NOS^3 기반 위성 공급망 및 통신 보안 취약점 연구

황선혁[0], 장대희
경희대학교(학부생), 경희대학교
buckelwal8979@khu.ac.kr, daehee87@khu.ac.kr

Analysis of Satellite Supply Chain and Communication Security Vulnerabilities Using NOS^3

SunHyuk Hwang[0], Daehee Jang
KyungHee University(Undergraduate), KyungHee University
buckelwal8979@khu.ac.kr, daehee87@khu.ac.kr

요 약

우주 산업의 민간화와 상업화가 가속화되면서 인공위성을 활용한 다양한 서비스가 사회 전반에 중요한 역할을 하고 있다. 인공위성 개발은 보통 설계, 제조, 운영 과정이 분리된 분산 개발 구조를 기반으로는 이루어져 각 단계에서 악의적 개입이나 취약점 발생 등의 supply chain 공격의 가능성을 높인다. 또한 인공위성은 특성상 지상국과의 통신이 무선 신호를 통해 이루어지기 때문에 현재와 같이 Security by Obscurity에만 의존하여 암호화되지 않은 데이터를 송수신할 시, 도청, 신호 위변조, 비인가 신호 삽입과 같은 공격에 취약하다. 본 논문에서는 NASA의 위성 시뮬레이터 NOS^{3}의 가상 하드웨어 코드를 수정해 악의적인 코드를 삽입하고, 비인가 신호를 통해 해당 기능을 작동시켜 위성 시스템을 마비시키는 공격 시나리오를 구현하였다. 이를 통해 공급망 공격과 통신 보안 취약점이 위성 시스템에 미치는 위협을 검증하고, 보안 강화의 필요성을 강조하였다.

1. 서 론

민간이 우주 산업을 주도하는 뉴스페이스(NewSpace) 시대의 도래로 인해 스페이스X의 스타링크(Starlink), AWS의 Ground Station 서비스와 같이 인공위성을 활용한 인터넷 접속 서비스, 원격 탐사, 클라우드 서비스 등 다양한 상업적 및 과학적 활용이 급증하고 있다. [그림 1] 이러한 서비스들의 증가는 위성 보안 더 나아가 우주 보안의 중요성이 더욱 부각되고 있음을 보여준다.

인공위성의 설계, 부품 제조, 소프트웨어 개발, 발사 및 운영의 각 단계는 고도의 전문성을 필요로 한다. 따라서 하나의 기업이나 기관에서 독립적으로 진행하지 않고, 다양한 기업과 기관이 각각의 역할을 분담하여 개발에 참여하는 복잡한 형태로 이루어진다. 이러한 분산 개발 구조로 인해 위성 시스템의 사이버 보안 중, 공급망(supply chain) 보안은 필수요소이다.

또한, 인공위성 특성상 지상국과의 통신은 무선 신호를 통해 이루어지므로, 도청, 신호 위변조, 비인가 신호 삽입과 같은 공격에 취약하다. 특히, 많은 위성 통신 시스템이 은폐에 의한 보안(Security by Obscurity)에 의존하고 암호화되지 않은 통신을 사용하여 공격자가 시스템에 쉽게 침투할 수 있는 환경을 제공한다. 이러한 문제는 공급망 보안 취약점과 통신 보안의 허점이 결합될 경우, 인공위성의 안전성과 신뢰성에 심각한 영향을 미칠 수 있음을 시사한다.

본 논문에서는 이러한 문제를 실증적으로 검증하기 위해 NASA의 비행 소프트웨어 프레임워크인 cFS(Core Flight System) 기반 위성 시뮬레이터 NOS^{3}를 활용하여 위성 시스템 공급망 공격을 구현하였다. NOS^{3}의 가상 하드웨어 중 하나인 Generic Reaction Wheel의 cFS 코드를 수정하여 악의적인 기능을 삽입하였으며, 해당 과정에서 비인가 신호를 송신하여 위성 시스템을 손상시키는 시나리오를 구현하였다. 이를 통해 공급망 공격이 위성 시스템에 미칠 수 있는 보안 위협을 검증하고, 위성 공급망 보안 및 통신 보안 강화의 필요성을 강조하였다.

[그림 1] 인공위성 산업의 확대

2. 본 론

2.1. 인공위성 시스템과 하드웨어 구조

인공위성 시스템은 수행하는 미션에 따라 그 구성이 다르지만, 일반적으로 지상국과 인공위성의 조합으로 파악할 수 있다. 인공위성은 지상국으로부터 Telecommand(TC) Uplink를 수신하여 자세를 제어하거나 펌웨어를 업데이트하거나, 버스나 페이로드 명령을 수신하여 관련 미션을 수행한다. 반대로 인공위성은 궤도 정보, 통신 상태 및 위성 모니터링 정보를 담은 Housekeeping 데이터, 그리고 페이로드 데이터를 지상국으로 송신하고 이는 Telemetry(TM) Downlink 라고 부른다.

인공위성 하드웨어는 버스(Bus)와 페이로드(payload)로 나눌 수 있다. 위성 버스는 위성의 전반적인 제어와 데이터를 관리하며 기본적인 기능과 운영을 지원하는 시스템을 말한다. 버스의 구성 요소로는 전력 시스템(EPS), 자세 제어 시스템(ADCS), 안테나를 포함한 통신 시스템(COM), 열 제어 시스템(TCS), 추진 시스템(Propulsion System) 그리고 위성의 중앙 제어 시스템(C&DH) 등이 있다. 특히 C&DH 시스템 내에서 온보드 컴퓨터(OBC)는 명령 처리와 데이터 관리의 중앙 제어 역할을 담당하여 인공위성의 두뇌라고 볼 수 있다. 인공위성 페이로드는 해당 위성이 수행하는 특정 임무와 관련된 장비나 시스템을 말한다. 통신 위성의 경우 송수신 안테나와 트랜스폰더 등이 있을 수 있으며, 관측용 위성의 경우 카메라, 레이더, 적외선 센서 등이 있을 수 있다. 위성 버스에는 극저온 및 우주 방사선과 같은 극한의 우주 환경을 버티는 기능과 지상국과의 통신 기능이 포함되기 때문에 공격을 받아 그 기능을 상실하면 위성 운영 전반에 영향을 주어 임무 수행이 불가능해지거나 심각한 경우 다른 위성에도 영향을 미칠 수 있기 때문에 보안의 관점에서 페이로드보다 상대적으로 중요하다.

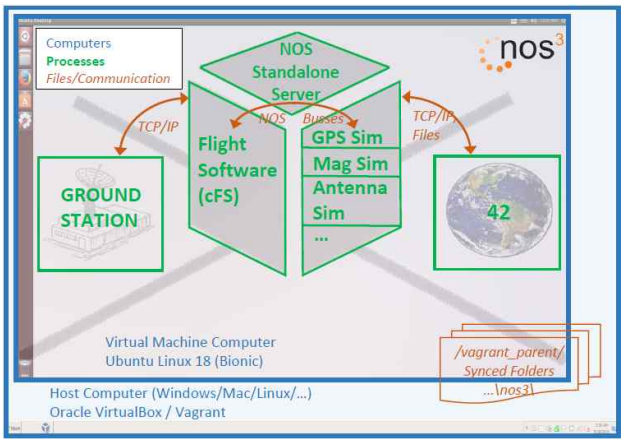
2.2. NASA Digital Twin과 NOS^3 시뮬레이터

미국 항공 우주국(NASA)은 우주 미션의 성공률을 높이기 위하여 class A로 분류되는 대형 미션 대부분에 Digital Twin 모델을 만들어 사전 테스트를 진행한다. Digital Twin은 실제 인공위성이나 우주 물리적 시스템을 정밀하게 모사한 가상 모델로, 인공위성의 하드웨어와 소프트웨어 조건, 지상국과의 통신 프로토콜, 궤도 역학과 같은 환경적 요인까지 포함하여 실제와 최대한 비슷한 방식으로 작동하도록 설계된다. NASA는 이를 이용하여 시스템 성능 검증, 오류 분석, 그리고 비상 상황 대응 전략 수립 등을 진행한다.

NOS^3(NASA Operational Simulator for Small Satellites)는 CubeSat과 같은 Small Satellite 규격의 위성을 위하여 NASA에서 개발한 오픈소스 위성 시뮬레이터로, 최초 제작 시 STF-1 큐브위성의 Digital Twin 모델로 제작되었으나 현재는 일반적인 소형 위성의 설계, 개발, 검증, 운용 준비까지 포괄적으로 지원하는 도구로 개발 및 사용되고 있다. STF-1 큐브위성 미션의 경우 일반적인 미션들과 다르게 NOS^3 시뮬레이터의 역량을 시연하는 것이었으며, [그림 2]에서 볼 수 있듯 2024년 2월 마지막 통신을 끝으로 현재는 운용이 중단되었다.

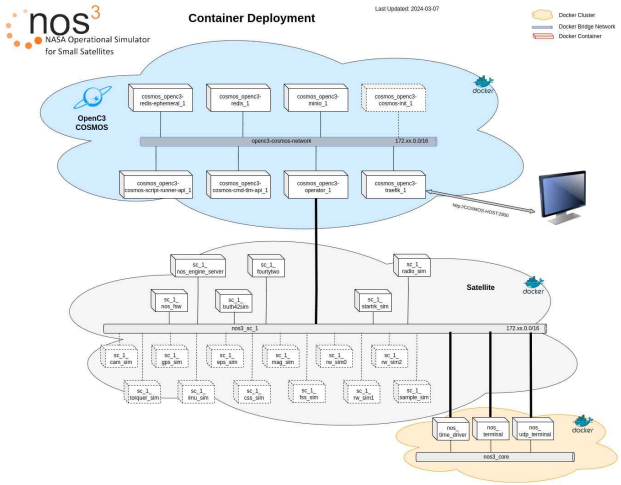
[그림 2] STF-1 미션 정보

NOS^3 시뮬레이터는 [그림 3]과 같이 지상국 역할을 하는 OpenC3 社의 COSMOS 소프트웨어, 인공위성의 자세와 궤도를 시뮬레이션하여 시각화하여 보여주는 42와 함께 작동한다. STF-1 큐브위성에 탑재된 하드웨어들을 기반으로 일반적인 위성 탑재 버스와 페이로드들을 가상화한 Generic Components와 비행 소프트웨어 cFS, 그리고 인공위성 하드웨어 간의 통신을 가상화한 NOS Engine으로 이루어져 있다. 지상국과 위성은 TCP/IP로 통신하며 위성 하드웨어는 UART, I2C와 같은 실제로 위성에서 사용되는 유선 통신 프로토콜을 NOS Engine으로 가상화하여 통신한다.



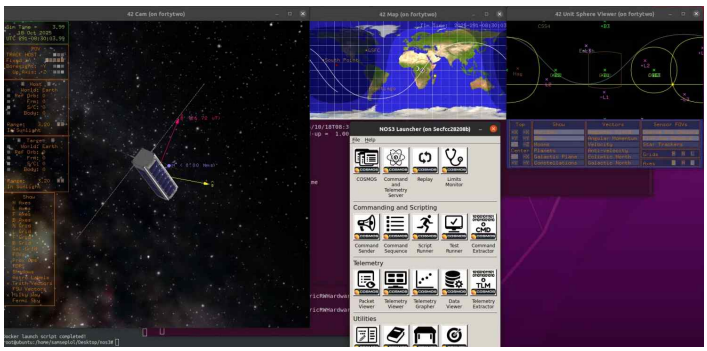
[그림 3] NOS^3 아키텍처

NOS^3 시뮬레이터 구축에는 Vagrant를 사용하는 방법과 Docker Image를 사용하는 방법이 있으며, Docker를 사용하는 방법으로 NOS^{3}를 구축하면 [그림 4]와 같이 위성 가상 하드웨어 컨테이너, cFS 컨테이너, COSMOS 지상국 컨테이너, 42 우주환경 시뮬레이터 컨테이너가 각각 생성되며 각각 IP 주소와 Port 번호를 할당받아 Docker network를 이용하여 통신한다.



[그림 4] 현재 Docker 아키텍처 (courtesy of contributor B.J. Kowalski)

본 논문에서는 Ubuntu 20.04 환경에서 Docker Image를 사용하여 구축하였다. NOS^{3}는 cmake 툴을 이용하여 시뮬레이터 전체를 빌드하고 실행하는데, `make all`과 `make launch` 명령을 사용하면 미리 설정한 가상 위성 하드웨어, 42, 그리고 COSMOS가 모두 빌드가 완료되고 실행되며 [그림 5]와 같은 화면(42: 화면 양옆 GUI와 화면 중앙의 세계지도 GUI, COSMOS: 화면 중앙 아래의 GUI)이 나타나는 것을 확인할 수 있다.



[그림 5] 실행된 NOS^3 시뮬레이터

2.3. NOS^3 시뮬레이터와 cFS

앞서 NOS^3 시뮬레이터에서 사용하는 비행 소프트웨어로 소개한 cFS(Core Flight System)는 위성을 포함한 항공우주 시스템에서 사용하기 위해 개발된 모듈화된 오픈소스 비행 소프트웨어 프레임워크이다. cFS는 소규모 프로젝트부터 제임스 웹 우주망원경(JWST)과 같은 대형 미션에 이르기까지 40개 이상의 NASA 임무를 지원해왔으며 현재도 계속해서 개발되고 있다.

cFS는 PSP(Platform Support Package), OSAL(Operating System Abstraction Layer), cFE(Core Flight Executive)의 단계로 구성되어 있다. PSP는 가장 하위 계층에서 하드웨어를 추상화해주는 단계이며, cFS startup, watchdog API 등의 라이브러리 기능들을 제공한다. PSP 위에 위치한 OSAL은 운영 체제와의 인터페이스를 추상화하여 운영 체제에 종속되지 않도록 설계되었다. OSAL은 RTEMS, VxWorks, Linux 등 다양한 운영 체제를 지원하며, 상위 계층인 cFE의 요청을 운영 체제와 연결하는 역할을 한다. 최상위 계층인 cFE는 비행 소프트웨어의 핵심 서비스를 담당하며, 명령 처리, 데이터 관리, 메시지 전달, 이벤트 로깅과 같은 위성 공통 기능을 제공한다. 이러한 계층적 구조를 통해 cFS는 코드 수정 없이도 다양한 하드웨어와 운영 체제 환경에 유연하게 적용될 수 있다.

cFS는 모듈화된 구조를 통해 새로운 하드웨어를 위성에 통합할 때 해당 하드웨어에 필요한 기능을 독립적인 컴포넌트(Component)의 형태로 추가할 수 있도록 설계되었다. 이러한 모듈화된 특성은 NOS^{3}에도 동일하게 적용되어, 새로운 cFS 컴포넌트를 추가하고 필요한 Telecommand 및 Telemetry 명령들을 COSMOS 지상국에 추가 정의함으로써 새로운 하드웨어를 쉽게 위성 시스템에 추가할 수 있다.

2.4. CCSDS 우주 통신 프로토콜

CCSDS(Consultative Committee for Space Data Systems)는 우주 임무에서 데이터 관리와 통신을 표준화하기 위해 설립된 국제 협력 기구로, 다양한 우주 프로토콜 표준을 개발하였다. 그 중 CCSDS 133.0-B-2 표준은 위성과 지상국 간의 효율적이고 신뢰할 수 있는 데이터 송수신하기 위하여 설계되었다. 해당 버전은 이전 버전인 133.0-B-1을 기반으로 2020년 개정판으로 발행되었다.

CCSDS 133.0-B-2 표준 프로토콜은 [그림 6]와 같은 형태의 Packet Primary Header를 사용하며 NOS^3 시뮬레이터도 CCSDS 표준 프로토콜을 따른다. 각각의 Component에 해당 가상 하드웨어가 사용할 CMD (Telecommand)와 TLM (Telemetry)의 포맷이 [그림 7]과 같이 정의되어 있으며 해당 CMD와 TLM은 빌드시에 COSMOS 컨테이너에 포함되어 사용자가 원하는 CMD를 cFS에 송신하거나 TLM을 수신받을 수 있도록 구성된다.

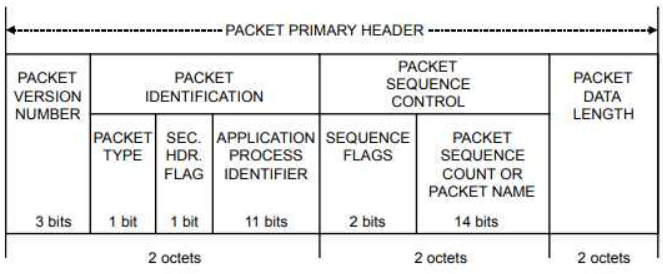


Figure 4-2: Packet Primary Header

[그림 6] CCSDS 133.0-B-2 Space Packet Primary Header

format

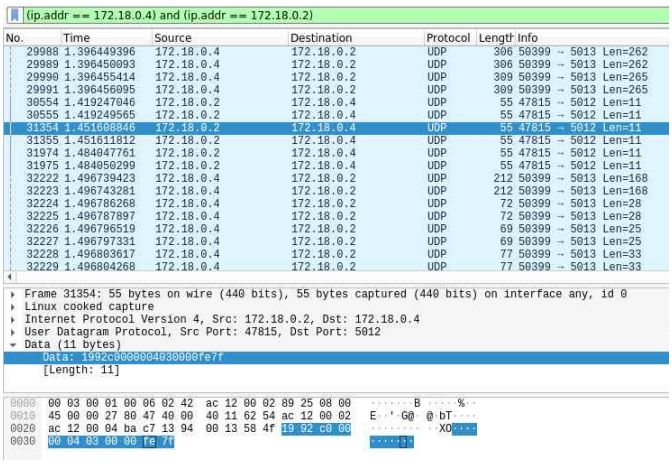
```
COMMAND GENERIC_REACTION_WHEEL GENERIC_RW_REQ_DATA_CC BIG_ENDIAN "Generic Reaction Wheel Request Data Command"
  APPEND_PARAMETER CCSDS_STREAMID      16 UINT MIN_UINT16 MAX_UINT16 0x1992 "CCSDS Packet Identification"
  APPEND_PARAMETER CCSDS_SEQUENCE      16 UINT MIN_UINT16 MAX_UINT16 0xC000 "CCSDS Packet Sequence Control"
  APPEND_PARAMETER CCSDS_LENGTH        16 UINT MIN_UINT16 MAX_UINT16 1      "CCSDS Packet Data Length"
  APPEND_PARAMETER CCSDS_FC             8  UINT MIN_UINT8 MAX_UINT8 2       "CCSDS Command Function Code"
  APPEND_PARAMETER CCSDS_CHECKSUM       8  UINT MIN_UINT8 MAX_UINT8 0       "CCSDS Command Checksum"
```

[그림 7] Generic Reaction Wheel CMD 중 "Request Data" 명령 정의

2.5. NOS^{3}의 명령 패킷 전송 흐름

본 논문에서 비인가 신호 송신 타겟으로 정한 인공위성 하드웨어 Component는 반작용휠(Reaction Wheel)이다. 반작용휠은 인공위성의 자세를 제어하기 위해 사용하는 장치이다. 반작용휠을 제어하는 RF 신호 modulation 특성과 구조를 알아내어 원하는 신호를 송신하면 인공위성을 필요 이상으로 회전시켜 안테나가 지상국과 정상적으로 송수신하지 못하게 하거나 내부 배터리를 빠르게 소진시키는 공격이 가능하다.

NOS^3 상에 구현된 지상국에서 반작용휠에 "Set Torque" 명령이 전송되는 과정은 다음과 같다. 우선 COSMOS의 Command Sender를 이용하여 GENERIC_REACTION_WHEEL의 GENERIC_RW_SET_TORQUE_CC 명령을 사용하여 Torque를 32766 (10^{-4}nm)로 설정하고 송신한다. COSMOS는 cosmos_openc3-operator_1 Docker 컨테이너에서 해당 명령을 UDP를 통하여 cFS가 작동 중인 sc_1_nos_fsw Docker 컨테이너로 [그림 8]과 같이 전송한다.

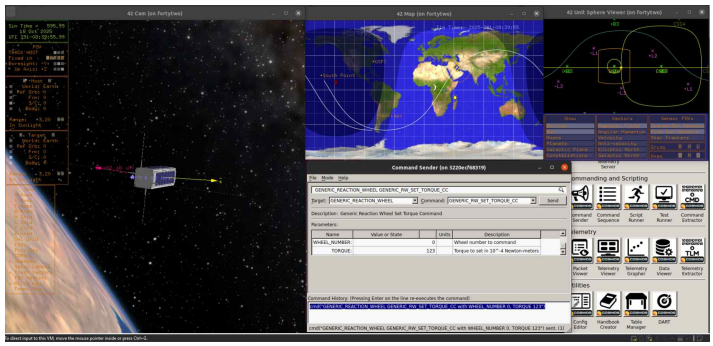


[그림 8] cosmos_openc3-operator_1에서 sc_1_nos_fsw로 명령이 전송되는 패킷 캡처

sc_1_nos_fsw는 명령 패킷을 NOS Engine에서 가상화한 UART를 통하여 sc_1_RW_sim0 Docker 컨테이너로 전송한다. sc_1_RW_sim0는 가상 반작용휠 중 첫 번째 하드웨어가 작동하는 컨테이너이며 여기서 Torque 값이 32766으로 설정되고 이를 42 화면을 통하여 인공위성이 빠르게 회전하는 것을 확인할 수 있다.

2.6. 공격 시나리오 구현

공격 시나리오 구현에 앞서 정상적인 위성의 작동과 비교하기 위하여 [그림 9]와 같이 NOS^3 시뮬레이터에서 정상적으로 COSMOS 지상국을 통하여 반작용휠에 "Set Torque" 신호를 송신하였다. 해당 과정은 https://youtu.be/26zJ99h-NhE 에서 확인할 수 있다.



[그림 9] 정상적인 RW 제어 신호를 송신하는 모습

본 논문에서 구현한 공격 시나리오는 비인가 신호가 위성 버스 하드웨어와 관련된 소프트웨어에 추가된 악성 코드를 트리거(trigger) 하는 것이 필요하다. 따라서 악성 코드는 비인가 신호 송신 타겟과 동일하게 반작용휠의 "Set Torque"기능에 추가하였다.

NOS^3 시뮬레이터의 Components는 개발의 편의를 위해 해당 Component가 필요로 하는 cFS 기능이 위성 전반에서 사용되는 cFE 코드와 분리되어 있다. 반작용휠의 cFS 기능의 경우 `generic_reaction_wheel_app.c`에서 정의되며 `GENERIC_RW_Set_Torque()` 함수에서 "Set Torque"를 처리한다. 해당 기능들은 NOS^3 빌드 시에 cFE와 합쳐져 하나의 Docker 컨테이너에서 동작하기 때문에 `GENERIC_RW_Set_Torque()` 함수는 cFS 프로세스에 접근할 수 있다. 따라서 [그림 10]과 같이 해당 함수 내부에서 torque 값을 가상 반작용휠 하드웨어에 전달하기 전 torque 값이 111.0으로 세팅되어 있는지 확인하고, cFS 프로세스의 이름인 "core-cpu1"이 작동하는지 검사하여 작동한다면 무한히 종료시키는 악성 코드를 삽입하고 빌드하였다.

```
/* Malicious func added! */
if ((torque - 111.0 > -0.000001) && (torque - 111.0 < 0.000001)) {
  system("echo 'Malicious activity started!!'");
  system("bash -c 'while true; do \
      pidof core-cpu1 > /dev/null && { \
          pkill -9 core-cpu1; \
          echo \"$(date): core-cpu1 killed\"; \
      }; \
      sleep 1; \
  done'");
}
```

[그림 10] 반작용휠 cFS 기능에 추가한 가상의 악성 코드

비인가 신호의 경우 실제 위성 통신과 NOS^3 시뮬레이터의 지상국 간 통신 방식이 상이하므로, 본 논문에서는 위성 통신 신호 분석과 위성 트래킹(tracking, 위성의 위치와 움직임을 실시간으로 추적하여 안테나를 정확히 조준하는 과정)이 완료되어 RF 단에서 데이터 패킷을 전송할 준비가 되었다는 가정을 바탕으로 시나리오 구현을 진행하였다.

실제 지상국에서 위성으로 신호를 송신하는 과정은 NOS^3 시뮬레이터 상에서는 cosmos_openc3-operator_1 컨테이너에서 sc_1_nos_fsw 컨테이너로 UDP 패킷을 전달하는 것과 동일한 과정이다. 따라서 [그림 6]과 같이 앞서 파악한 CCSDS 프로토콜 패킷 포맷에 기반하여 패킷을 생성하였다. 또한 NOS^3 시뮬레이터는 매 실행 시 Docker 컨테이너의 IP 주소와 Port 번호가 변경되기 때문에, NOS^3 실행 후 Wireshark와 [그림 11]과 같이 Docker inspect 명령어를 활용하여 변경된 IP 주소와 Port 번호를 확인하였다. 최종적으로 작성된 비인가 신호 송신 코드는 [그림 12]와 같다.

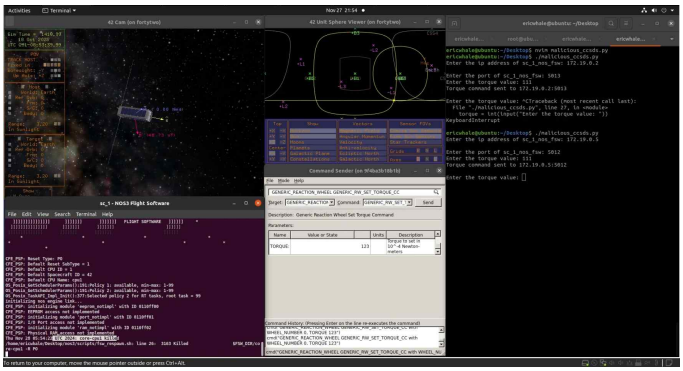

```
"Containers": {
    "0b1f4a17e09e079d9fce2d408f4a0d0a629dbd317ae642c6502d7715429f7bd8"
        "Name": "nos_udp_terminal",
        "EndpointID": "db3f308c21ea9d0e5f05b0c711cac7f86ebac43a0d404dd
        "MacAddress": "02:42:ac:12:00:09",
        "IPv4Address": "172.18.0.9/16",
        "IPv6Address": ""
    },
    "0ff61a8008b88124a152a4699b492ba379ec4c44a8555b53161c970f65b7dea1"
        "Name": "sc_1_css_sim",
        "EndpointID": "cd46aa33d67984016839dd3cb08c36b14750ea8a97624df
        "MacAddress": "02:42:ac:12:00:0b",
        "IPv4Address": "172.18.0.11/16",
        "IPv6Address": ""
    },
    "1fa2c1b20be4e1bd94b32313b259dac5d5f9f11952819a9c9f859d096d354ecf"
        "Name": "sc_1_fortytwo",
        "EndpointID": "0afe8d89304b9438ec318ca91e35bdce1d9d0c4da1ee80f
        "MacAddress": "02:42:ac:12:00:03",
        "IPv4Address": "172.18.0.3/16",
        "IPv6Address": ""
    },
    "273aa9882eeb85aa420af9323046fac99767c8e6bad528509b747cad1b3be92f"
```


[그림 11] docker inspect 명령어 출력의 일부

```
#!/usr/bin/env python3
import socket
import struct


def packet_generator(torque):
    ccsds_packet_id = hex(0x1992)[2:]
    ccsds_packet_sc = "c000"
    packet_len = "0004"
    ccsds_fc = "03"
    checksum = "00"
    wheel_number = "00"
    torque_formatted = struct.pack('<H', torque).hex()

    packet = (
        ccsds_packet_id +
        ccsds_packet_sc +
        packet_len +
        ccsds_fc +
        checksum +
        wheel_number +
        torque_formatted
    )
    return packet

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

ip_address = input("Enter the ip address of sc_1_nos_fsw: ")
print()
port = int(input("Enter the port of sc_1_nos_fsw: "))
while True:
    torque = int(input("Enter the torque value: "))
    packet = packet_generator(torque)
    packet = bytes.fromhex(packet)
    try:
        sock.sendto(packet, (ip_address, port))
        print(f"Torque command sent to {ip_address}:{port}\n")
    except Exception as e:
        print(f"Failed to send command: {e}\n")
    finally:
        sock.close()
```

[그림 12] 비인가 신호 송신 파이썬 스크립트

[그림 12]의 스크립트를 실행하고, NOS^3 시뮬레이터에서 필요한 IP 주소와 Port 번호를 입력하고 악성 코드를 트리거하기 위한 Torque 값인 111.0을 입력하면 [그림 12]와 같이 sc_1 - NOS3 Flight Software CLI 창에서 "core-cpu1 killed"라는 메시지가 출력되며 인공위성 시스템이 중지된다. 이 과정에서 COSMOS 지상국의 CMD 기능 또한 작동하지 않으며, 42 우주 환경 시뮬레이터도 더 이상 가상 위성으로부터 값을 받지 못하여 작동이 중지된다. 해당 시나리오는 https://youtu.be/nHiDZ2vwCBI 에서 확인할 수 있다.



[그림 12] 비인가 신호를 수신하고 NOS^{3}의 작동이 중지된 모습

3. 결 론

본 논문에서는 NOS^3 시뮬레이터를 활용하여 위성 시스템의 공급망(supply chain) 및 통신 보안 취약점을 실증적으로 분석하고, 이에 따른 위협을 검증하였다. 위성 시스템의 주요 구성 요소 중 하나인 반작용휠(Reaction Wheel)을 대상으로 "Set Torque" 기능에 cFS 프로세스를 중지시키는 악성 코드를 삽입하고, 실제 위성에서 사용하는 CCSDS 133.0-B-2 Space Packet Protocol로 제작한 비인가 신호를 송신하여 해당 악성 코드를 트리거하여 위성 시스템 전체를 중단시키는 공격 시나리오를 구현하였다. 이를 통해 위성 소프트웨어 공급망 보안의 취약점이 통신 보안의 취약성과 결합될 경우, 위성 운영 전체에 심각한 영향을 미칠 수 있음을 입증하였다.

본 논문의 결과는 위성 시스템 공급망 보안 및 통신 보안을 강화하기 위한 중요한 시사점을 제공한다. 우선, 위성 소프트웨어 공급망 전반에 걸친 신뢰성 검증 및 코드 무결성 보장을 위한 체계적인 검토가 필수적이다. 또한, CCSDS에서 개발한 SDLS(Space Data Link Security Protocol) 표준과 같은 보안 프로토콜의 사용을 의무화하여 도청 및 비인가 신호 삽입 공격에 효과적으로 대응해야 한다. 마지막으로, NOS^{3}와 같은 시뮬레이터를 활용하여 위성 시스템 개발 초기 단계에서 잠재적인 보안 취약점을 사전에 분석하는 과정이 필요하다.

또한, 이번 연구를 통해 공급망 공격과 CCSDS 프로토콜의 취약점을 식별하는 데 그치지 않고, 인공위성의 구조와 NASA가 개발한 비행 소프트웨어의 내부 작동 방식을 심층적으로 이해할 수 있었다. 또한, 코드 분석 과정에서 NASA Procedural Requirements (NPR) Software Engineering Requirements 7150.2D와 같은 안전하고 결함 없는 코드를 작성하기 위한 NASA의 개발 규칙을 발견하며, 우주 시스템의 독특한 개발 환경을 이해할 수 있었다. 인공위성은 일반적인 시스템과는 상당히 다른 특성을 지니고 있어 해킹 및 보안 관점에서 접근이 어려운 분야이기 때문에 지속적인 연구와 탐구가 요구될 것으로 보인다.

위성 시스템의 설계 및 운영 단계에서 공급망 및 통신 보안 강화를 위한 노력은 점차 복잡해지는 우주 산업 환경에서 필수적이며, 이를 위한 시뮬레이션 기반 검증 및 분석은 향후 위성 보안 기술 발전에 중요한 역할을 할 것으로 기대된다.

[참고문헌]

[1] Johannes Willbold, Moritz Schloegel, Manuel Vogele, Maximilian Gerhardt, Thorsten Holz, Ali Abbasi, Space Odyssey: An Experimental Software Security Analysis of Satellites, 2023, KIET Monthly Industrial Economic Review Vol.281 77-88, Feb, 2022.

[2] John Lucas, FSW 2024: Day 1- The state of NOS 3 NASA Operational Simulator for Small Satellites(https://www.youtube.com/watch?v=wRnh21jmYFQ&t=749s), Jul 29, 2024.

[3] Scott Zemerick, FSW 2023 - Day 2- NASA GSFC SmallSats - A NOS3 Case Study (https://www.youtube.com/watch?v=bGErcMs3oGA), May 8, 2024.

[4] James Pavur (Oxford University), Ivan Martinovic (Oxford University), SOK: Building a Launchpad for Impactful Satellite Cyber-Security Research, Oct 2020.

[5] http://www.stf1.com/NOS3Website/Nos3MainTab.php (NOS3 시뮬레이터 홈페이지)

[6] https://public.ccsds.org/Pubs/133x0b1s.pdf (CCSDS 133.0-B-1 공식 문서)

[7] https://public.ccsds.org/Pubs/355x0b2.pdf (CCSDS 355.0-B-2 공식 문서)

[8] https://etd.gsfc.nasa.gov/capabilities/capabilities-listing/cfs/?utm_source=chatgpt.com (NASA Goddard Engineering and Technology Directorate 홈페이지 cFS 소개문서)