# MP FUSE: 리눅스 FUSE 기반 메타모픽/폴리모픽 변이 기술

최재민*, 박준영**, 허남정*, 하영빈*, 장우현***, 김연재***, 허재원***, 박기웅†

*,**세종대학교 SysCore Lab. (대학원생*, 학부연구생**)

***LIG넥스원 (연구원***)

†세종대학교 정보보호학과 (교수†)

# MP FUSE: Metamorphic and Polymorphic Mutation Technique Based on Linux FUSE

Jae-Min Choi*,Jun-Young Park**, Nam-Jung Heo*, Young-Bin Ha*, Woo-Hyun Jang***, Yeon-Jae Kim***, Jae-Won Huh***, Ki-Woong Park†

*,**SysCore Lab., Sejong University (Graduate Student*, Undergraduate Student**)

***LIG Nex1 (Researcher***)

†Dept. of Computer and Information Security, Sejong University (Professor†)

**요 약**

소프트웨어 산업과 Internet-of-Things(IoT)의 급속한 발전으로 내부 코드와 민감 정보의 보호 중요성이 점점 더 커지고 있다. 기존의 소프트웨어 보호 기술인 난독화, 암호화, 패킹 등은 주로 정적 분석에 대한 방어 기능을 수행하지만, 실행 시점에서 발생하는 메모리 덤프나 동적 분석과 같은 공격에는 취약하다는 한계가 있다. 이러한 문제를 해결하고자 본 논문에서는 리눅스 Filesystem in Userspace(FUSE) 기반의 Metamorphic/Polymorphic FUSE(MP FUSE)를 제안한다. 제안하는 MP FUSE는 소프트웨어가 실행될 때마다 내부 코드에 대해 동적으로 변이시켜, 동일한 기능을 유지하면서 코드 구조를 지속적으로 변경함으로써 공격자의 분석에 어려움을 준다. MP FUSE를 통하여 다양한 형식의 실행, 내부 파일에 대하여 정상 실행을 보장한 채 폴리모픽, 메타모픽 변이를 성공적으로 적용함을 확인하였다. 향후 연구에서는 변이 알고리즘의 고도화 및 컴파일된 내부 변수에 대한 암호학적 변이를 통해 소프트웨어 보호 기술의 고도화를 목표로 한다.

## I. 서론

소프트웨어 산업과 IoT의 발전 및 고도화됨에 따라 내부 코드 및 내부 정보의 중요성이 점점 더 커지고 있다[1]. 이러한 내부 코드 및 정보를 보호하기 위해 여러 소프트웨어 보호 기술이 연구 및 개발되고 있다. 대표적으로 난독화, 암호화, 패킹 등이 있다.

해당 기술들은 대부분 정적 분석에 대한 보호 기능을 수행하고 있고, 실행 시점에서의 공격인 메모리 덤프, 동적 분석 등에 취약하다는 한계를 지니고 있다. 특히, 리버스 엔지니어링과 디버깅 도구의 고도화로 인해 난독화에 대한 복호화가 쉽고, 복호화된 실행 메모리 영역을 추출함으로써 난독화, 복호화를 통한 보호 효과가 무력화되고 있다[2].

이를 해결하기 위해 다형성 코드 기반 보호 기법 연구가 제안되고 있다. 다형성 코드 기반

† 교신저자: 박기웅 (세종대학교 정보보호학과 교수)

이 논문은 2022년 정부(방위사업청)의 재원으로 국방기술진흥연구소의 지원을 받아 수행된 연구임 (KRIT-CT-22-051)

보호 기법 연구는 동일 기능을 수행하는 코드에 대하여 형태를 변이시켜 동적 메모리 분석과 같은 패턴 기반의 분석에 어려움을 주고, 정적 분석에 대해서도 대응할 수 있다.

본 논문에서는 이러한 다형성 기반 코드 보호 기법을 리눅스 FUSE를 기반으로 구현하여 내부 정보, 내부 소프트웨어에 대하여 실행 시마다 동적으로 변이시키는 방식으로 보호 기법을 구현하고 해당 프레임워크 Metamorphic/Polymorphic FUSE(MP FUSE)에 대하여 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 다형성 기반 코드 보호 기법 연구에 대해 서술하고, 3장에서는 제안하는 프레임워크에 대해 서술한다. 4장에서는 프레임워크를 통해 구현된 코드 변이 기술에 대해 서술하고, 5장에서는 결론 및 향후 연구를 서술한다.

## II. 관련 연구

### 2.1. 다형성 코드 기반 보호 기법 연구

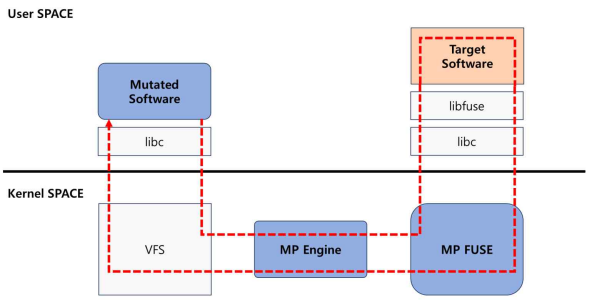
다형성 코드 기반 보호 기법은 다형성 기반의 악성코드 탐지 연구에서 간접적으로 확인할 수 있다.

Nazim et al.은 실행 시마다 형태를 달리하는 다형성 특성은 기존 탐지 기법으로 식별하기 어렵다는 점을 강조하며 멀티모달 딥러닝(Multimodal Deep Learning) 모델을 제안하였다[3].

Nawaz et al. 또한 이러한 한계를 보완하기 위해 순차 패턴 마이닝(Sequential Pattern Mining)을 기존 탐지 기법과 결합하였다[4].

이렇듯 다형성 코드가 악성코드 탐지를 어렵게 하는 것과 같이 소프트웨어의 역공학 또한 어렵게 만들 수 있음을 보여주고, 탬퍼링 상황에서 효과적인 수단으로 활용될 수 있음을 시사한다.

Morel et al.은 다형성 코드를 기반으로 소프트웨어를 보호하는 직접적인 사례를 보여준 연구로, 해당 연구에서 제안한 PolEn은 다형성과 암호화를 결합하여 실행 중에 코드 구조를 달리해 공격자가 사이드 채널(Side Channel) 정보를 분석하기 어렵게 만든다[5].




(그림 1) Metamorphic/Polymorphic Engine with Linux FUSE

Caliandro et al.은 VMorph 프레임워크를 통해 난독화와 다형성을 결합한 접근을 제안하며 실행 시마다 코드 구조를 변화시키는 방식으로 정적 분석을 회피하고 지적 재산을 보호할 수 있음을 보여준다[6].

이렇듯 기존 연구들은 다형성 기법이 악성코드 탐지를 어렵게 만든다는 점을 이용하여 소프트웨어의 역공학 및 탬퍼링을 방지하는 방법으로 사용할 수 있음을 보여준다.

## III. 바이너리 동적 변이를 위한 MP FUSE 프레임워크

우리는 리눅스 FUSE에서 변이 엔진을 구현하기 위해 (그림 1)과 같이 MP FUSE의 전체 구조를 설계하였다.

### 3.1. Polymorphic 엔진

사용자가 Userspace에서 read() 함수를 호출하면 FUSE에서 정의된 read_mp() 함수를 호출하고, 해당 함수에서는 MP Engine을 통해 호출한 데이터 원본 '/data/origin' 파일에 대하여 변이를 발생시킨다. 변이 로직은 다음과 같다.

보호 대상 소프트웨어의 확장자, 문자열에 대하여 기정의된 규칙을 기반으로 해당 확장자, 문자열에 대해서 변이를 발생시킨다. 이때, 변이에 대해서는 특정 문자열에 대한 대치, 암호화를 지원하며 이는 모듈 형태로 제공하여 사용자에 따라 원하는 암호화 방식, 대치 방식을 선언하여 사용할 수 있다.

이때, 파일의 실행과 직결되는 헤더 구조, 로직과 같은 부분에 대해서 손상을 방지하기 위해 특정 바이트 사이즈에 대해서는 변이를 발생시키지 않음으로써, 보호 대상 소프트웨어

의 실행을 보장한다.

최종적으로, 해당 엔진에서 발생시키는 변이에 대해서는 임의의 랜덤 문자열 기반의 키를 사용함으로써 소프트웨어의 최종 실행 결과에 대해서는 영향을 끼치지 않으며, 내부 문자열과 변수에 대한 변이를 일으킴으로써 공격자의 정적, 동적 분석에 어려움을 줄 수 있다.

### 3.2. Metamorphic 엔진

Metamorphic 변이 또한 Polymorphic과 동일하게 Userspace에서 호출된 사용자 read() 함수에 대하여 read_mp() 함수를 통하여 변이를 발생시킨다. Metamorphic 엔진의 변이 로직은 다음과 같다.

실행할 수 있는 코드 영역의 기계어 명령어 자체를 변형 대상으로 하여 같은 실행 결과를 출력함에도 내부 로직이 변환되도록 변이를 발생시킨다. read_mp() 함수를 통해 전달받은 바이너리 파일에서, 기정의된 규칙에 따라 특정 명령어 또는 명령어 조합을 탐지하고, 탐지된 명령어는 기능적으로 동일하지만 표현 방식이 완전히 다른 명령어 또는 명령어 조합으로 대체되고, 이때 다음과 같은 상황에 맞춰 엔진이 작동하도록 구현하였다.

(수식 1) $length(asm_{origin}) > length(asm_{mutated})$

(수식 2) $length(asm_{origin}) = length(asm_{mutated})$

(수식 3) $length(asm_{origin}) < length(asm_{mutated})$

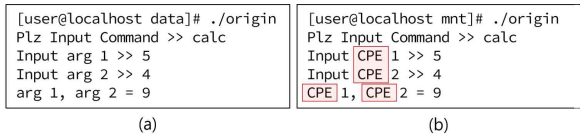
(수식 1)의 경우 NOP(0x90)을 활용하여 빈 공간에 대하여 '$asm_{orgin}$'과 크기를 맞춰주면 되고, (수식 2)의 경우 동일하게 변이 코드를 삽입하면 된다. 마지막으로 (수식 3)의 경우 '$as_{muated}$'를 Code Cave를 활용하여 작성한 뒤 JMP 명령어를 이용해 복귀하여 해결할 수 있다.

최종적으로, 해당 엔진에서 발생시키는 변이에 대해서는 기존의 실행 결과를 유지하며, 내부 논리적 실행 구조를 변형시킴으로써 공격자가 내부 정보 소프트웨어 대상 메모리 덤프를 통한 동적 분석에 대하여 어려움을 줄 수 있다.

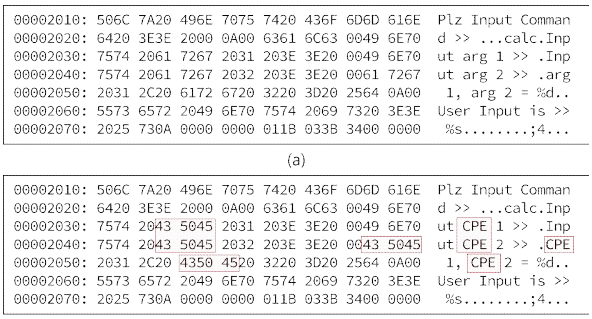
## IV. 실험 결과

### 4.1 Polymorphic 변이 결과

본 논문에서는 Polymorphic 변이 대상 소프트웨어 3종 elf, py, txt에 대하여 규칙을 작성




(그림 2) /data/origin and /mnt/origin result




(그림 3) /data/origin and /mnt/origin hex value

하였고, 변이를 진행하였다. 해당 파일들은 예시 계산기 예제로, 'arg 1', 'arg 2'를 입력받아 합을 출력하는 파일이다.

기존 '/data/origin' 파일의 실행 결과는 (그림 2-a)와 같다. 해당 파일을 FUSE를 통해 마운트된 폴더의 '/mnt/origin'에서 실행하면 (그림 2-b)와 같이 변형된 출력 결과를 출력한다. 이는 3장에서 언급한 기존 실행을 보장하며, 보호 대상 소프트웨어의 특정 문자열에 대해 XOR 연산을 수행한 결과이다.

또한, (그림 3)과 같이 컴파일된 실행 파일에 대해서도 올바르게 변이가 발생함을 확인하였다. 이때, 'rdata' 내의 strings 영역에 해당하는 값에 대해 변이가 가능하며 내부 변수명과 같은 컴파일되며 변경된 값에 대해서는 불가능하다.

### 4.2 Metamorphic 변이 결과

Metamorphic 변이 대상으로 컴파일된 실행 파일에 대해서 진행하였다. 실행 로직 변경을 위해 어셈블러 수준에서의 로직 변경을 진행하였다.

컴파일된 /data/origin 파일에 대하여 'gdb'를 통해 디스어셈블하여 내부 로직을 확인하면 (그림 4)와 같다. 해당 어셈블리어에서 파일의 실행 결과에 영향을 주지 않고, 로직을 변경하고자 3장의 수식 1,2,3 에 맞추어 'mov', 'jmp', 'add' 등에 대하여 변경을 진행하였고 변이 결과는 (그림 5)와 같다.

최종적으로, 실행 결과에 영향을 주지 않고,

```
Dump of assembler code for function main:
   0x00401141 <+0> :    push   %rbp
   0x00401142 <+1> :    mov    %rsp,%rbp
   0x00401145 <+4> :    sub    $0x10,%rsp
   0x00401149 <+8> :    movl   $0xa,-0x4(%rbp)
   0x00401150 <+15>:    movl   $0x14,-0x8(%rbp)
   0x00401157 <+22>:    mov    -0x8(%rbp),%edx
   0x0040115a <+25>:    mov    -0x4(%rbp),%eax
   0x0040115d <+28>:    mov    %edx,%esi
   0x0040115f <+30>:    mov    %eax,%edi
            . . .
   0x00401182 <+65>:    leave
   0x00401183 <+66>:    ret
End of assembler dump.
```

(그림 4) /data/origin with gdb

```
Dump of assembler code for function main:
   0x00401141 <+0> :    push   %rbp
   0x00401142 <+1> :    mov    %rsp,%rbp
   0x00401145 <+4> :    sub    $0x10,%rsp
   0x00401149 <+8> :    movl   $0xa,-0x4(%rbp)
   0x00401150 <+15>:    movl   $0x14,-0x8(%rbp)
   0x00401157 <+22>:    mov    -0x8(%rbp),%edx
   0x0040115a <+25>:    mov    -0x4(%rbp),%eax
   0x0040115d <+28>:    push   %rdx
   0x0040115e <+29>:    pop    %rsi
   0x0040115f <+30>:    mov    %eax,%edi
            . . .
   0x00401182 <+65>:    leave
   0x00401183 <+66>:    ret
End of assembler dump.
```

(그림 5) /mnt/origin with gdb

실행 로직을 변경함을 확인하였다. 하지만, 두 엔진 모두 기존의 패커와 같은 압축 바이너리에 대한 변이를 제공하지 못한다. 따라서, 변이 엔진에서의 변이 로직을 기존 기술을 모듈 형태로 활용하거나, 로직의 고도화가 필요하다.

## V. 결론 및 향후 연구

본 논문에서는 리눅스 FUSE를 통한 코드 변이 기술을 제안한다. 해당 기술을 통해 여러 확장자의 실행 파일, 문자열에 대하여 Userspace에서 호출이 발생할 때 변이를 발생시켜 보다 메모리 덤프와 같은 동적 분석 환경에 대응할 수 있음을 보여주었고, 모듈 방식으로 FUSE를 사용함으로써 사용자가 목적에 따라 암호화, 난독화, 변이 기술을 구현할 수 있음을 제시하였다. 향후 연구에서는 해당 FUSE에 대한 고도화를 진행하여 알고리즘 개선 및 컴파일된 내부 변수에 대한 암호학적 변이를 일으켜보고자 한다.

## [참고문헌]

[1] Malhotra, P, Singh, Y, Anand, P, Bangotra, D.K, Singh, P.K, Hong, W.-C. “Internet of Things: Evolution, Concerns and Security Challenges”. Sensors 2021, 21, 1809.

[2] Thorsten Jenke, Elmar Padilla, and Lilli Bruckschen. 2023. Towards Generic Malware Unpacking: A Comprehensive Study on the Unpacking Behavior of Malicious Run-Time Packers. In Secure IT Systems: 28th Nordic Conference, NordSec 2023, Oslo, Norway, November 16–17, 2023, Proceedings. Springer-Verlag, Berlin, Heidelberg, 245–262. https://doi.org/10.1007/978-3-031-47748-5_14

[3] Nazim, S., Alam, M.M., Rizvi, S. et al. Multimodal malware classification using proposed ensemble deep neural network framework. Sci Rep 15, 18006 (2025). https://doi.org/10.1038/s41598-025-96203-3

[4] Nawaz, M. S., Fournier-Viger, P., Nawaz, M. Z., Chen, G., & Wu, Y. (2022). MalSPM: Metamorphic malware behavior analysis and classification using sequential pattern mining. Computers & Security, 118, 102741. https://doi.org/10.1016/j.cose.2022.102741

[5] Morel, L., Couroussé, D., & Hiscock, T. (2022). Code Polymorphism Meets Code Encryption: Confidentiality and Side-channel Protection of Software Components. Digital Threats: Research and Practice, 4(2), 1–27.

[6] Caliandro, P. (2025). VMorph: A Virtualization/Metamorphic Framework for Binary Obfuscation and Intellectual Property Protection. CEUR-WS. https://ceur-ws.org/Vol-3962/paper22.pdf