
위성 SW를 위한 효율적인 Fuzzer 개발

202002473 김승혁

201902733 이정운

202002699 조민기



배경

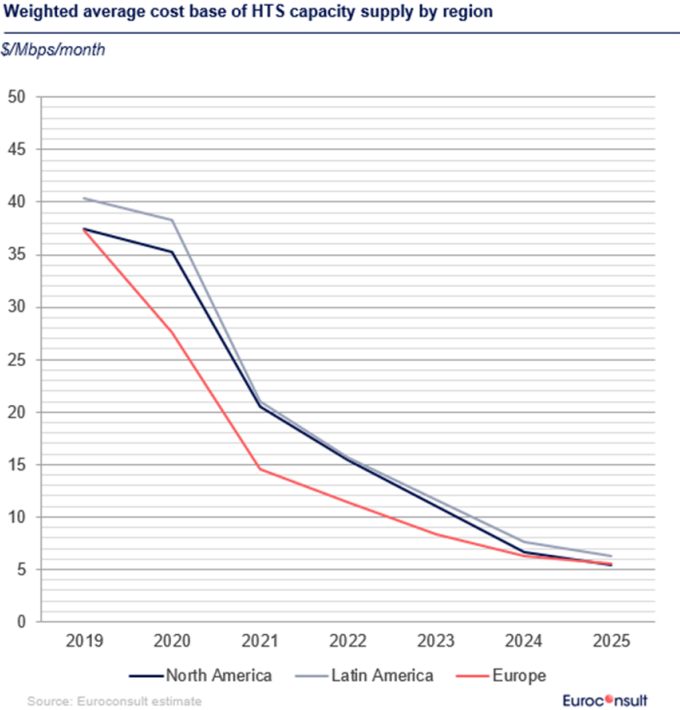


Figure 1. 무게당 발사 가격 추이



Global CubeSat Market

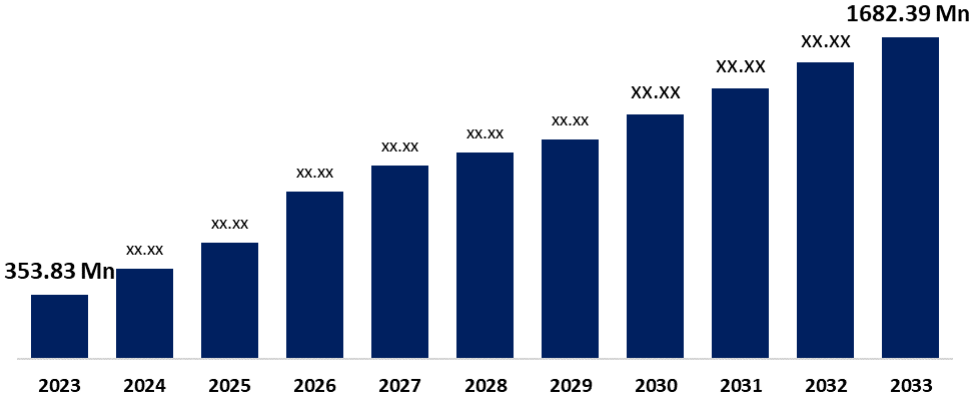


Figure 2. 소형 위성 시장 전망

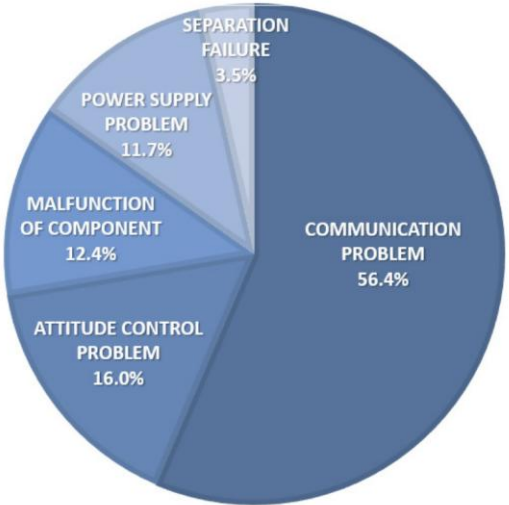


Figure 3. 실패 원인 분석

필요성

- 소형 위성 소프트웨어 검증 방법론의 필요성 대두 (특히 fprime)

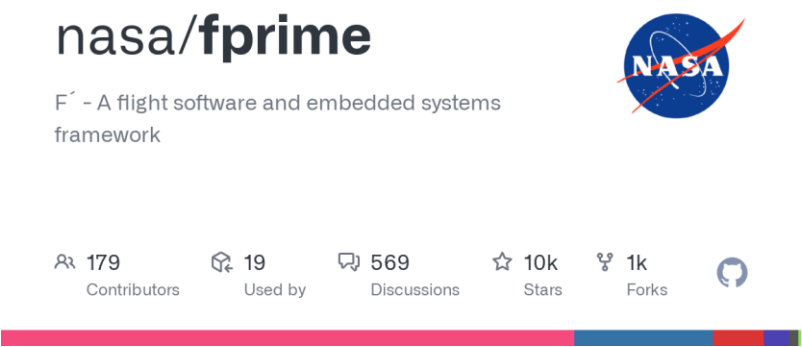


Figure 4. NASA의 F Prime

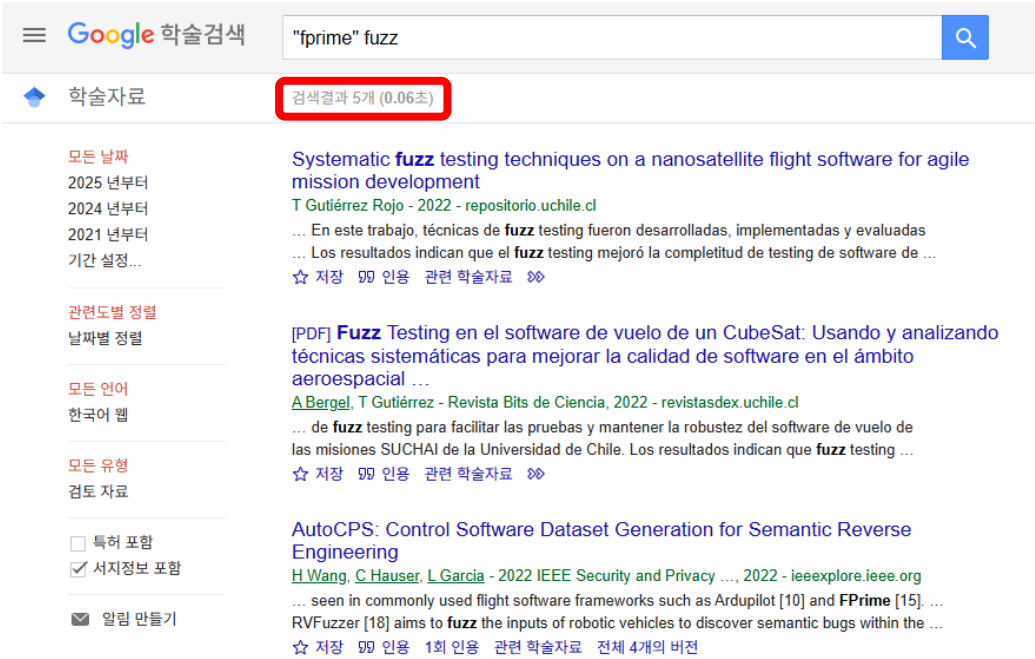


Figure 5. F Prime Fuzzing 연구

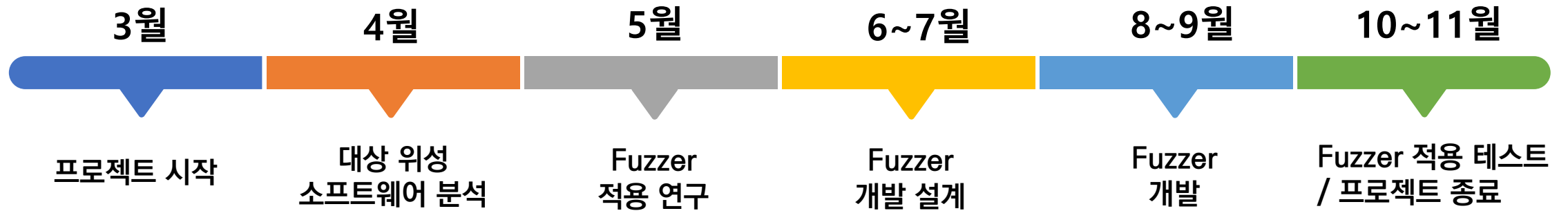
목표

- 따라서 Fuzzer를 통한 오류 검증 방식이 소형 위성에 적합한지 판단하고, 위성 검증에 적합한 Fuzzer 개발.
- Fprime의 내부 구조와 특징을 반영한 Fuzzer 개발.

-> 발사 전 지상 테스트 단계에서 결함을 미리 잡아내고, 실제 위성 운영 중 발생할 수 있는 문제를 사전에 차단하는 것이 목표



Fuzzer 개발 TIMELINE





김승혁



이정윤



조민기

이해 관계자 인터뷰

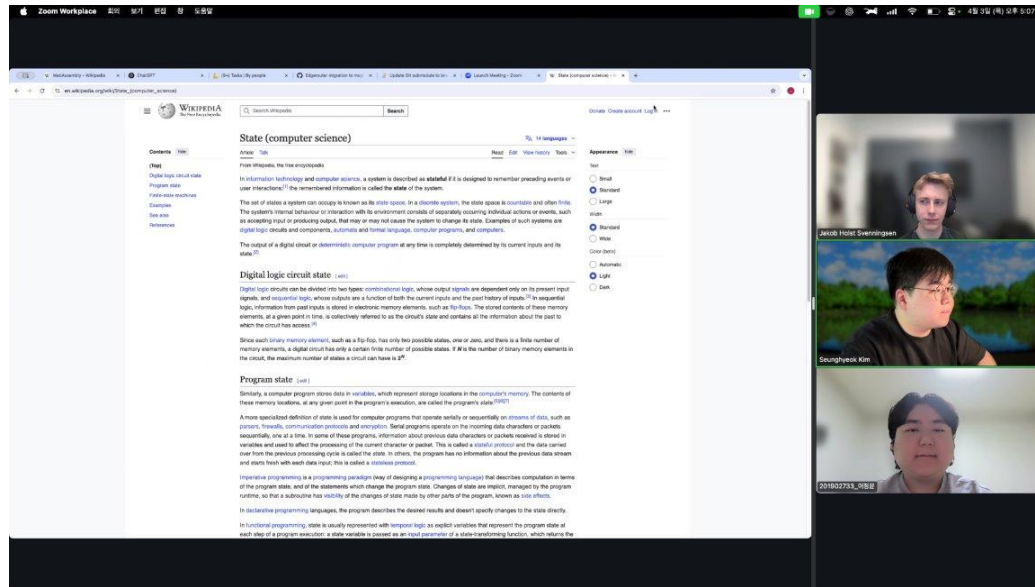


Figure 6. Jakob과의 인터뷰

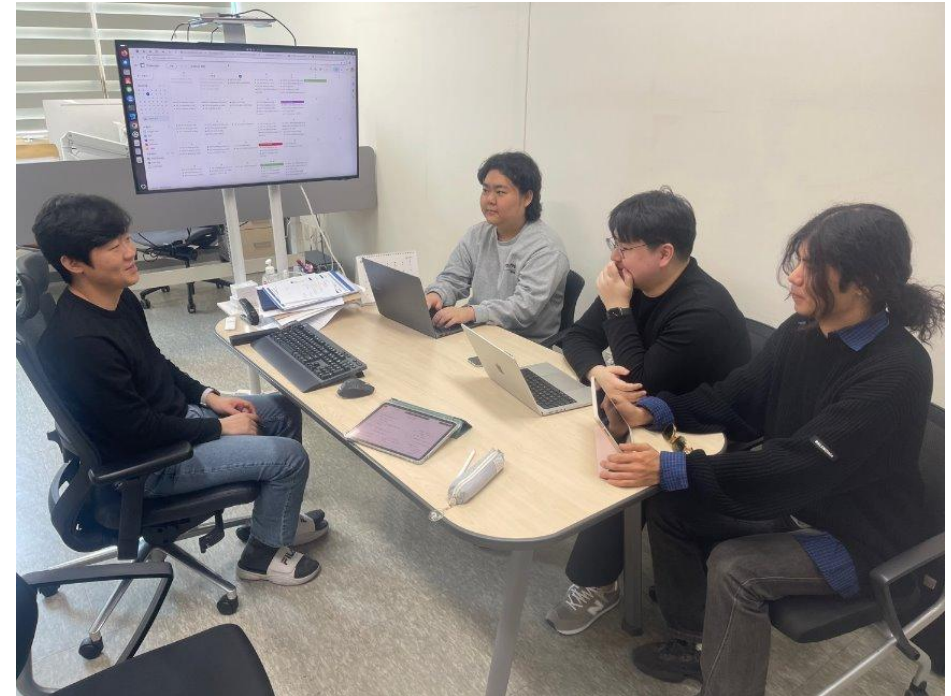


Figure 7. 이성호 교수님과의 인터뷰



이해 관계자 인터뷰

	이성호 교수님	김형신 교수님	Jakob
인터뷰 내용	<p>위성 SW 맞춤형 테스트 기법 부족. Fuzzer 성능 비교 후 결함 탐색 시도할 것.</p> <p>기존 Fuzzer는 위성 SW의 특성을 고려하지 못해 커버리지 확보 및 모듈 간 상호 작용 결함 탐지에 한계가 있을 수 있음.</p>	<p>위성 SW의 오류는 기능 상실 또는 임무 실패로 이어짐. 위성 시스템은 비정상적인 상태에서 탐지 및 복구가 어려움. 따라서 발사 전 지상에서의 철저한 테스트가 매우 중요함.</p> <p>대학의 소형 위성 같은 경우는 비용, 시간 문제 때문에 테스트가 비교적 적은 경향이 있음.</p>	<p>상태 기반 시스템 대상 Fuzzer 개발 경험 있음. fprime은 상태 기반 시스템이므로 Generation 기반 Fuzzer가 더 적합할 수 있음.</p> <p>Fuzzing 연구는 대상 프로토콜에 대한 깊은 이해가 중요함. (fprime의 fpp 등)</p>
인사이트	<p>대상 SW를 실제 구동해보고 단계적으로 Fuzzer를 적용해보는 것이 가장 중요함. 정량적 평가 지표 설정을 통해 분석해야함.</p>	<p>위성 SW는 특수성을 가지고 있으며, 테스트에서도 이러한 특성을 반영한 효과적인 테스트가 필요하다.</p>	<p>대상 SW의 특성을 잘 분석한 뒤, 이에 맞는 Fuzzing 전략을 찾아 적용하는 것이 중요하다.</p>



핵심 아이디어

제안 방법

: 여러 유형의 Fuzzer를 소형 위성 SW에 적용하여 분석. 이를 바탕으로 Fuzz Testing의 적합성 검증. 위성 SW에 적합한 Fuzzer 유형을 분석하고 제작

기존 방법 개선점

: 기존 테스트 방법에 더불어 새로운 동적 분석 방법을 제시하고 테스트를 자동화하여 검증 효율성을 높이하고자 함. 또한 테스트를 자동화하여 검증 효율성을 높이고 신뢰도를 높이하고자 함.



입력 형식 기반 Generation-Based Fuzzer



Figure 8. Google의 libProtobuf-mutator

입력 변이 기반 Mutation-Based Fuzzer

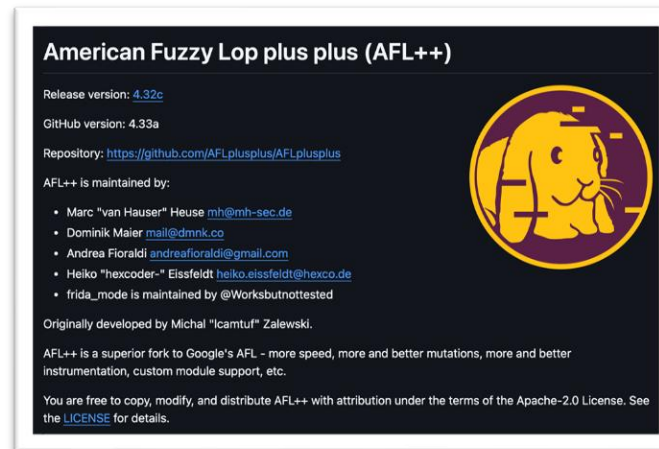


Figure 9. AFL++

실행 경로 기반 Coverage-guided Fuzzer

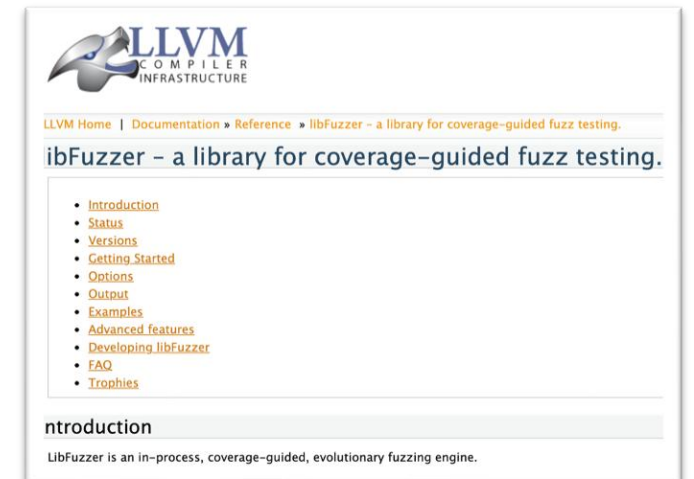


Figure 9. LLVM의 libFuzzer



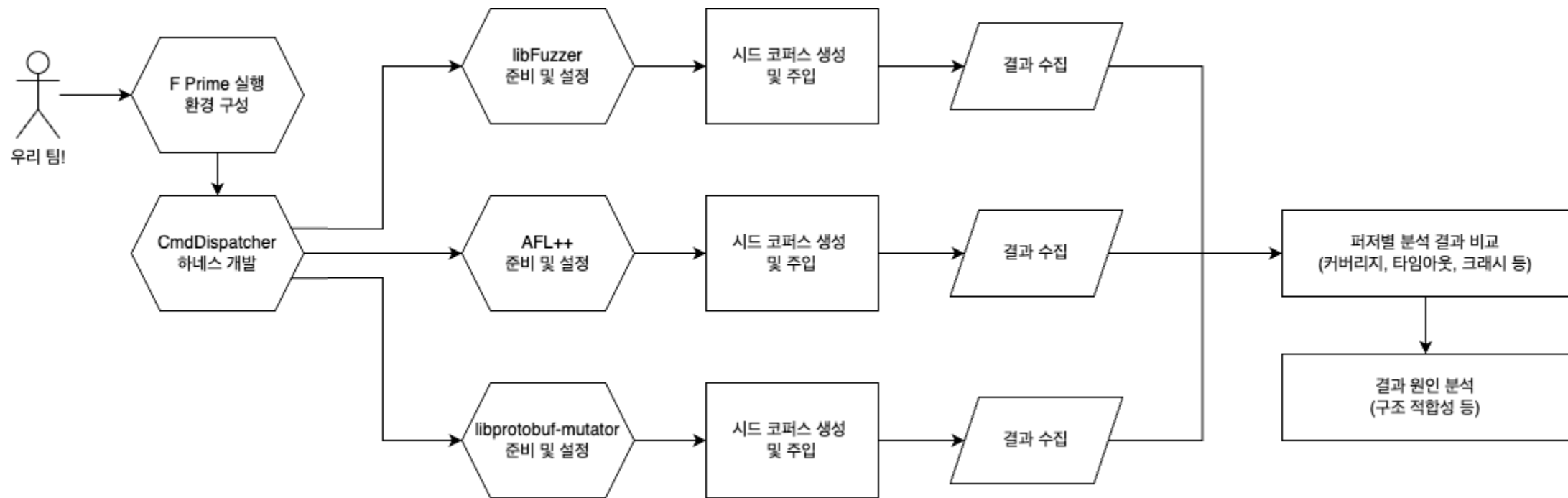


Figure 10. Fuzzer 적용 연구 순서도

```

docker logs -f fprime_aflpp

american fuzzy lop ++4.00c {default} (./cmd-fuzzer_afl) [fast]ts
process timing
  run time : 0 days, 8 hrs, 1 min, 19 sec
  last new find : 0 days, 7 hrs, 41 min, 0 sec
  last saved crash : 0 days, 7 hrs, 49 min, 57 sec
  last saved hang : 0 days, 8 hrs, 1 min, 17 sec
cycle progress
  now processing : 42*0 (95.5%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : arith 8/8
  stage execs : 1.31M/1.94M (67.46%)
  total execs : 5.91M
  exec speed : 200.9/sec
fuzzing strategy yields
  bit flips : 3/403k, 0/403k, 0/402k
  byte flips : 0/50.4k, 0/50.4k, 0/50.3k
  arithmetics : 0/964k, 0/10.9k, 0/3308
  known ints : 0/152k, 0/487k, 0/765k
  dictionary : 0/0, 0/0, 0/30.8k
  havoc/splice : 29/523k, 3/302k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 12.14%/2385, 98.13%
overall results
  cycles done : 26
  corpus count : 44
  saved crashes : 1
  saved hangs : 1
map coverage
  map density : 27.28% / 36.42%
  count coverage : 1.17 bits/tuple
findings in depth
  favored items : 21 (47.73%)
  new edges on : 22 (50.00%)
  total crashes : 692 (1 saved)
  total tmouts : 29.4k (14 saved)
item geometry
  levels : 3
  pending : 9
  pend fav : 0
  own finds : 34
  imported : 0
  stability : 100.00%
[cpu007: 62%]

```

```

docker logs -f fprime_libfuzzer

==63908==ERROR: AddressSanitizer: ABRT on unknown address 0x00000000f9a4 (pc 0xffff9742f1f0 bp 0xfffffcf04190 sp 0xfffffcf04190 T0)
#4290855: cov: 42 ft: 24 corp: 3 exec/s 0 oom/timeout/crash: 0/11252/4690 time: 28859s job: 15944 dft_time: 0
==63912==ERROR: AddressSanitizer: ABRT on unknown address 0x00000000f9a8 (pc 0xffff994cf1f0 bp 0xfffff018b590 sp 0xfffff018b590 T0)
#4291112: cov: 42 ft: 24 corp: 3 exec/s 128 oom/timeout/crash: 0/11252/4691 time: 28861s job: 15945 dft_time: 0
#4291243: cov: 42 ft: 24 corp: 3 exec/s 65 oom/timeout/crash: 0/11253/4691 time: 28863s job: 15946 dft_time: 0
#4291326: cov: 42 ft: 24 corp: 3 exec/s 0 oom/timeout/crash: 0/11254/4691 time: 28863s job: 15947 dft_time: 0
==63924==ERROR: AddressSanitizer: ABRT on unknown address 0x00000000f9b4 (pc 0xfffffbac1f0 bp 0xfffffe52a3a50 sp 0xfffffe52a3a50 T0)
#4292401: cov: 42 ft: 24 corp: 3 exec/s 537 oom/timeout/crash: 0/11254/4692 time: 28866s job: 15948 dft_time: 0
#4292574: cov: 42 ft: 24 corp: 3 exec/s 86 oom/timeout/crash: 0/11255/4692 time: 28868s job: 15949 dft_time: 0
#4293260: cov: 42 ft: 24 corp: 3 exec/s 343 oom/timeout/crash: 0/11256/4692 time: 28870s job: 15950 dft_time: 0
#4293282: cov: 42 ft: 24 corp: 3 exec/s 11 oom/timeout/crash: 0/11257/4692 time: 28872s job: 15951 dft_time: 0
#4293296: cov: 42 ft: 24 corp: 3 exec/s 7 oom/timeout/crash: 0/11258/4692 time: 28874s job: 15952 dft_time: 0
#4293319: cov: 42 ft: 24 corp: 3 exec/s 11 oom/timeout/crash: 0/11259/4692 time: 28877s job: 15953 dft_time: 0
#4293384: cov: 42 ft: 24 corp: 3 exec/s 32 oom/timeout/crash: 0/11260/4692 time: 28879s job: 15954 dft_time: 0
#4293540: cov: 42 ft: 24 corp: 3 exec/s 78 oom/timeout/crash: 0/11261/4692 time: 28881s job: 15955 dft_time: 0
#4293802: cov: 42 ft: 24 corp: 3 exec/s 131 oom/timeout/crash: 0/11262/4692 time: 28883s job: 15956 dft_time: 0
#4293918: cov: 42 ft: 24 corp: 3 exec/s 0 oom/timeout/crash: 0/11263/4692 time: 28883s job: 15957 dft_time: 0
==63964==ERROR: AddressSanitizer: ABRT on unknown address 0x00000000f9dc (pc 0xfffffbca2f1f0 bp 0xfffffcfe33630 sp 0xfffffcfe33630 T0)
#4295936: cov: 42 ft: 24 corp: 3 exec/s 559 oom/timeout/crash: 0/11263/4693 time: 28886s job: 15958 dft_time: 0
#4295460: cov: 42 ft: 24 corp: 3 exec/s 0 oom/timeout/crash: 0/11264/4693 time: 28886s job: 15959 dft_time: 0
==63972==ERROR: AddressSanitizer: ABRT on unknown address 0x00000000f9e4 (pc 0xfffff9e5ff1f0 bp 0xfffffd10e4c50 sp 0xfffffd10e4c50 T0)
#4295850: cov: 42 ft: 24 corp: 3 exec/s 0 oom/timeout/crash: 0/11264/4694 time: 28886s job: 15960 dft_time: 0
==63976==ERROR: AddressSanitizer: ABRT on unknown address 0x00000000f9e8 (pc 0xffff852cf1f0 bp 0xffffdabb5170 sp 0xffffdabb5170 T0)

```

```

docker logs -f fprime_protobuf_mutator

#537778: cov: 28 ft: 28 corp: 2 exec/s 49 oom/timeout/crash: 0/11734/390 time: 28855s job: 12125 dft_time: 0
#537815: cov: 28 ft: 28 corp: 2 exec/s 18 oom/timeout/crash: 0/11735/390 time: 28857s job: 12126 dft_time: 0
#537833: cov: 28 ft: 28 corp: 2 exec/s 9 oom/timeout/crash: 0/11736/390 time: 28860s job: 12127 dft_time: 0
#537860: cov: 28 ft: 28 corp: 2 exec/s 0 oom/timeout/crash: 0/11737/390 time: 28860s job: 12128 dft_time: 0
==48685==ERROR: AddressSanitizer: SEGV on unknown address 0xffffd741c29c (pc 0xffffb8f6d2e0 bp 0xffffc1b09d10 sp 0xffffc1b09d10 T0)
#537879: cov: 28 ft: 28 corp: 2 exec/s 9 oom/timeout/crash: 0/11737/391 time: 28862s job: 12129 dft_time: 0
#537893: cov: 28 ft: 28 corp: 2 exec/s 7 oom/timeout/crash: 0/11738/391 time: 28864s job: 12130 dft_time: 0
#537916: cov: 28 ft: 28 corp: 2 exec/s 11 oom/timeout/crash: 0/11739/391 time: 28866s job: 12131 dft_time: 0
#537925: cov: 28 ft: 28 corp: 2 exec/s 4 oom/timeout/crash: 0/11740/391 time: 28869s job: 12132 dft_time: 0
#537946: cov: 28 ft: 28 corp: 2 exec/s 10 oom/timeout/crash: 0/11741/391 time: 28871s job: 12133 dft_time: 0
#537963: cov: 28 ft: 28 corp: 2 exec/s 8 oom/timeout/crash: 0/11742/391 time: 28873s job: 12134 dft_time: 0
#538122: cov: 28 ft: 28 corp: 2 exec/s 79 oom/timeout/crash: 0/11743/391 time: 28875s job: 12135 dft_time: 0
#538194: cov: 28 ft: 28 corp: 2 exec/s 36 oom/timeout/crash: 0/11744/391 time: 28877s job: 12136 dft_time: 0
#538296: cov: 28 ft: 28 corp: 2 exec/s 51 oom/timeout/crash: 0/11745/391 time: 28880s job: 12137 dft_time: 0
#538332: cov: 28 ft: 28 corp: 2 exec/s 0 oom/timeout/crash: 0/11746/391 time: 28880s job: 12138 dft_time: 0
==48725==ERROR: AddressSanitizer: SEGV on unknown address 0xffffab21d99c (pc 0xffff8cd6d2e0 bp 0xfffff18527d0 sp 0xfffff18527d0 T0)
#538380: cov: 28 ft: 28 corp: 2 exec/s 24 oom/timeout/crash: 0/11746/392 time: 28882s job: 12139 dft_time: 0
#538388: cov: 28 ft: 28 corp: 2 exec/s 0 oom/timeout/crash: 0/11747/392 time: 28882s job: 12140 dft_time: 0
==48733==ERROR: AddressSanitizer: SEGV on unknown address 0xffffa901b91c (pc 0xffff8ab6d2e0 bp 0xffffce41b930 sp 0xffffce41b930 T0)
#538512: cov: 28 ft: 28 corp: 2 exec/s 62 oom/timeout/crash: 0/11747/393 time: 28884s job: 12141 dft_time: 0
#538524: cov: 28 ft: 28 corp: 2 exec/s 0 oom/timeout/crash: 0/11748/393 time: 28885s job: 12142 dft_time: 0
==48741==ERROR: AddressSanitizer: SEGV on unknown address 0xfffffa5c1b85c (pc 0xffff8776d2e0 bp 0xffffdfbb90d0 sp 0xffffdfbb90d0 T0)
#538560: cov: 28 ft: 28 corp: 2 exec/s 18 oom/timeout/crash: 0/11748/394 time: 28887s job: 12143 dft_time: 0

```

Figure 11,12,13. Fuzzer 실행 사진



timeout

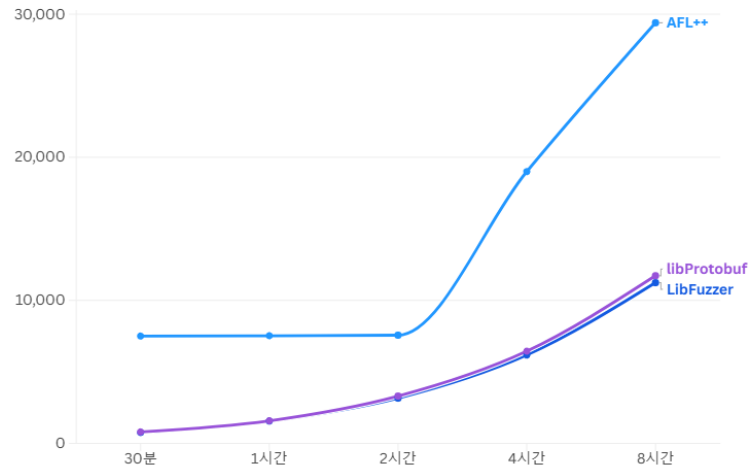


Figure 14. TimeOut 추이 그래프

edge count(coverage)

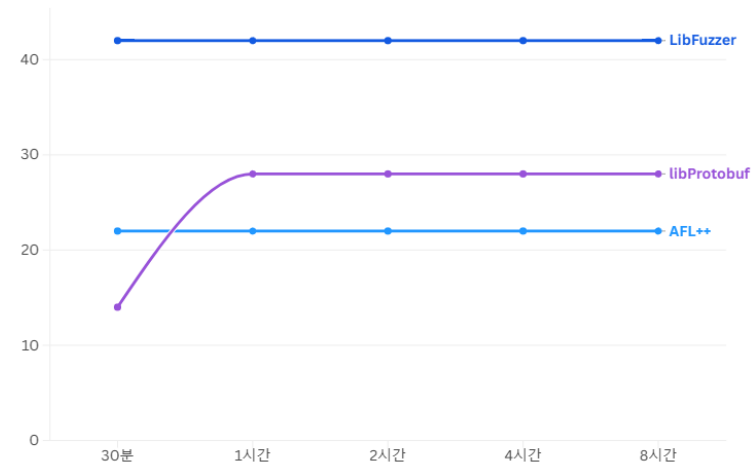


Figure 15. Code Coverage 추이 그래프



crashes

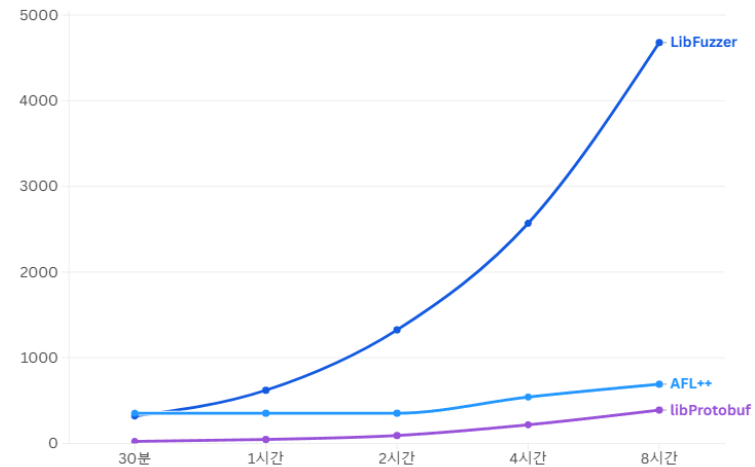


Figure 16. Crashes 추이 그래프

```
void CommandDispatcherImpl::compCmdReg_handler(NATIVE_INT_TYPE portNum, FwOpcodeType opCode) {
    // search for an empty slot
    bool slotFound = false;
    for (U32 slot = 0; slot < FW_NUM_ARRAY_ELEMENTS(this->m_entryTable); slot++) {
        if ((not this->m_entryTable[slot].used) and (not slotFound)) {
            this->m_entryTable[slot].opcode = opCode;
            this->m_entryTable[slot].port = portNum;
            this->m_entryTable[slot].used = true;
            this->log_DIAGNOSTIC_OpCodeRegistered(opCode, portNum, static_cast<I32>(slot));
            slotFound = true;
        } else if ((this->m_entryTable[slot].used) &&
            (this->m_entryTable[slot].opcode == opCode) &&
            (this->m_entryTable[slot].port == portNum) &&
            (not slotFound)) {
            slotFound = true;
            this->log_DIAGNOSTIC_OpCodeReregistered(opCode, portNum);
        } else if (this->m_entryTable[slot].used) { // make sure no duplicates
            FW_ASSERT(this->m_entryTable[slot].opcode != opCode, static_cast<FwAssertArgType>(opCode));
        }
    }
    FW_ASSERT(slotFound, static_cast<FwAssertArgType>(opCode));
}
```

Figure 17. CmdDispatcherImpl.cpp 소스 코드 일부

```

QueueInterface::Status Queue::receive(Fw::SerializeBufferBase& destination,
                                     QueueInterface::BlockingType blockType,
                                     PlatformIntType& priority) {
    FwSizeType actualSize = 0;
    destination.resetSer(); // Reset the buffer
    QueueInterface::Status status =
        this->receive(destination.getBuffAddrSer(), destination.getBuffCapacity(), blockType, actualSize, priority);
    if (status == QueueInterface::Status::OP_OK) {
        Fw::SerializeStatus serializeStatus =
            destination.setBuffLen(static_cast<Fw::Serializable::SizeType>(actualSize));
        if (serializeStatus != Fw::SerializeStatus::FW_SERIALIZE_OK) {
            status = QueueInterface::Status::SIZE_MISMATCH;
        }
    }
    return status;
}

```

```

Fw::QueuedComponentBase::MsgDispatchStatus CommandDispatcherComponentBase::
doDispatch()
{
    ComponentIpcSerializableBuffer msg;
    FwQueuePriorityType priority = 0;

    Os::Queue::Status msgStatus = this->m_queue.receive(
        msg,
        Os::Queue::BLOCKING,
        priority
    );
    FW_ASSERT(
        msgStatus == Os::Queue::OP_OK,
        static_cast<FwAssertArgType>(msgStatus)
    );

    // Reset to beginning of buffer
    msg.resetDeser();
}

```

Figure 18, 19. OS::Queue 소스 코드 일부

- Fprime의 cmddispatcher 모듈은 내부 상태 정보를 유지하는 준-상태 구조를 갖고 있음.
- 그러나 연구에서 사용한 상용 Fuzzer들은 이러한 상태를 고려하지 않고 입력만 무작위로 바꾸는 방식을 사용해서, 새로운 경로를 탐색하지 못하는 한계가 존재

-> 상용 Fuzzer의 한계점을 보완한 위성 소프트웨어 전용 Fuzzer 개발이 목표

- 해당 연구를 바탕으로 fprime에 적합한 fuzzing 기법 고안
- 테스트 범위 확장하여 fprime 이외의 다른 오픈 소스 위성 소프트웨어에도 적용 가능한지 검토
- 소형 위성의 신뢰성을 높이고 성공률 높이는데 기여하는 것이 목표

Thank You