industry.

Prof. Gupta has served as a Membership Secretary of the American Society for Composites and the Chair of the TMS Composite Materials Committee.

Ramesh Karri (Fellow, IEEE) received the B.E. degree in ECE from Andhra University, Visakhapatnam, India, in 1985, and the Ph.D. degree in computer science and engineering from the University of California at San Diego, San Diego, CA, USA. in 1993.

He is currently a Professor of electrical and computer engineering with New York University, Brooklyn, NY, USA. He also co-directs the NYU Center for Cybersecurity. He also leads the Cyber Security thrust of the NY State Center for Advanced Telecommunications Technologies, NYU. He co-founded Trust-Hub. His research and education interests include hardware cybersecurity including trustworthy ICs, processors and cyber-physical systems; security-aware computer-aided design, test, verification, validation, and reliability, nano meets security, hardware security competitions, benchmarks, and metrics, biochip security, and additive manufacturing security.