

메모리 버스 스누핑 및 치환 공격 가능성과 한계

Yeonjun Lee <yeonfish6040@gmail.com>¹

¹ Korea Digital Media Highschool / Department of Web programming

초록 — 메모리 값이 오르고 저가 모듈을 찾는 사람이 늘수록, 공급망에 변조된 모듈이 섞여 들어갈 여지도 함께 커진다. 본 논문은 그런 모듈 안에 숨은 장치가 CPU와 DRAM 사이에서 무엇을 할 수 있는지를 평가한다. 다루는 공격은 다음과 같다: 장치가 메모리에 쓰이는 내용을 들여다보아 표적을 찾아내고 그 자리의 DRAM 좌표를 기억해 둔 뒤, 나중에 같은 주소에 접근할 때 버스를 끊고 조작한 값을 대신 출력한다. 이 방식은 물리주소와 DRAM 좌표 사이의 비공개 매핑을 풀지 않아도 된다는 점에서 기존 인터포저 공격과 차별화된다. 본 논문은 이 공격 모델을 제안·설계하고, 같은 설계가 실제 시스템에서 어디서 무너지는지를 타이밍 예산·캐시 계층·비트 패턴 검출·메모리 암호화의 네 한계로 나누어 정량화한다.

Keywords — Memory Bus MITM, DRAM Interposer, Data Bus Substitution Attack, Content Fingerprinting, Linux, Page Cache, inode, DDR4, Intel TME, Memory Snooping, Intel TEE, LLC

1. 서론

1.1. 연구 배경

대규모 언어 모델과 생성형 AI의 폭발적 성장으로 인하여 고 대역폭·고용량 메모리 수요는 전례 없는 수준으로 상승하였고, 이로 인한 DIMM의 공급 부족은 가격의 상승과 함께, 소비자의 저가 제품을 찾는 심리를 자극한다. 이러한 저가 유통 경로는 변조된 하드웨어가 정상 공급망에 섞여 들어갈 수 있는 벡터가 된다. 본 연구는 이 지점에서 변조된 DIMM이 저가 유통 경로를 타고 들어오는 하드웨어 공급망 공격을 출발점으로 삼는다.

한편 최근의 TEE 연구에서는 CPU 패키지 바깥의 메모리를 아예 적대적인 것으로 보는 위협 모델에 관한 관심이 증가하고있다. 클라우드와 기밀 컴퓨팅이 퍼지면서 “장비에 물리적으로 접근할 수 있는 공격자”가 현실적인 가정이 되었기 때문이다. 그 결과 메모리 버스와 모듈 자체를 신뢰하지 않는 설계가 늘었다. 본 연구는 이러한 배경 위에서 진행된 선행 연구들의 기법들을 기반으로, DRAM 칩이 아닌 메모리 컨트롤러와 DRAM 사이의 버스에서의 데이터 치환형 중간자 공격을 설계하고 그 실현 가능성을 따진다.

1.2. 배경지식

본 연구를 보기에 앞서, 공격 대상인 “사용자 데이터”가 물리적으로 어디에 어떤 형태로 존재하며 언제 버스에 나타나는지를 이해하는 것이 좋다. 리눅스 커널은 파일 데이터를 페이지 캐시에 평문으로 상주시키며 [1], 편집 중인 문서·복호화된 미디어·데이터베이스 페이지, 그리고 파일시스템 메타데이터(inode 테이블 블록 등)가 DRAM 위에 CPU 관점의 평문으로 놓인다. 다만 CPU와 DRAM 사이의 다단계 캐시로 인하여, 트랜잭션이 실제로 버스에 나타나는 시점은 최종 레벨 캐시(LLC) 미스에 따른 채움(읽기)과 더티 캐시라인의 방출(eviction, 쓰기)로 한정된다 [2]. 본 공격은 바로 이 쓰기 트래픽에서 표적을 식별하고 읽기 트래픽에서 위조를 수행한다.

1.2.1. DDR4 프로토콜

DDR4에서 DIMM(램스틱)은 RANK, BANKGROUP, BANK, ROW, COLUMN 과 같은 요소로 구성된다. CPU에 내장되어 있는 메모리 컨트롤러는 DRAM 칩의 집합인 DIMM들의 핀에 병렬로 연결된다. 데이터를 읽을 때는 먼저 CS[0]#·CS[1]# 으로 이용할 RANK를 고르고, ACT 명령으로 CID(3DS)·BG·BA·ADDR을 조합해 ROW를 연 다음, READ 명

령으로 ADDR에 해당하는 COLUMN을 선택하여 한 버스트에 64바이트(8비트 × 8버스트 × 8레인)를 가져온다. 쓸 때도 순서는 같고 마지막 명령만 WRITE 로 바뀐다. 중요한 점은 이 명령들이 주소를 평문으로 실어 나른다는 것이다. 버스에 끼어든 장치는 명령만 읽어도 매 전송이 어느 좌표를 향하는지 알 수 있다.

1.2.2. 리눅스 페이지 캐시

본 연구에서 주로 다루는 공격면은 리눅스의 페이지 캐시이다. 리눅스는 파일을 읽고 쓸 때 페이지 캐시를 거쳐 성능을 확보한다. 파일을 읽으면 커널은 먼저 페이지 캐시에서 해당 파일의 folio를 찾고, 없으면 보조기억장치에서 읽어 캐시에 적재한 뒤 그 내용을 돌려준다. 파일을 쓰면 해당 페이지 캐시를 수정·적재하고 더티 플래그를 세운 다음, 일정 조건을 만족할 때 디스크로 내려보낸다 [3]. 이 과정에서 파일 내용과 메타데이터가 모두 DRAM에 평문으로 놓이는 구간이 생기며, 그 구간이 본 공격의 관측 창이다.

1.3. 관련 연구

수동 버스 스누핑. Membuster [2] 는 DDR4 주소 버스를 수동 관측해 enclave의 접근 패턴만으로 기밀을 복원한다. WireTap [4] 은 데이터(DQ) 버스를 수동 스누핑하고 결정론적 AES-XTS의 성질로 암호문 사전을 구축해 SGX attestation 키를 추출한다. 두 공격 모두 수동 관측에 초점을 두며 데이터를 능동적으로 바꾸지 않는다.

주소선 aliasing. BadRAM [5] 은 SPD를 위조해 정적 aliasing을, Battering RAM [6] 은 런타임 주소선 접지로 동적 aliasing을 일으켜 SEV-SNP/SGX의 무결성과 attestation을 무너뜨린다. 이들은 주소를 조작하지만 특정 콘텐츠를 표적으로 삼지는 않는다.

DRAM 주소 매핑 리버스 엔지니어링. 표적 주소를 알려면 물리주소에서 (뱅크, 행, 열)로 가는 매핑이 필요한데, 이는 비공개이며 DRAMA [7], DRAMDig [8], Sudoku [9] 가 행버퍼 충돌 타이밍으로 이를 선형 XOR 함수로 복원한다. 본 공격은 뒤에서 설명할 구조적 이유 때문에 이 매핑을 미리 복원하지 않고, 버스에 흐르는 콘텐츠만으로 표적을 특정한다.

1.4. 본 논문의 기여

기존 인터포저 공격이 수동 스누핑 [2] 또는 주소선 aliasing [5], [6] 에 머문 반면, 본 논문은 콘텐츠 트리거형 능동 읽기 위조라는 새로운 축을 제시한다. 구체적 기여는 네 가지다.

첫째, 정상시에는 모든 신호를 그대로 통과시키며 쓰기 내용을 훑어보는 상시 필터 도메인, 걸러진 후보를 분석해 표적을 정의하고 위조 데이터를 준비하는 느긋한 도메인, 그리고 읽기 순간에 응답을 바꿔치기하는 초저지연 하드와이어드 도메인으로 역할을 나눈 삼중 도메인 구조를 정의한다.

둘째, ext4의 inode·extent·슈퍼블록처럼 페이지 캐시에 놓이는 구조의 시그니처를 잡아내는 방법을 제시하고, 이를 물리 계층에 밀착한 병렬 패턴 매치와 뒤단 정밀 판정의 2단 구조로 설계하여 실현 가능성을 따진다.

셋째, 명령/주소 라인을 직접 보는 하드와이어드 비교기로 매치 신호를 만들고, 이를 단방향 직렬 차단과 데이터·스트로브 재구동에 직결하는 출력단을 회로 수준에서 상술한다.

넷째, 그렇게 만든 설계가 실제 시스템에서 어디서 무너지는지를 네 개의 한계로 나누어 정량 평가한다.

2. 위협 모델

공격자와 삽입 지점. 공격자는 변조 DIMM을 저가 유통하거나(공급망), 데이터센터 유지보수 중 잠시 물리 접근하여(내부자), 메모리 컨트롤러와 DRAM 모듈 사이에 라이저형 중간자 장치를 삽입한다. 장치는 CPU와 DRAM 양쪽에 대해 전기적으로 투명한 통과 경로를 기본 상태로 갖는다.

능력. 장치는 명령/주소 버스를 관측해 매 전송의 종류와 DRAM 좌표를 파악하고, 쓰기 버스트의 내용을 검사·필터링하며, 걸러진 데이터를 내부 프로세서로 넘겨 분석·위조한 뒤 관심 좌표를 감시 목록에 기록한다. 그리고 감시 좌표에 대한 읽기가 오면 실제 DRAM의 데이터 구동을 끊고 자신이 준비한 데이터를 스트로브 타이밍에 맞춰 대신 내보낸다. 표적을 정하는 일은 시간 여유가 있는 쪽에서, 읽기 순간의 판정과 구동은 명령 라인에 직결된 회로에서 수행된다(5. 공격 설계: 콘텐츠 트리거형 읽기 위조 중간자). 능동 구동을 위해 채널은 정격 이하로 다운클럭될 수 있다고 가정한다.

대상 모듈. 본 논문은 ECC UDIMM 을 가정한다. ECC 모듈에서는 위조 데이터에 맞는 ECC 심볼도 함께 만들어 내보내야 하는데, 서버 컨트롤러의 ECC 코드는 대개 비공개이고 x4 심볼 정정까지 쓰이고, 6.4. 한계 III — 비트 패턴 검출의 가정에서 다룰 스크램블러·스와즐과 같은 성격의 조건이 하나 더 생긴다. 하지만 Cojocar [10] 에서 부팅시 ZERO-FILL을 이용한 MASK 복원에 대한 내용을 보였으므로, 가능하다고 가정하며 메인 공격 모델을 집중하여 설명한다.

은닉성. 장치는 부팅·POST·메모리 트레이닝·측정 부팅 단계에서는 완전히 통과 모드로만 동작하여 어떤 무결성 검사도 통과하며, 공격은 런타임에 선택적으로 활성화된다.

목표. 주된 목표는 무결성 침해다. 특정 페이지 캐시 구조(예: inode)를 읽어 들이는 커널·프로세스에 공격자가 조작한 값을 반환하여 권한 상승·데이터 위조·코드 주입을 달성한다. 부차적으로 쓰기 감시를 통한 기밀성 침해도 제한적으로 가능하다.

3. 연구 질문과 분석 방법

3.1. 연구 질문

본 연구는 공격의 성공 여부를 하나의 이진 주장으로 다루지 않고 네 질문으로 나눈다. 첫째, 표적 구조가 실제로 쓰기 버

스트에 나타나는가, 그리고 1단 필터가 걸러 올리는 후보는 얼마나 되는가. 둘째, 읽기 명령을 본 뒤 DRAM이 데이터를 내보내기 전까지 차단과 출력 준비가 끝나는가. 셋째, 위조한 데이터가 자기 스트로브의 유효 구간 안에 들어오며 속도를 올릴수록 그 여유가 어떻게 변하는가. 넷째, 전력·감시 항목 수·버퍼 크기·플랫폼 변환 처리를 함께 고려할 때 어느 구성이 후속 실험의 가치가 있는가.

3.2. 가정을 다루는 방식

본 연구의 모든 수치는 실측이 아니라 공개 규격과 부품 데이터시트에서 유도한 값이다. 그래서 가정을 세 층으로 나누어 관리한다. 규격 고정값은 버스트 길이·버스 폭·명령 순서처럼 프로토콜이 정해 둔 값이다. 부품 범위값은 비교기 전파지연, 스위치 개방 시간, 출력 인에이블 지연처럼 데이터시트가 범위로 주는 값이다. 미확정 시스템값은 인터포저를 끼웠을 때의 삽입손실, 메모리 트레이닝 성공률, 실제 스크램블러 함수처럼 플랫폼 실험 없이는 알 수 없는 값이다. 결과 수치는 앞의 두 층만으로 계산하고, 세 번째 층은 별도의 위험 요소로 따로 적어 둔다.

기준 동작점은 다운클럭된 DDR4-1333이며(클럭 주기 1.5 ns, 읽기 지연 13.5 ns), 속도는 800에서 3200 MT/s까지 훑는다. 관측되는 쓰기량은 초당 100만 캐시라인을 기준으로 잡고 만에서 천만까지 훑는다. 이 두 값이 이후 계산의 출발점이다.

3.3. 무엇을 어떻게 계산했는가

읽기 위조의 타이밍은 사건 순서를 그대로 늘어놓고 계산했다. 행을 여는 명령, 읽기 명령, 첫 스트로브 프리앰블, 버스트의 여덟 구간, 스위치 재접속을 시간 축에 배치한 뒤 두 종류의 여유를 본다. 하나는 **명령 여유**로, 읽기 명령을 본 순간부터 DRAM이 실제로 데이터를 내보내기 시작할 때까지 남은 시간에서 장치의 판정·차단·출력 준비에 드는 시간을 뺀 값이다. 여기에는 명령 라인의 레벨을 논리값으로 바꾸는 시간, 좌표 일치를 따지는 논리 게이트 지연, 클럭으로 래치하는 시간, 매치 신호가 기판을 타고 가는 시간, 스위치가 열리는 시간, 출력이 유효해지는 시간, 그리고 모형을 넣지 못한 변동을 위한 가드밴드가 들어간다. 다른 하나는 **데이터 창 여유**로, DRAM이 한 비트를 유효하게 유지하는 짧은 구간에서 클럭과 레인과 스트로브의 정렬 오차, 그리고 수신측이 요구하는 셋업·홀드 시간을 뺀 값이다. 두 여유가 모두 양수인 경우만 그 회차를 통과로 세고, 부품 지연을 데이터시트 범위 안에서 무작위로 뽑아 20만 번 반복해 통과율을 구했다. 정렬 오차는 표준편차의 여섯 배까지 보수적으로 잡았다.

콘텐츠 검출부는 초당 후보 수로 평가한다. 관측되는 쓰기량에 필터별 통과확률을 차례로 곱하는 방식이며, 필터는 매직 상수 일치, 인접 필드의 범위 정합, 그리고 공격자가 심어둔 고유 마커의 일치다.

마지막으로 공격 한 회차의 성공확률을 생각한다. 공격이 한 번 성공하려면 다섯개의 조건을 순서대로 모두 통과해야 한다. 표적이 실제로 버스에 나타나야 하고, 버스에 나타난 바이트에서 그것이 표적임을 알아봐야 하고, 위조하는 동안 무관한 트래픽과 충돌하지 않아야 하고, 타이밍 제약을 만족해야 하고, 위조한 값이 커널의 검증을 통과해야 한다. 중요한 것은

이것이 모두 만족되어야 한다는 점이다. 6. 한계 정량 분석 은 다섯 조건 각각의 값과, 각 관문이 실패하는 조건을 차례로 따진다.

4. 배경: 데이터 경로와 표적 구조

이 장은 표적이 되는 구조가 DRAM에 어떤 형태로 나타나는지, 언제 버스에 노출되는지, 버스 물리 계층은 어떻게 생겼는지, 그리고 왜 이 공격이 물리주소 매핑을 풀 필요가 없는지를 설명한다.

4.1. 리눅스 가상 메모리와 페이지 캐시

각 프로세스는 페이지 테이블로 가상주소를 물리주소로 변환하며, 커널은 물리 메모리 전체를 선형 매핑으로 보유한다. 파일 I/O 성능을 위해 읽고 쓴 데이터는 페이지 캐시에 보관되고, 현대 커널은 struct page 대신 folio 추상화 [11] 로 이를 관리하며 각 address_space 의 내용을 파일 오프셋으로 색인된 XArray에 저장한다 [1]. 쓰기는 folio를 더티로 표시하고, 특정 조건에서 디스크로 writeback된다. 슈퍼블록·inode 테이블 블록·디렉터리 블록 같은 파일시스템 메타데이터도 버퍼/페이지 캐시를 거치므로, 이들이 디스크로 내려가기 전후로 DRAM에 쓰이는 순간 이 존재한다. 이 순간이 본 공격의 관측 창이다.

4.2. 표적 구조의 콘텐츠 시그니처

본 공격의 핵심은 쓰기 버스트의 내용만으로 “이것이 inode 테이블 블록”임을 낮은 오답으로 판별하는 것이다. ext4를 예로 들면 세 가지 단서가 유효하다.

슈퍼블록에는 매직 넘버 0xEF53 이 오프셋 0x38 에 리틀엔디언(53 EF)으로 들어 있다 [12], [13]. inode는 통상 256바이트이며, extent 기능을 쓰는 inode는 i_block 영역 선두, 즉 inode 시작으로부터 오프셋 40에 eh_magic 값 0xF30A(리틀엔디언 0A F3)를 둔다 [12]. 하나의 inode 테이블 블록에는 이런 inode가 조밀하게 채워지므로, 64바이트 캐시라인 단위의 쓰기에서 이 매직이 규칙적으로 반복 관측된다. 마지막으로 inode와 folio 서술자에는 커널 정규 주소(예: 0xfffff8... 대역)로 보이는 값들이 들어 있어, “상위 비트가 0xffff 로 시작하는 8바이트 정렬 값”이라는 형상 필터가 보조 지표가 된다.

이 매직들과 필드 제약(합리적인 i_mode, i_links_count, 정상 범위의 크기·타임스탬프)을 결합하면 쓰기 버스트만으로도 표적 구조를 실용적인 정밀도로 식별할 수 있다.

4.3. 캐시 계층, LLC, 메모리 컨트롤러 스케줄링

코어의 로드/스토어 대부분은 캐시에서 처리되어 버스에 나타나지 않는다. 버스에 실제로 트랜잭션이 나타나는 경우는 LLC 미스에 따른 캐시라인 채움과 더티 라인의 방출뿐이며, x86-64의 64바이트 캐시라인은 DDR4 버스트 길이 8에 정확히 대응한다. 따라서 관측과 위조의 최소 단위는 하나의 캐시라인이다.

LLC 포함성과 방출 유도. Intel 소비자 CPU의 LLC는 포함형이라 L1/L2의 라인이 LLC에도 존재하며, 서버는 비포함형에 포함 디렉터리를 쓴다만[14] 본 논문에서 공격대상으로 다루는 Xeon E5-2680 v4는 포함형이며 물리주소에서 LLC 슬라이스로 가는 해시는 비공개이나 리버스 엔지니어링되어 있다 [15]. 표적을 버스로 끌어내려면 그 라인이 캐시에서 방

출되어야 하므로, 공격자는 공유 매핑이 있으면 clflush 로, 없으면 같은 캐시 셋에 합동인 주소들을 모은 eviction set을 접근해 방출을 유도한다. Vila 등은 eviction set을 선형 시간에 축소하는 알고리즘을 제시했고 [16], 캐시라인을 특정하지 않고 전체를 덮어버리는 방법도 알려져 있다 [14] 또한 포함형 캐시에서 공통 LLC eviction으로 타 코어의 L1,L2 캐시를 만료시키는 [17] 공격도 존재한다. 즉 표적 라인을 DRAM 미스로 만드는 과정은 가능하다고 가정할 수 있다.

메모리 컨트롤러 스케줄링. 컨트롤러는 열려 있는 행에 적중하는 요청을 오래된 요청보다 우선하는 FR-FCFS 계열 정책을 널리 채택한다 [18]. 이 때문에 세 가지 일이 벌어진다. 버스 상의 명령 순서가 프로그램 순서와 달라지므로 인터포저는 뒤섞인 스트림을 좌표 기준으로 정리해야 한다. 하지만 쓰기는 큐에 모였다가 한꺼번에 배수되므로 inode 테이블 블록의 writeback이 군집해 도착해 캐치에는 오히려 유리하다.

이를 통하여 본 논문에서 가정할 수 있는 항목은 **inode 블록이 방출될 때 쓰기가 버스에 나타나 지문 검출 대상이 되고, 커널이 그 블록을 다시 참조하면서 LLC 미스로 DRAM에서 채울 때만 위조 응답이 전달되며, 공격자는 방출과 재적재의 창을 능동적으로 만들 수 있다.** 라고 볼 수 있다.

4.4. 리눅스 커널 코드 경로: inode의 생성·기록·재적재 위조 대상이 커널 안에서 어떤 경로로 DRAM에 기록되고 다시 읽히는지를 아는 것은 표적 유발과 타이밍의 근거가 된다.

chmod·chown·touch 같은 inode 필드 변경은 mark_inode_dirty() 를 거쳐 결국 ext4_do_update_inode() 에서 메모리상의 구조체를 디스크 레이아웃으로 직렬화한다 [19]. 이 함수는 i_mode·i_uid·i_gid·i_size·타임스탬프·i_block[] 을 채운 뒤 inode 메타데이터 체크섬(crc32c)을 계산해 기록하고 버퍼를 더티 표시한다. 더티 버퍼는 저널 커밋과 플러시를 통해 DRAM으로 방출된다. 반대로 캐시에서 밀려난 inode가 다시 필요해지면 ext4_iget() 에서 __ext4_get_inode_loc() 로 inode 테이블 블록을 읽어 오는데, 버퍼 캐시가 미스면 이 시점에 DRAM 읽기가 발생 한다. 읽은 뒤 커널은 체크섬을 검증하여 불일치 시 오류로 거부한다.

inode의 위치는 산식으로 좁힐 수 있다. 블록 그룹은 inode 번호를 그룹당 inode 수로 나눈 몫이고, 그룹 안의 인덱스는 그 나머지며, 블록 안의 바이트 오프셋은 인덱스에 inode 크기(기본 256바이트)를 곱한 값이다 [20]. 공격자 측 오케스트레이터는 이 산식으로 표적 inode가 놓일 자리를 좁힌다.

세 겹 정합 요구. 위 경로에서 드러나듯 inode 필드 위조는 값만 바뀌서는 통하지 않는다. 첫째, 커널이 재적재 때 검증하는 crc32c 체크섬을 다시 계산해 붙여야 한다. 실패하면 커널은 inode를 거부하고 파일시스템을 읽기 전용으로 리마운트한다. 둘째, 데이터 버스 반전이 켜져 있으면 그 규칙에 맞게 내보내야 하고, 스캐램블러가 있으면 같은 마스크로 다시 스캐램블해야 한다(5.3. 버스 반전 되돌리기와 디스크램블).

체크섬 계산 자체는 어렵지 않다. ext4 체크섬은 Castagnoli 다항식의 crc32c이며 [21], 커널은 부팅 시 슈퍼블록 UUID로 시드를 미리 계산해 둔다. inode별 시드는 여기에 inode 번호와 generation 값을 차례로 접어서 만들고, 최종 체크섬은 체크섬 필드를 0으로 둔 256바이트 레코드 전체에 대한 crc32c의 하위 16비트다. 인터포저는 이 계산을 읽

기가 오기 전 여유 있는 시간에 미리 해 두면 된다. 슈퍼블록 UUID는 0xEF53 지문으로 포착한 슈퍼블록에서 스누핑하거나 오케스트레이터가 제공하고, inode 번호와 generation은 각각 오케스트레이터와 관측된 inode에서 얻는다. 요컨대 위조 페이로드는 평균 필드, 체크섬, 버스 반전 규칙의 세 겹이 모두 맞아야 커널의 검증을 통과한다.

4.5. DDR4 물리 계층과 DRAM 좌표

DDR4 인터페이스는 차동 클록, 명령/주소 버스, 데이터 버스로 구성된다. 명령/주소 버스는 명령 신호와 뱅크 그룹·뱅크·주소선을 다중화하여 명령과 행·열 주소를 전달한다. 데이터는 64비트가 8개 바이트 레인으로 나뉘고, 각 레인은 차동 스트로브로 소스 동기 전송되며 양방향이다. 한 접근은 행을 여는 ACTIVATE, 열을 고르는 READ/WRITE, 필요 시 PRECHARGE 순서를 따른다. 따라서 중간자 장치는 명령/주소 버스를 디코딩하는 것만으로 매 버스트의 좌표와 방향을 알 수 있다.

다만 행 주소는 조합논리로 바로 얻어지지 않는다. READ/WRITE 명령에는 뱅크와 열만 실리고 행 주소는 앞선 ACTIVATE에만 있으므로, 장치는 랭크와 뱅크마다 지금 열려 있는 행을 상태로 기억하고 PRECHARGE 나 리프्रेस, 저전력 상태 진입에서 그 상태를 지워야 한다. 2랭크 16뱅크 구성이면 32개의 행 레지스터가 필요하다. 좌표는 조합논리의 출력이 아니라 상태 기계의 출력이며, 이 상태가 오염될 때 어떤 일이 벌어지는지는 6.4. 한계 III — 비트 패턴 검출의 가정에서 다룬다.

4.6. 왜 물리주소와 DRAM 좌표의 매핑을 풀 필요가 없는가

기존 인터포저 공격은 “피해자의 물리주소가 어느 DRAM 좌표인가”를 알기 위해 비공개 매핑을 리버스 엔지니어링해야 했다 [7], [9]. 반면 본 공격은 표적을 주소가 아니라 콘텐츠로 식별한다. 중간자 장치는 버스에서 DRAM 좌표를 직접 관측하므로, “inode 시그니처가 관측된 바로 그 좌표”를 기록해 두었다가 같은 좌표의 읽기를 위조하면 된다. 즉 물리주소와 매핑 함수는 공격 경로에서 완전히 우회되며, 이것이 본 방식의 핵심 실용적 이점이다. 하지만 매핑을 알아낼 수 있다면 표적 좌표군을 넓히거나 이웃 좌표를 예측하는 데 쓸 수 있으니, 선택적 보조 수단으로 둘 수 있다.

4.7. TME 암호화 엔진의 위치

Intel TME의 AES-XTS 엔진은 LLC와 메모리 컨트롤러 사이, 외부 버스로 나가는 경로에 위치한다 [22], [23]. 칩 내부에서는 평문이지만 버스로 나가는 순간 128비트 키로 암호화되며, XTS의 tweak이 물리주소에서 유도되어 “같은 평문에 같은 주소면 같은 암호문”이라는 결정론이 성립한다. TME는 기밀성만 제공하고 무결성과 신선도는 제공하지 않는다. 이 두 성질이 6. 한계 정량 분석의 평가를 가른다. 특히 결정론은 6.5. 한계 IV — 메모리 암호화에서 다루는 암호문 사전 공격을 가능하게 하는 성질이기도 하다.

5. 공격 설계: 콘텐츠 트리거형 읽기 위조 중간자

5.1. 상태 기계 개요

중간자 장치는 네 상태를 오간다. 평상시에는 모든 신호를 통과시키며 명령과 데이터를 관측만 하는 통과·감시 상태다. 쓰기 버스트에서 표적 시그니처를 인식하면 매직을 거쳐 그 좌

표를 감시 목록에 등록하는 감시 상태로 옮겨 간다. 이후 감시 좌표에 대한 읽기 명령을 명령 버스에서 감지하면 가로채기 상태로 진입하여 실제 DRAM의 데이터 구동을 끊고 위조 데이터를 내보낸 뒤 다시 통과 상태로 돌아온다. 쓰기는 어느 상태에서도 항상 통과시키며, 위조는 오직 등록된 좌표의 읽기에서만 일어난다.

5.2. 쓰기 감시와 콘텐츠 지문 엔진

지문 엔진은 통과하는 각 쓰기 버스트를 복사하여 세 가지를 규칙 기반으로 평가한다. 매직 상수가 기대 오프셋에 있는지, 인접 필드가 합리적 범위인지, 커널 포인터 형상 값이 있는지다. ext4 inode는 256바이트로 정렬·패킹되고 extent 트리 루트가 각 inode의 오프셋 40에 놓이므로, inode 테이블 블록에서는 256바이트마다 첫 캐시라인의 오프셋 40에 0A F3이 규칙적으로 반복된다 [12]. 이 “캐시라인 단위, 고정 오프셋” 성질이 아래에서 볼 인터리빙에도 견고한 검출을 가능케 한다.

교차 코어·DMA 인터리빙과 좌표 기반 처리. 메모리 컨트롤러는 여러 코어와 DMA 에이전트의 요청을 행 적중률과 지연 기준으로 재정렬하므로, 하나의 inode 테이블 블록을 이루는 버스트가 시간적으로 연속되지 않고 무관한 다른 좌표의 쓰기와 뒤섞여 도착한다. 따라서 엔진은 도착 순서가 아니라 좌표를 기준으로 동작한다. 각 쓰기를 랭크·뱅크그룹·뱅크·행·열의 조합으로 키잉해 소형 쓰기 캐시에 적재하는 방식이며, 이때 행은 앞서 말한 대로 뱅크별 열린 행 상태에서 얻는다.

여기서 성립하는 것과 성립하지 않는 것을 구분해야 한다. **라인 내부**의 오프셋은 매핑과 무관하게 복원된다. 버스트 경계가 곧 64바이트 캐시라인 경계임을 DDR4 프로토콜이 보장하므로, 매직이 놓이는 오프셋 40은 물리주소와 좌표 사이의 매핑을 전혀 몰라도 그대로 읽힌다. 반면 **라인 사이**의 관계는 복원되지 않는다. “256바이트 정렬”이나 “인접 좌표”는 물리주소 공간의 성질이고, 좌표 공간에서 그에 대응하는 비트가 무엇인지는 매핑 함수를 알아야 나오기 때문이다. 게다가 컨트롤러는 연속 접근을 서로 다른 뱅크 그룹으로 흩어 성능을 얻으려고 낮은 주소 비트를 뱅크 그룹 선택에 섞으므로 [7], [9], 한 inode를 이루는 네 캐시라인이 인접 좌표에 떨어진다 는 보장 자체가 없다.

본 설계가 단일 라인 표적에 의존하는 근거가 바로 이 비대칭이다. 개별 캐시라인의 매직 검출과 그 라인의 위조는 인터리빙·매핑과 무관하게 성립하는 반면, 여러 라인을 모아 붙이는 작업만이 매핑 리버스 엔지니어링이라는 추가 가정을 요구한다. 따라서 재조립은 기본 경로가 아니라 깊은 extent 트리나 디렉터리 블록 같은 심화 목표에만 조건부로 쓰인다. 한편 동일 좌표에 재기록이 오면 캐시 항목을 최신 값으로 갱신하므로, 커널의 후속 inode 수정도 자연스럽게 반영된다.

스크램블러가 켜져 있는 경우. 다수 플랫폼은 콜드 부트 잔류 데이터 노출과 동시 스위칭 잡음을 줄이기 위해 버스로 나가는 데이터를 주소 기반 의사난수로 화이트닝하는 스크램블러를 적용한다. 이 경우 버스 위의 바이트는 평문이 아니어서 매직이 그대로 보이지 않으므로, 지문 검출에 앞서 복원이 필요하다. 다행히 이 스크램블러는 선형 구조라 리버스 엔지니어링이 가능하며 [24], 엔진은 명령 디코딩으로 얻은 좌표에서 디스크램블 마스크를 계산해 버스트를 되돌린 뒤 매직을

찾는다. 다만 시드는 부팅할 때마다 새로 정해지므로, 마스크 계산에 앞서 장치가 런타임에 시드를 복원하는 단계가 반드시 선행되어야 한다(5.3. 버스 반전 되돌리기와 디스크램블). 반면 TME가 활성화되면 화이트닝이 AES-XTS로 대체되어 선행 역산이 불가능해지므로 이 지문 단계 자체가 무력화된다(6.5. 한계 IV — 메모리 암호화).

배경 트래픽. 비ECC 모듈에는 patrol 스크러빙이 없으므로 코어와 무관한 배경 트래픽은 주로 DMA 에이전트에서 온다. 좌표와 콘텐츠를 함께 보는 검출은 이런 트래픽에 견고하다. 다만 감시 좌표에 코어가 아닌 주체의 읽기가 닿으면 그 읽기도 위조 응답을 받으므로, 감시 좌표는 가능한 한 좁게 잡아야 한다.

중요한 성질은 지문 판정이 쓰기 응답을 지연시킬 필요가 없다는 점이다. 쓰기는 항상 통과시키고, 판정과 등록은 다음 읽기가 오기 전까지만 끝내면 된다.

5.3. 버스 반전 되돌리기와 디스크램블

버스에서 관측한 원시 비트는 평문이 아니다. 정확한 지문과 체크섬 계산을 위해 두 변환을 역으로 풀어야 하며, 위조할 때는 같은 변환을 정방향으로 다시 적용해야 한다. 순서는 컨트롤러와 DRAM이 적용한 것의 역순으로, 원시 데이터에서 버스 반전을 먼저 되돌리고 그다음 디스크램블하면 평문이 나온다.

데이터 버스 반전(DBI) 되돌리기. DDR4의 선택적 DBI는 한 구간에서 바이트 레인의 여덟 비트 중 논리 0이 네 개를 넘으면 그 바이트를 반전한다. 논리 0 구동에 비트당 15 mA가량이 들기 때문이며, 반전 여부는 바이트 레인마다 구간마다 별도로 라인으로 신호된다. 이 라인은 데이터 마스크 핀과 공유되어 모드 레지스터로 둘 중 하나를 택한다 [25], [26]. 판단 기준은 그 라인이 LOW면 반전된 데이터, HIGH면 참값이다 [26]. 따라서 인터포저는 데이터 레인뿐 아니라 이 라인도 함께 탭하여, LOW인 바이트를 다시 반전해 원래 값을 복원한다. 위조할 때는 반대로 위조 바이트의 0 개수를 세어 네 개를 넘으면 반전해서 내보내며 라인을 LOW로, 아니면 그대로 내보내며 HIGH로 구동한다.

여기서 실용적으로 중요한 성질이 하나 나온다. 반전은 바이트 레인 단위 로 계산되므로, 캐시라인의 일부 바이트만 바꾸는 부분 위조에서도 그 바이트가 실린 레인의 그 구간만 다시 판정하면 되고 나머지 레인은 원본 신호를 그대로 통과시키면 된다. 즉 부분 위조와 버스 반전이 국소적으로 정합한다. 쓰기 CRC나 명령 패리티가 켜진 경우에는 인터포저가 그 모드들도 인지하여 맞춰야 한다 [26].

스크램블러 디스크램블. Intel 등은 신호 무결성과 콜드부트 완화를 위해 주소에서 유도한 의사난수 수열을 데이터에 XOR하는 스크램블러를 둔다. 이 스크램블러는 여러 개의 LFSR을 병렬로 돌리는 선형 구조라 리버스 엔지니어링이 가능하다 [24], [27]. 다만 시드는 부팅할 때마다 새로 뽑히고, 여기에 접근 주소가 섞여 좌표별 마스크가 만들어진다 [24]. 따라서 마스크는 공장에서 미리 넣어 둘 수 있는 상수가 아니라, 부팅할 때마다 장치가 스스로 복원해야 하는 값이다.

여기서 자연스럽게 나오는 의문은 “그 복원에 CPU가 읽은 값이 필요하지 않느냐”는 것이다. 필요하지 않다. 스크램블러

의 구조, 즉 다항식과 탭과 주소를 섞는 방식은 CPU 세대마다 한 번만 알아내면 되는 오프라인 작업이고, 그 구조가 정해지면 마스크는 시드 비트에 대해 선형이다. 런타임에 남은 미지수는 시드 하나뿐이며, 버스를 관측하는 것만으로 세 가지 경로가 열린다.

첫째, 평문이 0인 구간 이 있다. 펌웨어의 메모리 초기화와 커널의 페이지 제로화는 0으로 가득 찬 쓰기를 대량으로 버스에 올린다. 평문이 0이면 관측된 데이터가 곧 마스크 자체이므로 그 좌표의 마스크가 직접 얻어진다.

둘째, 평문 값을 몰라도 평문이 같다는 사실만으로 제약을 모을 수 있다. 서로 다른 두 좌표에 같은 평문이 쓰이면 두 관측값을 XOR했을 때 평문이 소거되고 시드에 대한 선형 방정식만 남는다. 제로 페이지, 반복되는 트레이닝 패턴, 동일 좌표 재기록이 모두 그 공급원이며, 이렇게 모은 방정식은 가우스 소거로 푼다.

셋째, 후보 시드가 맞는지 판정하는 데에도 CPU 측 값이 필요 없다. 방정식을 다 풀어도 독립 제약이 시드 비트 수보다 적으면 해가 하나로 떨어지지 않고 여러 후보가 남는데, 이 후보들을 하나씩 대입해 디스크램블한 결과가 정위치에서 ext4 매직을 내놓거나 inode 레코드의 crc32c와 맞아떨어지는지 보면 된다. 틀린 시드는 사실상 난수를 만들어 내므로 32비트 체크섬이 우연히 맞을 확률은 40억분의 1 수준이고, 따라서 이 검사는 사실상 결정적인 판정기다. 요컨대 선형 풀이가 후보를 크게 줄이고 체크섬 검사가 그중 하나를 확정한다. 다만 이 방식이 성립하려면 남은 후보 수가 장치가 하나씩 대입해 볼 수 있는 규모여야 하며, 그 조건이 만족될 때까지 관측을 계속해야 한다.

정리하면 시드 복원은 공격자가 알려진 평문을 써 넣는 능동적 훈련이 아니라 관측 트래픽만 쓰는 수동 절차다. 이 점에서, 대상 보드에 사전 접근해야만 하는 DQ 스와치 훈련 (6.4. 한계 III — 비트 패턴 검출의 가정)과는 성격이 다르다. 대가는 부팅 직후 시드가 확정될 때까지의 관측 공백이다. 디스크램블은 같은 마스크를 다시 XOR하는 대칭 연산이라 위조 시 재스크램블도 동일하며, XOR과 조건부 반전 모두 비트 단위 선형 연산이라 캡처·위조 경로의 지연에 수백 피코초만 더한다.

통합 파이프라인. 캡처 경로는 데이터와 스트로브를 잡아 버스 반전을 조건부로 되돌리고 좌표에서 계산한 마스크를 XOR하면 평문이 나온다. 위조 경로는 그 역순으로, 평문 위조 바이트에 체크섬을 붙이고 마스크를 XOR한 다음 버스 반전 규칙을 적용해 반전 라인과 함께 내보낸다. 이 파이프라인이 4.4. 리눅스 커널 코드 경로: inode의 생성·기록·재적재 에서 말한 세 겹 정합을 하드웨어에서 실현한다.

5.4. 물리 계층 병렬 패턴 매치와 2단 검출

모든 쓰기 버스트를 프로세서로 넘겨 소프트웨어로 검사하면 데이터 이동량과 전력이 감당되지 않는다. 본 설계는 검출을 2단으로 나눈다. 1단은 물리 계층에 밀착한 조합논리 패턴 매치가 상시 동작하며 극소수 후보만 통과시키고, 2단은 통과한 라인만 FIFO를 거쳐 프로세서가 디스크램블·필드 정합·재조합 같은 정밀 판정을 수행한다. 이로써 프로세서는 후보가 올

라울 때만 깨어나 평균 소비가 급감하며, 이것이 저전력 구성의 전제가 된다.

매치 하나는 순수 조합논리다. 관심 없는 비트를 마스킹하고 관심 비트만 기대값과 XNOR로 비교한 뒤 전부 AND로 묶는 구조이며, 여러 패턴의 결과를 OR로 합쳐 후보 신호를 만든다. 클럭이 필요 없어 정적 소비가 마이크로암페어급이고, 각 매치가 자기 오프셋과 비트폭과 기대값만 보므로 매직·구조체 필드·문자열·고유 마커를 제각기 전용화할 수 있다. 어느 매치가 맞았는지를 비트벡터로 함께 올리면 프로세서가 inode 인지 슈퍼블록인지 디렉터리인지를 즉시 분류하여 파싱 분기를 건너뛰는다.

여기에 세 가지 최적화를 얹는다. 첫째는 오프셋 게이팅이다. 쓰기 버스트는 여덟 구간으로 시간분할되어 도착하므로, 매직이 놓인 바이트가 어느 구간에 실리는지를 명령에서 얻은 버스트 카운터로 판정하여 해당 구간에만 매치를 동작시킨다. 이는 오탐과 스위칭 전력을 함께 줄인다. 둘째는 고정과 프로그래머블의 혼합이다. 자주 쓰는 매직류는 고정 매치로 상시 두고, 표적마다 바뀌는 고유 마커는 기대값을 레지스터로 갱신하는 소수의 프로그래머블 매치로 재사용한다. 셋째는 2계층 게이팅이다. 1계층은 “매직이 있는가” 같은 값싼 필요조건만 보고, 비싼 필드 정합과 고유 마커 비교는 1계층을 통과한 라인에만 활성화한다. 설계 원칙은 미탐을 0으로 두고 오탐에는 관대하게 가는 것으로, 충분조건 확정은 뒤로 미룬다.

스크램블러가 켜지면 버스의 매직이 화이트닝되어 고정 상수 매치가 깨진다. 대응은 두 가지다. 1단을 스크램블에도 변하지 않는 성질(엔트로피 같은 통계)로 느슨히 잡고 정밀 디스크램블은 뒤에서 하거나, 관심 오프셋의 디스크램블 마스크를 미리 계산해 매치의 기대값 자체에 XOR로 반영하는 것이다. 후자는 좌표에 따라 값이 달라지므로 소수의 핫좌표에만 적용한다. TME가 켜지면 사전 없이는 어느 쪽도 불가능하다(6.5. 한계 IV — 메모리 암호화).

5.5. 블록 재조립과 구조 디코딩

버스는 논리 구조를 64바이트 캐시라인 단위로 조각내고, 게다가 순서까지 뒤섞어 전달한다. 따라서 엔진은 조각을 좌표로 키잉해 버퍼링하고, 필요한 조각이 모이면 디코딩하며, 그 결과에 따라 inode 위조와 데이터 블록 주소 추출로 분기한다. 여기서 핵심은 inode의 첫 캐시라인 하나에 필요한 것이 모두 들어 있다는 점이다. 권한 필드와 extent 헤더 매직, 그리고 첫 extent의 데이터 블록 포인터가 전부 0 63바이트 구간에 있다(표 1). 즉 위조와 주소 추출 모두 단일 라인으로 가능하며, 여러 라인을 모으는 작업은 깊은 extent 트리나 디렉터리 블록 같은 경우에만 필요하다.

필드	오프셋	용도
i_mode	0	권한/타입 → setuid 위조
i_uid / i_gid	2 / 24	소유자 → root(0) 위조
i_links_count	26	필드 정합성 검사
eh_magic	40	0xF30A 앵커
eh_entries / eh_depth	/ 42 / 46	엔트리 수 / 리프 여부
ee_block / ee_len	/ 52 / 56	논리 블록 / 범위 길이
ee_start_hi / ee_start_lo	58 / 60	데이터 블록 물리주소

표 1: 디코딩에 쓰는 ext4 inode 오프셋(256바이트 inode 기준). 모두 첫 캐시라인 안에 있다.

좌표 기반 부분 재조립. 기본 경로에서 버퍼 슬롯을 여는 것은 앵커 라인, 즉 오프셋 40에 매직을 가진 첫 라인뿐이며, 이 판정은 앞서 말한 대로 매핑을 요구하지 않는다. 권한 위조와 extent 리다이렉트는 이 라인 하나로 완결되므로 여기서 이미 디코딩이 시작된다. 128바이트 이후의 필드가 필요한 심화 목표에 한해서는 앵커가 아닌 라인을 어느 inode의 몇 번째 조각인지 귀속시켜야 하는데, 이는 물리주소 공간의 정렬 조건이므로 매핑 리버스 엔지니어링이 선행되어야 한다. 매핑을 모르는 상황에서 앵커가 아닌 라인은 귀속이 불가능하므로 그대로 버린다. 완결 전의 조각은 버퍼에 머무르고, 같은 조각이 다시 쓰이면 최신값으로 갱신되며, 오래된 미완결 슬롯은 타임아웃으로 방출한다.

디코딩과 분기. 앵커 라인을 파싱하면 권한 필드와 extent 포인터를 얻는다. 그다음 이 inode가 정말 표적인지 확인해야 하는데, 방법은 두 가지다. 공격자 오케스트레이터가 표적 inode에 미리 심어 둔 고유 마커(특이한 크기나 타임스탬프)로 콘텐츠 매칭하거나, 디렉터리 블록을 따로 지문해서 파일명과 inode 번호의 대응을 얻는 것이다. 표적이 확정되면 목적에 따라 갈라진다. 권한 상승이 목적이면 i_mode 에 setuid 비트를 세우고 소유자를 root로 바꾼 뒤 체크섬을 다시 계산해 프리로드한다. 데이터 리다이렉트가 목적이면 extent의 시작 블록 번호를 공격자가 통제하는 블록으로 바꾼다. 파일 내용 자체를 보고 싶다면 시작 블록 번호를 추출한 뒤, 오케스트레이터가 pagemap [28] 으로 그 블록의 페이지 캐시 물리주소를 알아내고, 그 물리주소에 해당하는 DRAM 좌표를 관측으로 확보해 데이터 블록 자체를 감시 대상으로 등록한다.

깊은 extent 트리나 디렉터리 기반 파일 식별에서는 여러 블록과 라인이 필요하므로 위의 재조립이 그 조각들을 모아야 하는데, 이 경로는 매핑을 아는 경우에만 성립한다. 따라서 매핑을 확보하지 못한 플랫폼에서 본 공격의 사정거리는 앵커 하나로 완결되는 목표로 제한된다. 모든 파싱은 리틀엔디언과 경계 검사를 포함하며, 오탐은 매직과 필드 범위의 결함으로 억제된다(6.4. 한계 III — 비트 패턴 검출의 가정).

5.6. 좌표 부기와 이중 도메인 구조

본 설계의 핵심 원칙은 역할 분리다. 표적을 정의하는 쓰기 감시·지문·재조립은 시간 여유가 있는 느긋한 프로세서 도메인에서 수행하고, 위조 응답을 만드는 읽기 가로채기는 펌픽 왕복을 배제한 초저지연 하드와이어드 도메인에서 수행한다.

두 도메인은 감시 목록이라는 작은 상태만 공유하며, 앞의 것이 이를 기록하고 뒤의 것이 조회한다.

인식된 표적 좌표와 대응 페이로드는 프로세서 쪽 메모리에 저장되지만, 읽기 경로의 실제 주소 매칭은 그 메모리가 아니라 다음 절의 하드와이어드 비교기가 담당한다. 프로세서는 감시 항목이 확정될 때마다 좌표 상수를 비교기의 레지스터에 로드하고 위조 페이로드를 출력단 근처의 고속 버퍼에 미리 넣어 둘 뿐이다. 즉 프로세서는 “느린 두뇌”, 하드와이어드 회로는 “빠른 반사”다. 감시 항목이 소수의 핫셋을 넘으면 핫셋만 비교기 어레이에 두고 나머지는 프로세서 쪽에서 처리하는 2티어 구성을 취한다.

5.7. 하드와이어드 저지연 트리거

읽기 위조의 시간 예산을 지배하는 것은 “감시 좌표에 대한 읽기인가”를 판정하는 지연이다. 이를 프로세서 펌픽에서 하면 입력 버퍼부터 직렬화, 라우팅, 비교, 출력까지 여러 클럭이 소요된다. 본 설계는 이 판정을 명령/주소 라인에 직접 결선된 회로로 옮겨 게이트 지연 수준으로 줄인다.

DDR4의 명령/주소 라인은 단일 종단 신호이므로 먼저 기준 전압과 비교하는 입력 리시버로 논리값화한다. 그다음 감시 좌표의 각 비트에 대해, 원하는 값이 0인 라인은 반전한 뒤 전부 AND로 묶는다. 명령/주소는 30비트급이라 단일 게이트로는 불가능하므로 2 3단 AND 트리로 구성한다. 같은 주소 핀이라도 행을 열 때와 열을 고를 때 의미가 다르므로, 명령 신호의 극성을 함께 몰려 “이 사이클이 표적 열에 대한 읽기일 때만” 매치가 성립하도록 조건을 건다. 행 일치와 열 일치를 각각의 게이트로 두고 둘을 AND하면, 행을 열고 나서 읽기가 오기까지의 시간 간격이 회로 구조에 자연스럽게 반영된다.

명령/주소는 클록 동기 신호이므로 조합 AND 출력을 그대로 쓰면 글리치와 준안정 상태가 생긴다. 따라서 AND 결과를 차동 클록으로 한 번 래치하여 그 사이클에 유효한 매치만 뽑아낸다. 이는 DRAM이 명령을 샘플하는 것과 같은 타이밍 기준을 쓴다는 이점이 있다. 다만 명령이 유효해지는 시점과 AND 전파지연을 고려해 클록을 약간 지연복제한 예지로 샘플해야 하고, 비교기의 차동 출력은 곧바로 스위치 선택 신호와 출력 인에이블에 결선하며, 프로세서로 가는 사후 통지 경로에만 동기화 플립플롭을 두어 준안정 상태를 흡수한다. 적합한 부품으로는 래치 입력을 갖춘 나노초 이하급 고속 비교기가 있다. 이 매치 신호를 프로세서 인터럽트로 올리면 인터럽트 지연이 도로 붙으므로, 신호는 하드웨어 트리거 라인으로만 쓰고 펌웨어를 경유시키지 않는다.

5.8. 초저지연 출력단

매치 신호가 뜨면 DRAM의 실제 응답을 끊고 위조 데이터를 그 자리에 구동해야 한다. 핵심 관찰은 개입이 읽기 방향에만, 그것도 아주 짧은 창에만 필요하다는 점이다. 쓰기와 주소, 제어 라인은 항상 통과하고 쓰기 방향 데이터도 손대지 않으므로, 완전한 양방향 멀티플렉서는 불필요하며 라인당 직렬 차단 스위치 하나와 트라이스테이트 구동 하나면 충분하다.

각 데이터 스트로브 라인에는 고속 FET 버스 스위치를 직렬로 두어 평소에는 통과시키다가 매치 신호로 DRAM 쪽을 개방한다. FET 스위치는 삽입 지연이 사실상 없는 수동 소자이며 전환에는 나노초 안팎이 걸린다. 읽기 데이터는 읽기 지

연이 지난 뒤에야 DRAM이 구동하므로, 매치 시점에 미리 개방해도 정상 트랙픽을 해치지 않고 여유만 확보한다. DDR4의 출력 방식은 종단 기반이라 DRAM을 끊지 않고 컨트롤러 쪽에서 덮어쓰면 드라이버 충돌이 생기므로, 최소한의 DRAM 측 차단은 필수다.

위조할 캐시라인은 매치 이전에 이미 출력단 근처의 홀딩 레지스터에 프리로드되어 있다. 개념적으로는 데이터를 미리 넣어 두고 한 번의 스트로브에 전 비트를 내보내는 병렬 레지스터와 같지만, DDR4 정격에서는 일반 로직으로 스트로브 정렬과 비트별 디스큐가 불가능하므로 그 기능을 프로세서의 직렬화기와 지연 소자로 구현한다. 직렬화기는 프리로드된 병렬 워드를 버스트로 방출하고, 출력 인에이블이 매치 신호에 의해 열린다.

데이터가 소스 동기이므로 위조 스트로브도 함께 만들어야 한다. 위조 스트로브는 별도 채널로, 데이터 대비 90도 위상에서 데이터 창 중앙에 오도록 구동한다. 위상은 PLL로 만든 시프트 클록으로 직렬화기를 클럭킹하거나, 데이터와 스트로브 경로에 각각 미세 지연 소자를 두어 레인별로 맞춘다. 읽기 프리앰블과 포스트앰블도 이 채널이 재현하여 컨트롤러의 스트로브 게이팅이 위조 신호를 받아들이게 한다.

여기서 중요한 이점이 하나 나온다. 자체 스트로브 기준 정렬은 DRAM의 스트로브 대 클록 출력 스큐(3200에서 대략 ± 0.19 ns이며 읽기마다 변동)를 통째로 흡수한다. 인터포저가 DRAM의 스트로브를 격리하고 위조 데이터와 스트로브를 함께 자체 기준으로 구동하므로, 위조 데이터는 DRAM의 지터와 무관하게 자체 스트로브의 데이터 창 중앙에 놓인다. 다만 DRAM 스트로브에서 자체 스트로브로의 전환은 컨트롤러의 스트로브 게이팅이 열리기 전, 즉 읽기 프리앰블 구간에 완료되어야 하는데, 프리앰블 자체를 자체 스트로브로 재현하므로 이 경계가 자연히 맞는다.

본 논문은 비ECC 모듈을 가정하므로 ECC 심볼을 따로 구동하는 경로는 필요 없다. ECC 모듈로 확장하려면 위조 데이터에 맞는 심볼도 함께 내보내야 하는데, 서버 컨트롤러의 ECC 코드는 대개 비공개이므로 이는 6.4. 한계 III — 비트 패턴 검출의 가정 의 스크램블러스와줄과 같은 종류의 “모르면 0”인 조건을 하나 더 만든다.

5.9. 읽기 위조 타이밍 예산

기준 동작점은 다운클럭된 DDR4-1333으로, 클록 주기가 1.5 ns, 읽기 지연이 13.5 ns, 버스트 데이터 창이 6 ns다. 부품은 다음과 같이 잡았다. 데이터 스트로브 차단에는 대역폭 2 GHz급 FET 버스 스위치(도통 저항 2 Ω , 개방 시간 15 ns)를, 위조 구동에는 프로세서의 직렬화기와 지연 소자(출력 인에이블에서 유효 출력까지 약 0.5 ns, 지연 스텝 2.5 ps)를, 명령 매치에는 전파지연 150 ps에 차동 래치 입력을 갖춘 고속 비교기를, 좌표 일치 판정에는 단당 13 ns인 저지연 단일 게이트를 2 3단으로 쓴다. 두뇌 역할은 DDR4 물리 계층과 대용량 온칩 메모리, 프로세서 코어를 한 칩에 갖춘 MPSoC가 맡는다.

이 값들을 누적하면 임계경로는 표 2 과 같다.

단계	근거	지연
명령 레벨을 논리값으로	고속 비교기	0.15 ns
좌표 일치 AND 트리	저지연 게이트 3단	~3.0 ns
클록 래치	고속 비교기	0.15 ns
매치 신호 라우팅	기판 + 버퍼	~0.5 ns
데이터·스트로브 스위치 개방	FET 스위치	~2.0 ns
출력 인에이블에서 유효 출력	직렬화기	~0.5 ns
임계경로 합계		≈ 6.4 ns
가용 읽기 지연	DDR4-1333	13.5 ns
여유		≈ 7.1 ns (약 53%)

표 2: 읽기 위조 임계경로 예산(DDR4-1333). 매치와 차단 경로가 읽기 지연의 절반 이하로 끝난다.

흥미로운 점은 읽기 지연이 속도 등급을 올려도 CAS 지연이 함께 커져 대략 13 15 ns로 거의 일정하다는 것이다. 따라서 임계경로 관점의 여유는 어느 등급에서도 넉넉하다. 실제 병목은 읽기 지연이 아니라 버스트 안에서의 구간별 정렬이다. 단위 구간이 1067에서 937 ps인 데 반해 3200에서는 312 ps로 좁아져 스트로브 정렬과 비트별 디스큐가 급격히 어려워진다. 즉 다운클럭의 정당성은 “차단할 시간이 부족해서”가 아니라 “정렬을 쉽게 하려고”에 있다. 이 구분은 6. 한계 정량 분석에서 중요해진다.

전체 흐름은 두 창을 파이프라인화한다. 행을 여는 명령 시점에 행 일치가 확정되어 감시 목록을 활성 행으로 좁히고, 읽기 명령 시점에 열 일치와 명령 조건이 클럭으로 래치되어 매치 신호가 뜨며, 같은 순간 DRAM측 스위치를 열고 출력 인에이블을 개방하여 읽기 지연이 지난 시점에 위조 버스트를 방출한다.

5.10. 캐시 일관성과 타이밍 고려

세 가지 실질적 제약이 있다. 첫째는 가시성이다. 표적 구조가 CPU 캐시에 있으면 읽기가 버스에 나타나지 않으므로, 공격자는 표적 라인이 방출되어 다음 참조가 DRAM 미스가 되도록 캐시 압박이나 cflush로 창을 만들어야 한다. 둘째는 쓰기와의 경합이다. 지문 판정 전에 읽기가 도착하면 그 회차는 위조하지 못하나 다음 방출·재적재 회차에서 성립한다. 셋째는 검사 시점과 사용 시점의 차이다. 측정 부팅이나 dm-verity는 로드하고 측정하는 시점의 무결성만 보장하므로, 장치가 측정 중에는 통과하고 이후 재적재 시에만 위조하면 소프트웨어 무결성 검증을 우회한다. 한편 메모리 컨트롤러의 재정렬로 버스 명령 순서가 프로그램 순서와 다를 수 있으나, 장치는 재정렬된 스트림을 그대로 관측·대응하므로 문제되지 않는다.

5.11. 하드웨어 구현과 비용

느긋한 도메인은 하드 DDR4 물리 계층과 대용량 온칩 메모리, 제어 코어를 갖춘 MPSoC로 구현한다. AMD Zynq UltraScale+ 계열은 이 셋을 한 칩에 제공하여 지문·재조립·오케스트레이션을 통합하기에 적합하며, 오픈소스 DRAM 컨트롤러의 물리 계층 제어 코드를 재활용할 수 있다. 하드와이어드 도메인은 명령 리시버, 저지연 AND 게이트, 래치형 고속 비교기, 데이터·스트로브 차단용 FET 버스 스위치, 주소선

아날로그 스위치로 구성하며, 전체는 DIMM 소켓과 모듈 사이에 삽입되는 4층 이상 라이저 기판에 실장한다.

비용은 수동 관측이나 주소선 조작 구성(수십 수백 달러 [4], [6]) 보다 높지만, 뒤에서 볼 부분 위조 변형이나 저전력 변형은 대형 MPSoC를 피해 수백 달러 규모로 낮출 수 있다. 비용의 대부분은 부품값이 아니라 라이저 기판 설계와 출력단 신호 무결성에 드는 공학 비용이다. 참고로 WireTap이 수동 캡처를 위해 채널을 1333 MT/s로 낮춘 사례는 고속 DDR 인터포저의 계측 난도를 보여주는 비교점이다 [4].

5.12. 정격 후보 변형: 다상 클럭과 부분 위조

앞 절들은 출력단의 구간별 정렬 난이도 때문에 다운클럭을 전제했다. 그러나 다운클럭 없이 정격에 접근할 가능성도 별도 후보로 평가할 수 있다. 핵심은 두 가지다. 위조 재구동을 상용 물리 계층 대신 이산 소자와 PLL 다상 클럭으로 직접 구성하는 것, 그리고 전 64비트가 아니라 표적 바이트가 실리는 소수 레인과 소수 구간만 부분 위조하여 소자 규모를 실용 범위로 낮추는 것이다.

직렬화란 결국 “여덟 개의 병렬 비트를 미리 로드해 두고 구간 간격의 스트로브로 하나씩 방출하는 것”에 지나지 않는다. 따라서 필요한 것은 여덟 값을 담는 고속 래치와 각 구간 경계를 정확히 찍는 다상 클럭뿐이다. 인터포저는 DIMM의 차동 클럭을 기준으로 체배·분주하여 구간 간격(3200에서 312.5 ps)으로 등간격 정렬된 여덟 개의 위상을 만들고, 위조 스트로브는 같은 PLL에서 90도 시프트된 탭으로 뽑는다. 정격에서 결정적 제약은 위상 사이의 스큐다. 구간이 312 ps이므로 여덟 위상의 상호 스큐가 수 피코초 이내여야 아이가 닫히지 않으며, 이를 위해 저지터 PLL과 길이를 맞춘 클럭 분배, 레인별 미세 지연 보정이 필요하다. 각 구간에서 천이가 312 ps 안에 끝나야 하므로 래치와 멀티플렉서는 기가헤르츠급 소자여야 하고, 일반 CMOS 레지스터는 톨한 한계로 부적합하다.

표적 바이트레인 부분 위조. 전체 레인을 이산 직렬화하면 소자가 500개를 넘어 기판이 감당하지 못한다. 그러나 본 공격의 위조는 캐시라인 전체가 아니라 몇 바이트만 바꾼다. 권한 필드는 2바이트씩이고 extent 포인터는 6바이트다. 64바이트 캐시라인에서 이 바이트가 실리는 특정 레인과 특정 구간만 위조하면 되고 나머지는 DRAM 신테이터를 그대로 통과시키면 된다. 즉 차단도 전체가 아니라 표적 레인만 끊고, 그 레인의 해당 구간에만 위조 비트를 삽입한다. 이렇게 하면 권한 위조에 필요한 직렬화 소자가 두어 개에서 십여 개 규모로 줄어들며, 정렬해야 할 레인 수가 줄어 스큐 관리 대상도 함께 축소된다. 앞서 본 대로 버스 반전이 바이트 레인 단위로 계산된다는 점이 여기서 다시 도움이 된다.

이 변형은 다운클럭 흔적을 남기지 않아 은닉성이 높고 큰 프로세서 없이 구성되어 저렴하다. 대가로 저지터 클럭 분배가 필요하고, 표적 바이트가 정확히 어느 레인의 어느 구간에 실리는지를 알아야 하므로 DQ 스와즐과 버스 반전 상태 파악이 선행되어야 한다. 또한 이 변형은 비ECC 가정에 강하게 의존한다. ECC 모듈에서는 바이트 하나만 바뀌도 캐시라인 전체의 ECC가 달라지므로 부분 위조라는 개념 자체가 성립하지 않는다.

5.13. 정격 구현이 요구하는 정밀도

정격 DDR4-3200에서 구현을 시도한다면 무엇이 결정적인지만 짚어 둔다 [26], [29]. 클록 주기는 0.625 ns, 읽기 지연은 13.75 ns, 버스트 창은 2.5 ns, 단위 구간은 312.5 ps다. 매치와 차단 임계경로는 6.4 ns이므로 읽기 지연 대비 여전히 7 ns 이상 남는다. 즉 정격에서도 시간이 부족해서 실패하지는 않는다.

유일한 실질 제약은 데이터 유효창이다. DDR4에서 스트로브 대 데이터 레인 스큐가 클록 주기의 0.18배, 데이터 출력 홀드가 0.74배이므로, 실제로 데이터가 유효한 창은 그 차이인 클록 주기의 0.56배, 즉 구간당 약 0.35 ns에 불과하다. 위조 비트는 이 0.35 ns 안에 스트로브 대비 정렬되어야 한다. 다상 클록의 위상 스큐와 레인 스큐, 지터를 합친 오차가 이 창을 잠식하므로 총 정렬 오차를 창의 5분의 1인 70 ps 아래로, 가능하면 30 ps 아래로 눌러야 한다. 이 예산이 부품 선택을 좌우한다. 지연 스텝이 2.5 ps인 하드 물리 계층의 직렬화기는 이 예산에 여유 있게 들어가는 반면, 이산 다상 클록 구성은 저지터 PLL을 쓰더라도 경계에 놓인다. 부수적으로 두 가지 제약이 더 있다. DRAM 스트로브에서 자체 스트로브로의 전환은 12 클록의 읽기 프리앰블 안에 끝나야 하고, 2667을 넘는 속도에서 기본 활성인 geardown 모드는 명령을 클록의 절반 속도로 래치하므로 비교기의 클럭킹을 거기에 맞춰야 한다.

5.14. 저전력 변형: 단일 슬롯 셀프 파워

앞의 정격 변형은 은닉성과 속도를 취한다면, 이 변형은 반대로 저전력과 저가, 그리고 자체 전원을 취한다. 2단 검출과 단방향 차단에 강한 다운클럭을 결합하면 전력 예산상 DIMM 슬롯 전원만으로 동작하는 구성을 그려 볼 수 있다. 다만 슬롯 레일의 허용 전류와 DRAM 전원망 교란을 실측하지 않았으므로, 이는 확정 설계가 아니라 후속 전원 무결성 실험의 가설이다.

DDR4 DIMM은 1.2 V 주 전원과 함께 워드라인 부스트 전원의 2.5 V 레일을 통과 레일로 받는다. 주 전원에서 큰 전류를 빼면 DRAM의 전압 마진을 침해하므로, 인터포저 전원은 헤드룸이 있는 2.5 V 레일에서 저정지전류 레귤레이터로 만들고 벌크 커패시터로 리프레시 펄스를 받친다. 통과 레일은 두꺼운 플레인과 촘촘한 디커플링으로 보강하고 인터포저의 리턴을 DRAM 리턴과 분리한다.

셀프 파워를 성립시키려면 총소비를 수백 밀리와트 이하로 눌러야 한다. 방법은 세 가지다. 고속 비교기와 대형 프로세서를 배제하고, 상시 동작부를 조합논리 매처로만 한정하며, 프로세서는 후보가 발생할 때만 깨운다. 이렇게 하면 상시부는 마이크로암페어에서 수 밀리와트, 명령 리시버는 수십 밀리와트, 차단 스위치는 수동 소자라 사실상 0, 후보가 올라올 때만 깨는 초저전력 프로세서가 10 50 밀리와트로, 평균 총소비는 0.1 0.4 W 수준이 된다.

대가는 타이밍이다. 저전력 소자는 매치 지연이 커져서 (CMOS 비교기 약 2.5 ns, 저전력 게이트 단당 1 3 ns) DDR4-1333에서 임계경로가 11.7 ns까지 늘어나 읽기 지연 13.5 ns 대비 여유가 13%로 줄어든다. 따라서 한 단 더 강한 다운클럭(800 1066 MT/s)으로 여유를 30% 이상 회복해야

하며, 이는 단위 구간도 함께 넓혀 저전력 소자의 출력 정렬을 쉽게 만든다.

결국 저전력 변형은 세 가지 스코프 중 하나를 택하게 된다. 셀프 파워와 강한 다운클럭을 묶어 능동 위조를 유지하거나, 능동 위조를 포기하고 스누핑과 주소선 조작만 하거나, 벌크 커패시터로 위조 버스트의 순간 피크만 감당하는 것이다. 어느 경우든 부품은 수십에서 수백 달러 규모로, 정격 구성 대비 대폭 저렴하다. 이 저가성이야말로 저렴한 제품을 선호하는 조달 심리를 악용하는 본 위협 모델의 핵심을 강화한다.

6. 한계 정량 분석

앞 장까지 본 논문은 콘텐츠 트리거형 읽기 위조가 설계로서 성립하는가를 다루었다. 이 장은 방향을 뒤집어, 같은 설계가 실제 시스템에서 어디서 무너지는가를 정량화한다. 결론을 먼저 말하면, 본 공격의 실현 가능성은 회로의 정교함이 아니라 네 개의 외부 병목이 결정한다. 타이밍 예산, 캐시 계층에 의한 관측 가능성, 비트 패턴 검출의 가정, 그리고 메모리 암호화다. 이 중 둘은 값에 따라 확률을 낮출 뿐이지만 나머지는 둘은 조건에 따라 성공확률을 정확히 0으로 만든다.

6.1. 분석 프레임

3.3. 무엇을 어떻게 계산했는가 에서 말한 다섯 관문을 실제 병목에 맞추어 다시 정리한다. 한 회차가 성공하려면 표적이 재적재 읽기로 버스에 나타나야 하고(한계 II), 버스에 나타난 바이트에서 표적임을 식별해야 하며(한계 III·IV), 위조 버스트가 인접 트래픽과 충돌하지 않고 격리·재접속되어야 하고(한계 I), 각 버스트가 타이밍 제약을 통과해야 하며(한계 I), 마지막으로 위조 페이로드가 평문·체크섬·버스 반전의 세 겹 정합을 모두 만족해야 한다. 여기에 더해, 정합을 위해 동시에 위조해야 하는 캐시라인이 여러 개면 타이밍 관문은 그 수만큼 거듭 통과해야 한다(6.6. 부가 한계: 다중 캐시라인 정합 요구).

이 분해의 요점은 관문들이 곱이라는 데 있다. 어느 하나가 0이면 반복 횟수와 무관하게 전체가 0이다. 아래에서 각 관문의 값과, 각 관문이 0이 되는 조건을 차례로 제시한다.

6.2. 한계 I — 타이밍 예산

6.2.1. 데이터 창 여유가 지배 병목이다

표 3 는 3.3. 무엇을 어떻게 계산했는가 의 방식으로 20만 회 반복 계산한 결과다.

구성	MT/s	명령 여유	데이터 창 여유	통과율
저전력 CMOS	800	+4.50 ns	+0.56 ns	99.98%
저전력 CMOS	1066	+2.00 ns	+0.21 ns	87.31%
저전력 CMOS	1333	+0.50 ns	≈0 ns	33.13%
프로세서 물리계층	2400	+6.43 ns	+0.23 ns	≈100%
프로세서 물리계층	3200	+6.85 ns	+0.12 ns	99.87%
부분 위조+PLL	1333	+7.30 ns	+0.47 ns	≈100%
부분 위조+PLL	2400	+7.13 ns	+0.10 ns	94.17%
부분 위조+PLL	3200	+7.55 ns	-0.02 ns	38.89%

표 3: 20만 회 반복 계산. 통과율은 두 여유가 모두 양수인 비율이다. 채널 신호무결성과 메모리 트레이닝 실패는 포함하지 않았으므로 실제 성공률이 아니다.

결과는 5.9. 읽기 위조 타이밍 예산의 낙관적 서술을 직접 반박한다. 세 구성 모두 명령 여유는 고속에서도 양수이지만, 저전력 CMOS와 이산 PLL 구성은 데이터 창 여유가 먼저 소진된다. 즉 “임계경로 6.4 ns가 읽기 지연 13.5 ns보다 짧으므로 정격에서도 성립한다”는 결론은 충분조건이 아니다. 정격 3200에서 분석적 가능성이 남는 경로는 비트별 캘리브레이션을 갖는 하드 물리 계층뿐이며, 이산 부분 위조는 2400에서 경계, 3200에서는 비가능 구간으로 분류된다. 저전력 구성은 800에서만 여유롭고 1066에서 경계에 놓인다. 이 결과의 함의는 5.14. 저전력 변형: 단일 슬롯 셀프 파워가 제시한 “저가·정격·셀프 파워” 삼중 달성이 현재 모델에서 지지되지 않는다는 것이다.

6.2.2. 새 제약: 버스트 사이의 전환 예산

앞선 타이밍 분석은 고립된 한 번의 읽기를 가정했다. 그러나 실제 버스에서 읽기 버스트는 연속해서 도착하며, 이때 인터포저는 위조 버스트마다 DRAM측 스위치를 열었다가 다시 닫아야 한다. 이 재접속에 필요한 시간이 버스트 사이의 간격을 넘으면, 통과되어야 할 다음 버스트가 손상된다.

DDR4에서 연속 읽기의 최소 간격은 뱅크 그룹이 다르면 $t_{CCD_S} = 4$ 클록이고, 같으면 t_{CCD_L} 로 정해진다. t_{CCD_L} 은 고정 상수가 아니라 속도 등급별로 규정되어 DDR4-1600~2133에서 5 클록, 2400~2666에서 6 클록, 2933~3200에서 8 클록이다 [29]. BL8 버스트가 데이터 버스를 4 클록 동안 점유하므로 버스가 비는 간격은 t_{CCD_S} 에서 0 클록, t_{CCD_L} 에서 $t_{CCD_L} - 4$ 클록이다. 반면 FET 버스 스위치는 열고 닫는 데 각각 1.5 ns가 걸려 왕복에 최소 4.10 ns가 필요하다. 표 4이 그 비교다.

동작점	클록 주기	t_{CCD_L}	같은 뱅크 그룹 간격	전환 가능
800 (다운클럭)	2.50 ns	5 클록	2.50 ns	불가
1067 (다운클럭)	1.875 ns	5 클록	1.88 ns	불가
1333 (다운클럭)	1.50 ns	5 클록	1.50 ns	불가
DDR4-1600	1.25 ns	5 클록	1.25 ns	불가
DDR4-2400	0.833 ns	6 클록	1.67 ns	불가
DDR4-3200	0.625 ns	8 클록	2.50 ns	불가

표 4: 연속 읽기 사이의 데이터 버스 유휴 간격과 스위치 왕복 요구(4.10 ns) 비교. 800 1333은 JEDEC 속도 등급이 아니라 다운클럭 동작점이며 t_{CCD_L} 을 최소값인 5 클록으로 가정하였다. 뱅크 그룹이 다르면 간격이 아래 0이고, 같은 경우에도 간격이 1.25 2.50 ns 범위를 벗어나지 않아 어느 동작점에서도 개별 격리·재접속이 성립하지 않는다.

결론은 명확하다. 인터포저는 버스트 단위로 스위치를 토글할 수 없다. 실현 가능한 유일한 운용은 표적 라인들이 포함된 버스트 열 전체에 걸쳐 격리를 유지하고, 위조하지 않는 라인은 미리 캡처해 둔 원본을 그대로 재생해 내보내는 것이다. 이는 두 가지 추가 요구를 낳는다. 첫째, 표적이 속한 레코드 전체가 최신 상태로 캡처되어 있어야 한다. 둘째, 격리 구간 안으로 무관한 좌표의 읽기가 끼어들면 그 읽기에 잘못된 데이터가 반환되어 즉시 시스템 오류로 드러난다.

무관한 읽기가 끼어들 확률은 버스 부하가 결정한다. 여기서 중요한 것은 어느 범위의 부하인가. 가로채기는 한 DIMM의 한 랭크에서만 일어나므로, 관련 있는 값은 시스템 전역 읽기율이 아니라 그 랭크에 도달하는 읽기율이다. 4채널에 채널당 2랭크 구성이면 전역 부하는 8로 나뉜다. 보호가 필요한 구간은 위조 버스트 자체의 2.5 ns뿐 아니라 그 앞뒤의 스위치 개방·재접속 구간을 포함하므로, 단일 라인 위조 기준으로 약 10 ns로 잡는다. 이 구간에 다른 읽기가 오지 않을 확률을 랭크당 읽기율 λ 의 포아송 도착으로 계산하면 $e^{-\lambda t}$, $t = 10$ ns이며 그 값이 표 5다.

랭크당 읽기율	8랭크 전역 환산	격리 성공확률 상한
10^6 /s (유휴)	8×10^6 /s	99.0%
10^7 /s (경부하)	8×10^7 /s	90.5%
5×10^7 /s (중부하)	4×10^8 /s	60.7%
10^8 /s (메모리 바운드)	8×10^8 /s	36.8%

표 5: 랭크당 읽기율에 따른 격리 성공확률의 상한. 보호 구간 10 ns, 단일 캐시라인 위조 기준이다. 메모리 집약 워크로드에서는 위조 시도 셋 중 둘이 무관한 읽기와 충돌하며, 충돌은 조용히 실패하지 않고 반드시 손상을 남긴다. metadata_csum이 켜져 세 라인을 한 격리 구간 안에서 처리해야 하면 구간이 그만큼 길어져 이 값의 세제곱 수준으로 떨어진 다(6.6. 부가 한계: 다중 캐시라인 정합 요구).

이 값이 상한이라는 점을 강조해야 한다. 격리 구간에 끼어들 읽기를 원본으로 재생하려면 그 좌표의 최신 쓰기를 이전 에 관측해 두었어야 하는데, 감시 대상이 아닌 좌표는 쓰기 캐시에 없으므로 재생 자체가 불가능하다. 즉 끼어들 읽기는 “재

생에 실패할 수도 있는” 것이 아니라 반드시 손상된 데이터를 받는다.

즉 타이밍 한계는 “속도를 낮추면 된다”로 해소되지 않는다. 다운클럭은 데이터 창 여유를 회복시키지만 버스트 간격은 회복시키지 못한다. `tccd_l` 이 절대시간이 아니라 클럭 수로 규정되고 고속 등급일수록 클럭 수가 커지기 때문에, 유휴 간격은 800에서 3200까지 1.25 2.50 ns 범위에 갇혀 있고 다운클럭으로 늘어나지 않는다(표 4). 여기에 더해 부하가 높은 실제 서버에서는 랭크 기준으로 환산해도 격리 확률이 떨어진다. 그리고 이 공격이 조준하는 데이터센터야말로 부하가 높은 환경이라는 점에서, 이 한계는 위협 모델과 정면으로 상충한다.

6.3. 단계 II — 캐시 계층과 관측 가능성

6.3.1. 표적은 대부분 버스에 나타나지 않는다

본 공격은 DRAM 버스에 나타난 트랜잭션만 볼 수 있다.

4.3. 캐시 계층, LLC, 메모리 컨트롤러 스케줄링 에서 정리했듯 버스에 실제로 나타나는 것은 LLC 미스에 따른 채움과 더티 라인의 방출뿐이며, 나머지 접근은 전부 캐시에서 흡수된다. 문제는 본 공격의 표적인 파일시스템 메타데이터가 정확히 캐시에 잘 남는 종류의 데이터 라는 데 있다. 반복되는 `stat()`·`open()` 은 커널의 inode 캐시에서 처리되어 DRAM은 물론 LLC에도 닿지 않는다.

따라서 위조가 성립하려면 세 조건이 동시에 만족되어야 한다. 첫째, 표적 inode가 커널의 inode 캐시에서 방출되어 커널이 `ext4_iget()` 경로를 다시 타야 한다. 그렇지 않으면 inode 테이블 블록은 아예 참조되지 않는다. 둘째, 그 inode 테이블 블록이 블록 디바이스 페이지 캐시에 같은 물리 페이지로 남아 있어야 한다. 캐시에서 밀려났다면 커널은 디스크에서 다시 읽어 새로운 물리 페이지에 적재하며, 이 순간 등록해 둔 감시 좌표는 표적과 무관한 좌표가 된다. 셋째, 그 캐시라인이 LLC에 없어야 DRAM 읽기가 발생한다. 이 세 번째 조건만이 공격자가 `clflush` 나 `eviction set`으로 능동 제어할 수 있다.

6.3.2. 첫째 조건과 둘째 조건은 서로를 배제한다

앞의 두 조건이 이 공격의 구조적 급소다. 둘 다 메모리 압력이라는 같은 변수에 의해 좌우되는데, 방향이 반대다. inode 캐시를 비우려면 압력을 높여야 하지만, 압력을 높이면 블록 디바이스 페이지 캐시의 클린 메타데이터 버퍼가 먼저 회수되어 표적이 다른 물리주소로 이동한다. 게다가 리눅스의 inode 슈링크와 페이지 캐시 회수는 같은 LRU 압력에 의해 함께 구동되므로, 두 사건은 독립이 아니라 양의 상관 을 갖는다.

이것을 계산해 볼 수 있다. 메모리 압력이 커질수록 표적 inode가 캐시에서 밀려날 확률은 올라가고, 동시에 표적 페이지가 원래 자리에 남아 있을 확률은 내려간다. 두 확률을 각각 지수 형태로 놓고 곱한 값의 최댓값, 즉 공격자가 압력을 가장 잘 맞췄을 때의 관측 가능성 상한을 구하면 표 6 를 얻는다. 표의 기준은 페이지 캐시가 inode 캐시보다 몇 배 빨리 회수되는가 다.

회수 속도비	관측 가능성 상한	해석
0.5배	38.5%	메타데이터 버퍼가 inode보다 오래 남는 이상적 경우
1배	25.0%	두 캐시가 같은 속도로 회수
2배	14.8%	클린 메타데이터 버퍼가 먼저 회수(현실적)
4배	8.2%	강한 페이지 캐시 회수 편향
8배	4.3%	<code>vfs_cache_pressure</code> 상향 등 방어 튜닝

표 6: 두 조건을 동시에 만족할 확률의 이론적 상한. 속도비는 측정되지 않은 값이지만, 어떤 값에서도 상한이 40%를 넘지 못하며 현실적 영역에서는 10%대다. 여기에 LLC 미스 조건이 더 곱해진다.

이 값이 상한 임을 강조한다. 실제 관측 가능성은 여기에 LLC 미스 조건, 그리고 압력을 원하는 시점에 정확히 걸 수 있는 능력을 더 곱한 값이다. 또한 이 상한은 압력을 최적점에 맞출 수 있다고 가정하지만, 비특권 공격자가 시스템 전역 메모리 압력을 세밀히 조절할 수단은 대량 익명 메모리 할당 정도이며, 이는 그 자체로 눈에 띄는 비정상 동작이다.

6.3.3. 파생 한계: 위조의 휘발성

세 조건을 모두 통과해 위조가 성공해도, 그 값은 커널이 읽어 상위 캐시와 메모리상의 구조체에 채운 순간부터 DRAM과 무관해진다. 위조된 inode 테이블 블록은 클린 버퍼이므로 다시 `writeback`되지 않고, 커널이 그 inode를 다시 더티화하면 오히려 진짜 값 이 DRAM에 덮어써진다. 즉 위조는 지속적 번조가 아니라 일회성 관측 오도 이며, 효과가 커널 자료구조에 반영되는 단 한 번의 창에서만 의미를 갖는다. 한 번 반영되면 목적이 달성되는 시나리오에는 충분하지만, 지속적 백도어를 심는 시나리오에는 부적합하다.

6.4. 단계 III — 비트 패턴 검출의 가정

6.4.1. 후보율은 관리 가능하다

콘텐츠 지문 자체의 오탐은 지배적 문제가 아니다. 기준 쓰기율인 초당 100만 캐시라인에서 16비트 매직을 라인 안의 고정 오프셋 에서 비교하면 우연히 맞을 확률이 6만 5536분의 1이므로 기대 후보는 초당 15.26개다. 여기에 인접 필드의 범위 정합(통과율 2%)을 적용하면 초당 0.305개로 줄고, 공격자가 심어 둔 고유 마커까지 쓰면 사실상 0이 된다.

여기서 “256바이트 정렬”을 별도 필터로 세지 않는 이유는 두 가지다. 첫째, 그것은 물리주소 공간의 조건이어서 매핑 없이는 좌표 공간에서 검사할 수 없다 (5.2. 쓰기 감시와 콘텐츠 지문 엔진). 둘째, 매치가 실제로 거는 조건인 “라인 안의 고정 오프셋”은 이미 첫 항에 반영되어 있으므로 이를 다시 곱하면 이중 계상이 된다. 요컨대 본 설계가 쓰는 필터는 전부 라인 내부 조건이며, 그래서 매핑을 요구하지 않는다.

이 후보율은 버퍼 설계로 이어진다. 필드 정합 전 후보율 15.26개/s에 보존 시간 100 ms와 안전계수 16을 곱해도 동시에 살아 있는 슬롯은 25개 미만이므로 32 64개 슬롯이면 충분하다. 반면 1단 필터 없이 모든 쓰기를 저장하는 구조는 초

당 100만 라인에서 10 ms만 보존해도 1만 라인, 약 625 KiB가 필요하다. 즉 물리 계층 1단 필터는 전력뿐 아니라 메모리 용량의 성립 조건이기도 하다.

다만 이 슬롯 예산은 기준 쓰기를 자체에 선형으로 의존한다는 점을 명시해야 한다. 초당 100만 캐시라인은 랭크당 약 64 MB/s로, 6.2.2. 새 제약: 버스트 사이의 전환 예산의 유휴 경부하 구간에 대응하는 값이다. 같은 절이 보수 구성에서 가정한 메모리 바운드 부하(랭크당 읽기 10^8 /s)에 비례하는 쓰기가 동반되면 후보율도 두 자릿수 배로 올라 32 64 슬롯으로는 부족해진다. 즉 “1단 필터로 슬롯이 충분하다”는 결론과 “부하가 높아 격리가 어렵다”는 결론은 서로 다른 부하 구간의 진술이며, 두 조건을 동시에 만족시키는 운용 지점은 좁다.

6.4.2. 진짜 한계는 오답이 아니라 미답이다

문제는 매처가 보는 비트가 평문이 아니라는 데 있다. 5.3. 버스 반전 되돌리기와 디스크램블에서 정리했듯 버스의 비트는 여러 겹의 플랫폼 의존 변환을 거친다. 각각은 역산 가능하지만, 역산에 필요한 정보를 얻지 못하면 검출확률은 낮아지는 것이 아니라 0이 된다. 성격이 다른 다섯 요인이 있다.

데이터 스캔블러. 주소에서 유도된 의사난수가 XOR되어 매직 상수 매칭이 깨진다. 선형이라 리버스 엔지니어링은 가능하지만 [24], [27], 두 층으로 나누어 봐야 한다. 구조, 즉 다항식과 탭과 주소를 섞는 방식은 CPU 세대마다 한 번씩 해 두면 되는 오프라인 작업이다. 반면 시드는 부팅할 때마다 새로 뽑히므로 장치는 전원이 들어올 때마다 관측 트래픽만으로 시드를 다시 복원해야 한다. 이것은 사전작업이 아니라 런타임 제약이다. 다만 5.3. 버스 반전 되돌리기와 디스크램블에서 보였듯 이 복원은 수동적으로 가능하므로, 시드가 검출확률을 0으로 만드는 요인은 아니다. 0으로 만드는 것은 구조의 미확보다. 스캔블러 구조가 공개되지 않은 세대를 사전 리버스 엔지니어링 없이 마주치면 복원할 대상 자체가 정의되지 않는다. 시드가 남기는 것은 실패가 아니라 지연이다. 전원 인가부터 시드 확정까지의 구간에 지나간 쓰기는 해석할 수 없다. 표적인 inode 기록은 부팅 한참 뒤의 파일시스템 이벤트이므로 이 공백이 치명적이지는 않으나, “전원 인가 직후 즉시 발화”는 성립하지 않으며 재부팅이나 특정 저전력 상태 복귀처럼 시드가 재설정되는 이벤트마다 이 공백이 되풀이된다.

DQ 스와칭. 보드 배선 편의를 위해 컨트롤러는 바이트 레인 안의 비트 순서와 레인 순서를 임의로 교환한다. 이 순열은 보드마다 다르고 공개되지 않으므로 매처의 기대값 상수를 그대로 쓸 수 없다. 알려진 평문을 써 넣고 관측하는 훈련 절차가 필요하며, 이는 공격자가 대상 보드에 사전 접근할 수 있을 때만 가능하다. 스캔블러 시드와 달리 이쪽은 수동 관측으로 해결되지 않는다.

버스 반전. 바이트마다 구간마다 조건부 반전이 걸리므로 매처는 반전 신호 라인 여덟 개를 추가로 탭해야 하고, 그 탭 자체가 채널 부하와 핀 수를 늘린다.

열린 행 상태 오염. 위 셋과 성격이 다른, 좌표 쪽 미답 요인이다. 4.5. DDR4 물리 계층과 DRAM 좌표에서 정리했듯 행 주소는 명령에 실리지 않고 뱅크별 상태로 추적된다. 행을 여는 명령을 한 번이라도 놓치면(프리앰블 손실, 명령 패리티 오류, 저전력 상태 전이) 그 뱅크의 이후 쓰기 전체에

틀린 행이 붙어, 표적 좌표를 놓치거나 무관한 좌표를 표적으로 등록한다. 결정적인 것은 이 오염이 무성이라는 점이다. 데이터 자체는 멀쩡하므로 장치가 스스로 검출할 수단이 없고, 다음 프리차지나 리프레시로 그 뱅크가 재동기화될 때까지 지속된다. 검출확률에 곱해지는 항이지만 측정값이 없어 3.2. 가정을 다루는 방식의 미확정 시스템값으로 남긴다.

extent를 쓰지 않는 inode. 앵커가 extent 헤더 매직 하나뿐이므로 extent 매핑을 쓰지 않는 inode는 원리적으로 검출되지 않는다. 내용을 inode에 직접 담는 작은 파일과, i_block 60바이트에 경로를 넣는 짧은 심볼릭 링크가 여기 해당한다. 표적이 실행 파일인 권한 상승 시나리오에서는 실질 제약이 아니지만, 표적 일반화의 상한을 정한다(7.3. 파일시스템 일반화와 그 한계).

이 중 앞의 셋은 성격이 같다. 셋 중 하나라도 되돌리지 못하는 플랫폼에서는 식별 확률이 0이고, 따라서 반복 횟수와 무관하게 공격 전체가 0이 된다. 이것이 본 공격이 “임의의 서버에 꽂으면 동작하는 장치”가 될 수 없는 첫 번째 이유다. 공급망 공격이라는 위협 모델은 공격자가 대상 플랫폼을 미리 알 수 없다는 것을 전제하므로, 이 한계는 위협 모델의 핵심 가정과 직접 충돌한다.

6.5. 한계 IV — 메모리 암호화

6.5.1. TME는 평문 지문 경로를 없앤다

본 공격의 표적 획득은 버스 위의 평문을 전제한다. Intel TME [22]가 켜지면 LLC와 메모리 컨트롤러 사이의 AES-XTS 엔진이 모든 라인을 암호화하므로 버스에는 암호문만 흐른다. 매직 상수도, i_mode의 값 범위도, 256바이트마다 반복되는 패턴도 전부 사라진다. 스캔블러와 달리 AES-XTS는 선형이 아니어서 좌표에서 마스크를 유도해 되돌릴 수도 없다. 즉 5.2. 쓰기 감시와 콘텐츠 지문 엔진의 지문 엔진은 그대로는 아무것도 찾지 못한다.

그러나 “불가능”은 아니다. TME의 tweak은 물리주소에서 유도되므로 “같은 주소에 같은 평문이면 같은 암호문”이라는 결정론이 성립한다(4.7. TME 암호화 엔진의 위치). 공격자가 표적이 놓일 주소에 알려진 평문을 미리 써 넣고 그때 버스에 나타난 암호문을 기록해 두면, 그 암호문 자체가 지문이 된다. WireTap [4]이 SGX 키를 뽑을 때 쓴 것이 정확히 이 성질이다. 이것은 뒤에서 말할 “다른 표적화 축”이 아니라 같은 콘텐츠 트리거 축의 암호문 판본이므로, 본 논문의 주장은 이렇게 좁혀야 한다. **TME가 켜져 있고 주소별 암호문 사전이 없으면 식별 확률이 0이며, 따라서 몇 번을 시도하든 성공확률은 0이다.** 즉 TME는 확률을 낮추는 것이 아니라, 사전이 없는 공격자에 대해서만 경로를 없앤다.

사전을 만드는 비용은 그러나 매우 크다. 첫째, tweak이 주소마다 다르므로 사전은 좌표마다 따로 만들어야 하고 다른 좌표로 재사용되지 않는다. 둘째, 표적이 놓일 바로 그 물리주소에 알려진 평문을 쓸 수 있어야 하는데, 이는 6.3. 한계 II — 캐시 계층과 관측 가능성의 “같은 물리 페이지 유지” 조건을 사전 구축 단계에서 한 번 더 통과해야 한다는 뜻이다. 셋째, 키는 부팅마다 새로 생성되므로 사전은 재부팅과 함께 무효가 된다. 요컨대 TME는 “임의의 피해자 시스템에 꽂아 두면 동작하는 장치”라는 공급망 위협 모델을 사실상

무너뜨리고, 표적 시스템에 미리 접근해 사전을 만들 수 있는 공격자만 남긴다. AMD SME/SEV 계열도 같은 구조다 [30].

방어 관점에서 이 결과는 주목할 만하다. TME는 무결성도 신선도도 주지 않아 데이터 버스 능동 변조 일반에는 무력하다고 평가되어 왔지만, 콘텐츠로 표적을 잡는 공격에 대해서는 사전 없는 공격자를 완전히 막는다. 표적 획득과 무결성 보호를 구분하지 않은 기존 평가가 이 효과를 놓쳐 왔다.

6.5.2. 잔여 위험: 표적을 다른 수단으로 특정하는 경우

TME 하나로 데이터 버스 공격 전반이 막히는 것은 아니다. 무결성과 신선도가 없으므로, 표적을 평문 콘텐츠가 아닌 다른 방법으로 지목한 공격자는 여전히 능동 조작을 할 수 있다. 크게 두 축이 남는다. 하나는 접근 패턴으로 좌표를 지목하고 맹목적으로 재전송이나 손상을 일으키는 축이고 [2], 다른 하나는 앞 절의 암호문 사전 축이다 [4], [31]. 전자에서는 본 설계의 지문 엔진과 재조립기가 아무 역할도 하지 못한다. 반면 후자에서는 매처가 비교하는 상수가 평문 매직에서 암호문 블록으로 바뀔 뿐 5. 공격 설계: 콘텐츠 트리거형 읽기 위조 중간자 의 회로는 그대로 쓰인다. 즉 사전을 가진 공격자에게는 본 설계가 TME 환경에서도 그대로 유효하며, 이때 6.2. 한계 I — 타이밍 예산 와 6.3. 한계 II — 캐시 계층과 관측 가능성 의 타이밍·캐시 한계는 하나도 사라지지 않고 남는다.

6.6. 부가 한계: 다중 캐시라인 정합 요구

5.5. 블록 재조립과 구조 디코딩 는 inode의 첫 캐시라인 하나에 권한 필드와 extent 매직, 데이터 블록 포인터가 모두 들어 있으므로 단일 라인 위조로 충분하다고 서술했다. 그러나 ext4의 metadata_csum 기능이 켜져 있으면(현대 mke2fs 의 기본값) 이 서술은 성립하지 않는다.

inode 레이아웃에서 i_generation 은 오프셋 100, 체크섬 필드는 124와 130에 있다 [12], [20]. 64바이트 캐시라인 경계로 나누면 이들은 첫 라인이 아니라 둘째와 셋째 라인에 놓인다. 게다가 체크섬은 체크섬 필드를 0으로 둔 256바이트 레코드 전체에 대해 계산되므로 네 라인 모두의 값이 필요하다. 결과적으로 위조는 표 7 과 같이 여러 라인을 동시에 다루어야 한다.

캐시라인	포함 필드	필요 동작
라인 0 (0~63)	i_mode, i_uid, i_gid, eh_magic	위조 (권한 필드)
라인 1 (64~127)	i_generation@100, 체크섬 하위@124	위조 + 시드 획득
라인 2 (128~191)	체크섬 상위@130	위조 (64비트 기능 시)
라인 3 (192~255)	확장 타임스탬프 등	체크섬 입력으로 캡처 필수

표 7: metadata_csum 활성화 시 inode 위조에 관여하는 캐시라인. 단일 라인 위조는 불가능하며, 최소 두 개에서 세 개의 좌표를 같은 회차에 위조하고 나머지 라인은 최신 상태로 캡처하고 있어야 한다.

이 요구는 타이밍 관문을 두세 번 거듭 통과하게 만든다. 각 라인이 서로 다른 좌표에 놓이므로 하드와이어드 비교기도 그만큼 늘어나고, 세 좌표의 읽기가 같은 회차에 모두 가로채여야 하며, 6.2.2. 새 제약: 버스트 사이의 전환 예산

의 전환 예산에 따라 세 라인은 하나의 격리 구간 안에서 연속 재구동되어야 한다. 또한 체크섬 시드 계산에 필요한 i_generation 이 둘째 라인에 있으므로, 그 값을 얻기 위해서라도 5.5. 블록 재조립과 구조 디코딩 의 재조립이 반드시 선행되어야 한다. 그런데 5.2. 쓰기 감시와 콘텐츠 지문 엔진 에서 보았듯 재조립은 매핑을 알아야 성립한다. 요컨대 “단일 캐시라인으로 끝난다”는 낙관은 기본 설정의 ext4에서 성립하지 않으며, 이 조건은 매핑 리버스 엔지니어링 요구를 다시 불러들인다.

XFS v5 역시 crc32c 메타데이터 체크섬을 쓰므로 같은 다중 라인 요구가 발생하며, btrfs는 Copy-on-Write로 메타데이터 좌표 자체가 매 트랜잭션마다 이동하여 감시 좌표가 즉시 무효화된다.

6.7. 종합: 종단 성공확률과 가능성 경계

네 한계를 곱 모형에 대입한 결과가 표 8 다. 세 가지 구성과 암호화 환경을 비교한다.

관문	낙관	기준	보수	TME(사전 없음)
버스에 나타남 (한계 II) [†]	0.35	0.15	0.05	—
표적 식별 (한계 III) [‡]	0.95	0.90	0.70	0
격리 성공 (한계 I)	0.99	0.741	0.050	—
타이밍 통과 (한계 I)	0.99	0.97	0.059	—
검증 통과 (6.6. 부가 한계: 다중 캐시라인 정합 요구)	0.98 [‡]	0.95	0.90	—
한 회차 성공 확률 상한	0.319	0.092	9.3×10^{-5}	0
95% 도달 회차	8	31	32,400	∞

표 8: 종단 성공확률의 상한. 낙관 = 무암호화·스크램블러 시드 복원 완료·metadata_csum 끄(단일 라인)·DDR4-1333 하드 물리계층·유틸 버스. 기준 = metadata_csum 켜(세 라인 동시 위조)·경부하. 보수 = 정격 3200 이산 부분 위조·메모리 바운드 부하. 기준과 보수의 타이밍·격리 관문은 세 라인을 요구하므로 표 3 와 표 5 의 단일 라인 값을 세제곱한 것이다. [†] 표 6 의 이론적 상한을 그대로 쓴 값이며 여기에 LLC 미스 조건이 더 곱해지므로 실제 값은 이보다 작다. [‡] 유도된 값이 아니라 가정 값이다(6.8. 민감도와 모델 밖의 미확정 위험). TME 열은 암호문 사전이 없는 공격자에 대해 다른 관문과 무관하게 0이며, 사전을 가진 공격자에게는 식별 관문이 회복되고 나머지는 그대로 남는다.

이 표가 본 논문의 핵심 결과다. 읽는 방식을 먼저 못박아 둔다. 관문 II는 표 6 의 이론적 상한을 그대로 쓴 값이고 관문 III과 검증 관문은 유도되지 않은 가정값이므로, 아래 세 숫자는 성공확률의 추정치가 아니라 상한 이다. 본 논문이 주장하는 것은 “성공확률이 얼마다”가 아니라 “이 상한조차 어디서 무너지는가”이며, 다음 네 가지를 읽을 수 있다.

첫째, 가장 유리한 실험실 조건에서만 공격은 실용적이다. 낙관 구성은 8회 반복으로 95%에 도달하나, 이 구성은 메모리 암호화 비활성, 대상 플랫폼의 스크램블러와 스와즐 사전 확보, 기본값이 아닌 파일시스템 설정, 정격 이하 다운클럭, 그리고 유틸에 가까운 버스를 모두 요구한다. 이는 통제된 벤

치 환경이지 공급망 공격이 상정하는 임의의 피해자 시스템이 아니다.

둘째, 기본 설정에 가까운 기준 구성에서도 상한 기준 31회면 도달하지만, 한 회차는 inode 캐시 방출을 위한 시스템 전역 메모리 압력을 요구한다. 비특권 공격자에게 이 압력은 대량 익명 할당으로만 만들 수 있고 회차당 수 초에서 수십 초가 걸리므로, 공격 전체는 수 분에서 수십 분간 지속되는 비정상 메모리 압력으로 관측면에 드러난다. 즉 기준 구성의 한계는 확률이 아니라 은닉성 이다.

셋째, 정격 속도와 고부하를 동시에 요구하는 보수 구성, 즉 실제 데이터센터 조건에서는 한 회차 성공확률 상한이 9.3×10^{-5} 로 떨어져 3만 회 이상이 필요하다. 이 구성에서는 격리와 타이밍 두 관문이 모두 세 라인을 요구하며 각각 세제곱으로 들어가는 것이 지배적이다. 6.2.2. 새 제약: 버스트 사이의 전환 예산 에서 보았듯 실패 회차는 조용히 끝나지 않고 무관한 읽기에 잘못된 데이터를 반환하므로, 수만 회의 시도는 대량의 데이터 손상과 커널 패닉을 남긴다. 비ECC 모듈에서는 이 손상이 오류 로그로 남지 않아 원인 추적은 어렵지만, 시스템이 계속 죽는다는 사실 자체는 그대로 드러난다. 이 구성은 확률적으로 가능하되 실질적으로 자기 파괴적 이다.

넷째이자 가장 중요하게, TME나 AMD SME가 커진 플랫폼에서는 암호문 사전이 없는 공격자에 대해 다른 모든 관문과 무관하게 성공확률이 0이다. 사전을 만들려면 표적 주소에 알려진 평문을 미리 써 넣을 수 있어야 하는데, 이는 “모르는 시스템에 쫓아 두면 동작한다”는 공급망 전제와 양립하지 않는다. 최신 서버에서 메모리 암호화가 기본으로 켜지는 추세를 함께 보면, 콘텐츠 트리거 측은 시간이 지날수록 적용 대상이 줄어드는 공격 측이다.

정리하면 표 9 와 같다. 네 한계 중 둘은 확률을 낮추는 데 그치지만, 나머지 둘은 조건에 따라 성공확률을 아예 0으로 만든다.

한계	지배 요인	효과
I-a 데이터 창	유효창 0.35 ns, 정렬 오차	확률 저하
I-b 전환 예산	tccd 유티 간격 < 스위치 왕복	설계 제약 + 확률 저하
II 캐시 계층	두 조건의 반상관, 물리 페이지 무효화	상한 38.5% 이하
III 패턴 검출	스크램블러·스와즐·버스 반전 역산	미확보 시 0
IV 메모리 암호화	AES-XTS 비선형성	사전 없으면 0
부가 정합	metadata_csum 다중 라인	관문 반복 통과 요구

표 9: 한계 요약.

6.8. 민감도와 모델 밖의 미확정 위험

타이밍 향의 민감도를 보면 결론을 지배하는 것이 무엇인지가 분명해진다. 정렬 오차를 25% 줄이면 이산 PLL 구성이 경계에서 가능성 구간으로 올라오고, 반대로 25% 늘리면 비가능으로 떨어진다. 즉 정렬 보정의 효과가 가장 크다. 스위치 개방 시간을 2 ns 더 늘려도 두 구성 모두 가능성 구간에 남고 경로 지연 전체를 25% 늘려도 마찬가지인데, 이는 거시적 지연

이 1차 병목이 아님을 뜻한다. 반면 실제 데이터 유효창이 추정보다 20% 좁으면 하드 물리 계층 구성마저 경계로 내려가고 이산 구성은 비가능이 된다. 요컨대 결론을 좌우하는 것은 스위치 지연이 아니라 정렬 오차와 실제 데이터 창 크기다.

세 변형은 하나의 우열관계가 아니라 파레토 경계를 이룬다. 저전력 CMOS는 0.1 0.4 W와 단일 슬롯 전원이라는 장점이 있으나 800 MT/s 부근에서만 가능성 구간에 든다. 하드 물리 계층 구성은 2.5 W와 외부 전원·열설계를 요구하지만 타이밍 모델에서 가장 강하다. 부분 위조와 PLL 조합은 0.6 1.5 W와 소수 라인 구현이 가능하나 2400 이상에서 정렬 캘리브레이션이 필수다. “저가·정격·셀프 파워”의 동시 달성은 현재 모델로 지지되지 않는다.

끝으로 위 수치가 포함하지 못한 위험을 명시한다. 첫째, 라يز러와 스위치의 기생 성분이 채널의 아이를 단거나 메모리 트레이닝을 실패시키면 장치는 런타임에 도달하지도 못한다. 이는 단일 계수로 전체 결과를 0에 가깝게 만들 수 있는 미측정 변수다. 둘째, 실제 스크램블러 구조와 DQ 스와즐은 플랫폼마다 다르고 공개되지 않을 수 있으며, 부팅마다 바뀌는 시드의 런타임 복원에 걸리는 시간은 측정된 바 없다. 셋째, 버스 반전과 쓰기 CRC의 사용 여부는 컨트롤러 설정에 따라 달라진다. 넷째, 표적이 버스에 나타나는 빈도는 워크로드·저널링·메모리 압력에 강하게 의존하며 6.3. 한계 II — 캐시 계층과 관측 가능성 의 회수 속도비는 측정되지 않았다. 다섯째, 표 8 의 표적 식별 관문(0.95/0.90/0.70)과 검증 통과 관문(0.98/0.95/0.90)은 본문에서 유도된 값이 아니라 배정한 가정값이다. 식별 관문은 스크램블러·스와즐·버스 반전을 이미 역산했다는 전제 아래 남은 잔여 미탐, 즉 6.4.2. 진짜 한계는 오탐이 아니라 미탐이다 의 열린 행 상태 오염을 나타내는데 그 발생률은 측정된 바 없고, 검증 관문 역시 체크섬·버스 반전 정합의 실패율을 직접 측정한 값이 아니다. 두 관문을 모두 1로 두면 상한은 낙관 0.343, 기준 0.108, 보수 1.47×10^{-4} 로 올라가지만 결론의 순서는 바뀌지 않으며, 반대로 열린 행 상태 오염이 흔하다면 세 값 모두 비례해 내려간다. 따라서 후속 연구는 수동 인터포저의 전기적 검증, 읽기 경로를 건드리지 않는 캡처 전용 보드, 저속 단일 라인 재구동, 다중 레인과 정격 확장의 순서로 진행해야 하며, 각 단계에서 위 한계 중 하나가 탈락 기준이 된다.

7. 시나리오 재평가

7.1. 시나리오와 한계 귀속

6. 한계 정량 분석 의 한계를 네 공격 시나리오에 적용하면, 시나리오마다 무너지는 지점이 다르다.

(S1) inode 위조를 통한 권한 상승. 표적 inode가 방출될 때 좌표를 등록하고 재적재 시 권한 필드를 조작한 값을 반환한다. 한 번의 반영으로 목적이 달성되므로 위조의 휘발성 한계에 강하다. 대신 metadata_csum 으로 인한 다중 라인 요구와 inode 캐시 방출 요구를 정면으로 받는다. 네 시나리오 중 성립 가능성이 가장 높지만, 그것이 곧 실용성을 뜻하지는 않는다.

(S2) extent 포인터 위조를 통한 데이터 리다이렉트. 시작 블록 번호를 공격자 통제 블록으로 바꾼다. S1과 같은 라인, 같은 제약을 공유되되, 리다이렉트가 의미를 가지려면 위조된

extent가 여러 번 참조되어야 하는데 첫 참조 이후 값이 캐시에 상주하므로 휘발성이 직접 작용한다.

(S3) 코드 페이지 위조를 통한 코드 주입. 가장 강력해 보이나 실제로는 가장 약하다. 실행 파일의 코드 페이지는 페이지 캐시에서 회수되는 순간 디스크에서 새 물리 페이지로 재적재되므로 등록된 감시 좌표가 필연적으로 무효화된다. 반대로 페이지가 캐시에 남아 있으면 커널은 그 페이지를 다시 읽지 않는다. 즉 S3는 6.3. 한계 II — 캐시 계층과 관측 가능성의 두 조건이 원리적으로 양립하지 않는 사례이며, 측정 부팅 우회라는 매력적인 서술과 달리 본 설계로는 성립하지 않는다.

(S4) 설정 파일이나 인증서 위조. S3와 같은 구조적 문제를 갖는다. 다만 자주 갱신되는 설정 파일은 페이지 캐시에 더티 상태로 오래 머물러 둘째 조건을 만족할 여지가 있어 S3보다는 차이가 있다.

시나리오	주 병목	부 병목	평가
S1	inode 캐시 방출	다중 라인 정합	조건부 가능
S2	위조의 휘발성	inode 캐시 방출	제한적
S3	두 캐시 조건의 배타	물리 페이지 무효화	비성립
S4	페이지 캐시 회수	식별 시그니처 약함	제한적

표 10: 시나리오별 한계 귀속. 어느 시나리오도 네 한계를 우회하지 못하며, S3는 캐시 계층 구조상 원리적으로 성립하지 않는다.

7.2. 최선 사례 전개와 실패 지점

이 절은 S1을 가장 유리한 조건에서 단계별로 전개하여 각 단계가 어느 한계와 만나는지를 드러낸다. 환경은 DDR4-1333 다운클럭, ext4, 비ECC UDIMM, TME 비활성, 스크램블러 구조를 이미 알고 있으며 이번 부팅의 시드 복원도 끝난 상태로 가정한다. 표적은 setuid 바이너리로 만들 파일의 inode다.

단계 0, 표적 물성 확정. ext4에서 inode는 256바이트, extent 사용 시 오프셋 40에 매직이 있고 관심 필드는 권한과 소유자다. **한계:** metadata_csum이 켜져 있으면 여기에 i_generation과 체크섬 필드가 더해져 라인 0에서 2까지를 동시에 다루어야 한다(6.6. 부가 한계: 다중 캐시라인 정합 요구).

단계 1, 표적 유발과 지문. 공격자 프로세스가 chmod나 touch로 표적 inode를 더티화하고, 공유 매핑이 가능하면 clflush로, 아니면 eviction set을 접근해 방출을 유도한다. 후자의 경우 /proc/self/pagemap[28]으로 가상주소를 물리주소로 바꾸고, 알려진 슬라이스 해시로 같은 캐시 셋에 합동인 주소들을 모은 뒤 최소 크기로 축소해 접근한다[16]. 방출되면 캐시라인이 쓰기 버스트로 나타나고, 인터포저는 캡처하여 버스 반전을 되돌리고 디스크램블한 다음 오프셋 40의 매직을 확인하고 필드 정합으로 지문을 확정한다. **한계:** 이 단계 전체가 스크램블러·스왑·버스 반전을 모두 되돌릴 수 있다는 전제 위에 있고, 스크램블러 시드는 이 부팅 사이클에서 이미 복원되어 있어야 한다. TME가 켜져 있고 암호문 사전이 없다면 공격은 여기서 끝난다. 그리고 중요한 점은 이 방출 유도가 보장하는 것이 오직 LLC 미스뿐이라는 것이다. inode 캐시 방출은 시스템 전역 메모리 압력을 요구하며, 그 압력은 동시에 페이지 캐시 적중 조건을 깨뜨린다.

단계 2, 감시 등록과 프리로드. 확정된 라인들의 DRAM 좌표를 비교기의 상수 레지스터에 로드하고, 위조 페이로드와 다시 계산한 체크섬을 출력단 홀딩 레지스터에 프리로드한다. 방출에서 재참조까지의 간격이 마이크로초에서 밀리초 규모인 데 반해 프리로드는 수십에서 수백 나노초이므로 이 단계는 병목이 아니다. **한계:** 등록 시점과 재적재 시점 사이에 페이지 캐시가 회수되면 좌표는 조용히 무효화되며, 인터포저는 이를 알 방법이 없다.

단계 3, 재적재 유발과 위조 응답. 커널이 ext4_iget()으로 inode를 재적재하면 다음 순서가 전개된다. 행을 여는 명령을 관측해 행 일치를 확정하고, 읽기 명령 시점에 열 일치와 명령 조건이 클록으로 래치되어 약 0.3 ns 만에 매치 신호가 뜬다. 그 신호가 0.5 ns 안에 DRAM측 스위치를 열고 출력 인에이블을 개방하며, 임계경로 전체가 약 6.4 ns로 읽기 지연 13.5 ns보다 7.1 ns 일찍 끝난다. 읽기 지연이 지난 시점에 라인 0에서 2는 위조한 값으로, 라인 3은 캡처해 둔 원본으로 연속 방출한다. **한계:** 네 라인이 하나의 격리 구간 안에서 연속 처리되어야 하는데, 열이 끝난 뒤 스위치를 다시 닫는 데 필요한 시간이 버스트 간격 3.0 ns보다 길다(6.2.2. 새 제약: 버스트 사이의 전환 예산). 그리고 그 구간에 무관한 좌표의 읽기가 끼어들면 그 읽기는 손상된 데이터를 받는다. 부하가 높을수록 이 확률이 커진다.

단계 4, 커널의 신뢰와 상승. 커널은 위조된 권한 필드를 정상 inode로 수용하고, 이후 표적 바이너리 실행 시 setuid 비트를 신뢰한다. 체크섬이 맞으므로 커널은 오류를 남기지 않는다. **한계:** 위조는 이 순간 이후 DRAM과 분리되어 휘발되며, 커널이 해당 inode를 다시 더티화하면 진짜 값이 DRAM에 복원된다.

정합과 신뢰성 재평가. 위 전개에서 실패 요인은 5. 공격 설계: 콘텐츠 트리거형 읽기 위조 중간자 의 서술과 달리 타이밍이 아니다. 임계경로 여유와 오타율은 넉넉하지만, 실제 실패는 inode 캐시가 방출되지 않아 재적재 자체가 일어나지 않는 경우, 방출 유도 과정의 메모리 압력이 페이지 캐시를 함께 회수해 좌표가 무효화되는 경우, 격리 구간에 무관한 읽기가 끼어드는 경우에서 발생한다. 세 요인 모두 캐시 계층과 버스 부하에 기인하며 회로 설계로 개선되지 않는다. 즉 단일 표적 시나리오에서 실패 요인은 타이밍이 아니라 캐시 계층의 구조다.

7.3. 파일시스템 일반화와 그 한계

지문과 위조 원리는 다른 파일시스템으로 확장되지만, 확장이 곧 완화의 부재를 뜻하지는 않는다.

XFS는 슈퍼블록 매직 xfsb와 inode 매직 in이 강한 앵커이며[32], 권한 필드가 선두 근처에 있어 지문은 오히려 쉽다. 그러나 v5 XFS는 전 메타데이터에 crc32c를 강제하므로 6.6. 부가 한계: 다중 캐시라인 정합 요구의 다중 라인 요구가 ext4보다 강하고, inode가 512바이트 이상이라 모아야 할 캐시라인 수가 여덟 개로 늘어난다.

btrfs는 Copy-on-Write이므로 메타데이터가 트리를 재기록마다 다른 위치로 이동한다. 이는 6.3. 한계 II — 캐시 계층과 관측 가능성의 “같은 물리 페이지 유지” 조건을 구조적으로 위반하여 감시 좌표가 등록 직후 무

효화된다. 체크섬을 sha256이나 blake2로 설정하면 위조 페이로드의 재계산 비용도 커진다. 요컨대 CoW 파일시스템은 본 공격에 대해 의도치 않은 완화를 제공한다.

8. 방어와 탐지

8.1. 무결성과 신선도를 갖춘 메모리 암호화

6.5. **한계 IV — 메모리 암호화** 에서 보았듯 TME는 콘텐츠 지문 경로를 차단하지만 무결성과 신선도가 없어 다른 표적화 축의 재전송과 손상은 막지 못한다. TME-MK [23] 는 KeyID로 도메인별 키를 분리해 교차 도메인 열람을 어렵게 하나 도메인 내부에서는 여전히 결정론적이다. 읽기 위조 자체를 차단하려면 암호화가 무결성과 신선도를 함께 제공해야 한다. Intel SGX의 MEE [33] 는 카운터와 Merkle 트리로 셋을 모두 제공하나 보호 메모리가 작아 확장성이 제한되고, Intel TDX [23] 는 TME-MK에 MAC과 소유권 비트, 부팅 시 alias 검출을 결합하여 재전송·손상·정적 aliasing을 차단한다. 능동 읽기 위조가 이를 무력화하려면 데이터와 MAC을 모두 정합적으로 위조해야 하므로 난이도가 급증한다.

공격 단계 / 유형	무암호화	TME/ TME-MK	TDX/ MEE
콘텐츠 지문 표적 획득	가능	차단 [¶]	차단
읽기 응답 위조(S1~S4)	조건부 [‡]	가능 [†]	차단
맹목적 재전송/손상	가능	가능	차단
접근패턴 표적화	가능	가능	완화

표 11: 단계별 방어 여부. ‡는 암호화가 없어도 6. 한계 정량 분석의 캐시 계층·전환 예산·다중 라인 한계가 남음을 뜻한다. †는 표적을 다른 수단으로 특정한 경우 결정론성 때문에 위조와 재전송이 여전히 가능함을 뜻한다. ¶는 암호문 사전이 없는 공격자에 한한 차단이며, 사전을 가진 공격자는 암호문 자체를 지문으로 쓸 수 있다.

8.2. 탐지 가능성

본 연구의 한계 분석은 그대로 탐지 지표가 된다. 첫째, 능동 구동에 필요한 채널 다운클럭은 대역폭과 지연 저하로 나타나 성능 원격 측정으로 포착된다. 둘째, 전환 예산 부족으로 인접 버스트가 손상된다. ECC 모듈이라면 이 손상이 정정 불가 오류로 기록되지만, 본 논문이 가정한 비ECC 모듈에서는 로그가 남지 않고 원인 불명의 크래시로만 나타난다. 즉 비ECC 가정은 공격을 쉽게 만드는 동시에 탐지도 어렵게 만든다. 다만 실패율이 높은 구성일수록 크래시가 잦아지므로 크래시 빈도 자체는 여전히 지표가 된다. 셋째, inode 캐시를 비우기 위한 시스템 전역 메모리 압력은 수 분 규모로 지속되는 비정상 할당 패턴으로 드러난다. 넷째, 부팅과 런타임의 구성 무결성 검증, 즉 예상 데이터율과 타이밍 확인이 보조 지표가 된다.

8.3. 완화 방안

암호학적. 근본 대책은 결정론성을 없애는 것이다. 캐시라인마다 nonce나 카운터로 신선도를, MAC으로 무결성을 붙이면 지문과 재전송과 위조가 모두 막힌다. 대역폭·면적·지연 비용이 크므로 서버 규모 적용에는 설계 절충이 필요하다. 다만 본 연구의 결과는 콘텐츠 트리거 축에 한해서는 무결성 없이 기밀성만으로도 사전 없는 공격자를 막을 수 있음을 보여주며, 이는 TME를 켜는 것만으로 얻어지는 저비용 방어다. 반

대로 말하면, 결정론이 남아 있는 한 사전을 만들 수 있는 공격자에게는 TME가 방어가 되지 못한다.

아키텍처. 메모리를 패키지 안으로 통합하여 외부 버스 노출 자체를 줄이고, 부팅에 국한된 alias·구성 검증을 런타임까지 확장한다.

운영체제. 부수적이지만 효과가 확인된 완화가 있다. vfs_cache_pressure를 높여 inode 캐시를 더 빨리 회수하면

6.3. **한계 II — 캐시 계층과 관측 가능성** 의 회수 속도비가 커져 관측 가능성 상한이 4%대로 떨어진다. CoW 파일시스템 채택은 메타데이터 좌표의 안정성을 제거한다. 어느 쪽도 근본 대책은 아니나 공격 비용을 한 자릿수 이상 올린다.

시스템과 공급망. 위협의 출발점이 공급망 변조이므로 1차 방어선은 모듈 정품성 검증이다. SPD에 제조사 서명을 넣어 부팅 시 원격 검증하고, 모듈과 소켓에 물리 봉인과 탬퍼 증거를 적용한다. 민감 구조를 메모리에 쓰기 전 마스킹하는 컴파일러 기반 완화 [34] 도 병행 가치가 있다.

9. 논의

본 연구를 시작할 때의 가설은 “명령 버스를 직접 보는 하드와이어드 회로로 판정 지연을 게이트 수준까지 낮추면 읽기 위조가 성립한다”였다.

5. **공격 설계: 콘텐츠 트리거형 읽기 위조 중간자** 의 설계는 이 가설을 회로 수준에서 지지한다. 임계경로는 6.4 ns로 읽기 지연의 절반 이하이며, 자체 스트로브 구동은 DRAM 쪽 지터를 흡수하고, 부분 위조는 소자 규모를 실용 범위로 낮춘다. 그러나 6. 한계 정량 분석의 정량 분석은 이 가설이 잘못된 병목을 겨냥했음을 보인다.

실제 병목은 세 층에 있다. 첫째는 회로 안이 아니라 회로 사이에 있다. 개별 버스트의 타이밍이 아니라 버스트 사이의 전환 예산이 DDR4의 연속 접근 규칙과 충돌한다. 둘째는 하드웨어가 아니라 운영체제에 있다. 공격의 관측 창은 inode 캐시와 페이지 캐시와 LLC 세 계층이 특정 조합으로 정렬될 때만 열리고, 그 조합 중 둘은 서로를 배제한다. 셋째는 설계가 아니라 플랫폼에 있다. 버스의 바이트가 평문이라는 전제는 스캐램블러·스와즐·버스 반전을 되돌릴 수 있어야 성립하는데, 이 중 스캐램블러 시드는 부팅마다 바뀌어 매 부팅 런타임 복원을 요구하고, 메모리 암호화 앞에서는 사전 없이는 전제 자체가 무너진다.

이 구조는 하나의 교훈으로 요약된다. 하드웨어 인터포저 공격의 난이도는 인터포저가 아니라 그 위에 쌓인 캐시와 암호화 계층이 결정한다. 본 연구가 콘텐츠 트리거라는 축을 선택하여 물리주소 매핑 리버스 엔지니어링을 우회한 것은 실제 이점이지만, 그 대가로 공격이 버스 위 평문의 존재라는 단일 전제에 전부를 걸게 되었다. 이 전제는 방어 측이 스위치 하나로 제거할 수 있는 것이다. 게다가 6.6. **부가 한계: 다중 캐시라인 정합 요구** 에서 보았듯 metadata_csum이 기본으로 커진 환경에서는 여러 라인을 모아야 하므로, 우회했다고 믿었던 매핑 문제가 뒷문으로 다시 들어온다.

본 연구의 방법론적 한계도 명시한다. 모든 수치는 공개 규격과 데이터시트, 설계 가정에서 유도한 분석값이며 실측이 아니다. 특히 두 캐시의 회수 속도비, 랭크당 읽기율, 채널 신

호 무결성은 플랫폼 실험 없이는 확정할 수 없다. 따라서 표 8의 절대값보다는 어느 관문이 결과를 지배하는가라는 순위 정보에 의미를 두어야 한다. 범위의 한계도 하나 더 있다. 본 논문은 비ECC 모듈을 가정했는데, 정작 공격이 겨냥하는 데이터센터 서버는 ECC 모듈이 기본이다. ECC 모듈에서는 위조 데이터에 맞는 심볼도 함께 만들어야 하고 컨트롤러의 ECC 코드는 대개 비공개이므로, 스캠블러스와즐과 같은 성격의 “모르면 0”인 조건이 하나 더 붙는다. 즉 본 논문의 결과는 공격 가능성의 상한이며, ECC 환경으로의 확장은 남은 과제다. DDR5는 온다이 ECC와 사이드밴드 버스, 더 높은 데이터율로 구현 난도를 한층 높이지만, 결정론적 암호화라는 구조적 취약점 자체는 매체 세대와 무관하게 남는다.

10. 결론

본 논문은 DDR4 메모리 버스 중간자가 쓰기 콘텐츠에서 표적 좌표를 학습하고 이후 읽기 응답을 선택적으로 대체하는 공격 축을 정의하고, 그 실현 가능성을 실물 성공 주장 대신 한계 연구로 평가하였다. 설계를 콘텐츠 후보 검출, 좌표 재조립, 명령 매치, 데이터 재구동, 캐시 방출과 재적재의 다섯 단계로 나누고, 공격 한 회차의 성공을 다섯 관문의 곱으로 보아 네 개의 병목을 정량화하였다.

타이밍 측면에서 명령 여유는 대부분의 구성에서 양수였으나 데이터 창 여유가 지배 병목이었다. 하드 물리 계층 구성은 3200 MT/s에서 99.87%, 부분 위조와 PLL 조합은 2400에서 94.17%이나 3200에서 38.89%, 저전력 CMOS는 1333에서 33.13%의 통과율을 보였다. 여기에 더해 본 연구는 기존 분석이 다루지 않은 버스트 사이의 전환 예산 제약을 새로 식별하였다. tccq가 클럭 수로 규정되는 탓에 연속 읽기 사이의 데이터 버스 유휴 간격은 다른 뱅크 그룹에서 0이고 같은 뱅크 그룹에서도 800부터 3200까지 1.25 2.50 ns를 벗어나지 않는데, 이는 FET 스위칭 지연($t_{on} + t_{off}$, 4 10 ns)보다 짧다. 따라서 인접 버스트를 개별적으로 격리하는 운용은 어느 동작점에서도 성립하지 않으며, 다운클럭으로도 해소되지 않는다.

캐시 계층 측면에서는, 위조가 성립하려면 inode 캐시 미스와 페이지 캐시 적중과 LLC 미스가 동시에 만족되어야 하는데 앞의 두 조건이 같은 메모리 압력에 의해 반대 방향으로 움직여 서로를 배제함을 보였다. 이 반상관에서 유도한 관측 가능성의 이론적 상한은 최적 조건에서도 38.5%이며 현실적 값에서는 10%대다. 나아가 코드 페이지 위조는 페이지 캐시 회수가 필연적으로 물리 페이지를 바꾸므로 원리적으로 성립하지 않음을 확인하였다.

검출 측면에서는 16비트 매직 비교가 기준 쓰기율에서 초당 15.26개의 후보를 만들지만 필드 정합으로 초당 0.305개까지 낮아져, 32 64개 슬롯으로 처리 가능한 설계 공간이 형성됨을 보였다. 그러나 진짜 한계는 오탐이 아니라 미탐이며, 스캠블러·DQ 스와즐·버스 반전 중 하나라도 되돌리지 못하면 검출 확률이 0이 된다. 특히 스캠블러 시드는 부팅할 때마다 새로 정해지므로 장치는 전원이 들어올 때마다 시드를 다시 복원해야 하는 런타임 부담을 진다. 다만 이 복원은 관측만으로 가능하여 공격을 막는 요인이자기보다 부팅 직후의 공백을 만드는 요인이다. 또한 `metadata_csum`이 기본 활성화인 현대 ext4에서

는 체크섬 필드가 별도 캐시라인에 놓여 단일 라인 위조가 불가능하고, 최소 두세 개 좌표를 같은 회차에 위조해야 한다.

가장 강한 한계는 메모리 암호화다. Intel TME가 켜지면 버스에는 암호문만 흐르고 AES-XTS는 선형이 아니어서 좌표로부터 되돌릴 수 없으므로, 주소별 암호문 사전이 없는 공격자에 대해서는 다른 모든 관문과 무관하게 종단 성공확률이 정확히 0이 된다. 다만 XTS가 결정론적이라는 성질 때문에, 표적 주소에 알려진 평문을 미리 써 넣어 사전을 만들 수 있는 공격자에게는 암호문 자체가 지문이 되어 경로가 되살아난다. 사전은 좌표마다 따로 필요하고 재부팅으로 무효가 되므로, 이 조건은 공급망 위협 모델이 전제하는 “모르는 시스템”과 양립하지 않는다.

종합하면 한 회차 성공확률의 상한은 실험실에 유리한 낙관 구성에서 0.319(95% 도달 8회), 기본 설정에 가까운 기준 구성에서 0.092(31회), 정격과 고부하의 보수 구성에서 9.3×10^{-5} (3만 회 이상)이며, 메모리 암호화가 켜지고 사전이 없으면 0이다. 이 값들을 상한이라 부르는 이유는 캐시 관문에 이론적 최댓값을 그대로 넣었고 식별과 검증 두 관문에는 유도되지 않은 가정값을 배정했기 때문이며, 실제 값은 이보다 작다. 즉 본 공격은 통제된 저속·비암호화 환경에서만 실용적이며, 공급망 공격이 상정하는 임의의 현대 서버에서는 성립하지 않는다.

방어 관점에서 이 결과는 실용적 함의를 갖는다. 무결성과 신선도가 없어 데이터 버스 능동 변조 일반에는 무력하다고 평가되어 온 TME가, 콘텐츠로 표적을 잡는 공격에 대해서는 사전 없는 공격자를 완전히 막는다. 따라서 단기적으로는 메모리 암호화의 기본 활성화가, 근본적으로는 무결성과 신선도를 결합한 암호화와 공급망 단계의 모듈 정품성·물리 무결성 검증이 대책이 된다. 후속 연구는 수동 채널 검증에서 캡처 전용 인터포저, 저속 단일 레인 재구동, 다중 레인과 정격 확장의 순서로 진행하되, 각 단계에서 본 논문이 식별한 네 한계를 탈락 기준으로 삼는 것이 타당하다.

참고문헌

- [1] Linux Kernel Community, “Memory Management Documentation”. [Online]. Available at: <https://www.kernel.org/doc/html/v6.12/mm/index.html>
- [2] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai, 와/과 R. A. Popa, “An Off-Chip Attack on Hardware Enclaves via the Memory Bus”, in 29th USENIX Security Symposium, 2020, pp 487–504. [Online]. Available at: <https://www.usenix.org/conference/usenixsecurity20/presentation/lee-dayeol>
- [3] Linux Kernel Community, “Linux page cache writeback implementation”. [Online]. Available at: <https://github.com/torvalds/linux/blob/v6.12/mm/page-writeback.c>
- [4] A. Seto 기타, “WireTap: Breaking Server SGX via DRAM Bus Interposition”, in ACM SIGSAC Conference on Computer and Communications

- Security, 2025. [Online]. Available at: <https://wiretap.fail/>
- [5] J. D. Meulemeester, L. Wilke, D. Oswald, T. Eisenbarth, I. Verbauwhede, and J. V. Bulck, "BadRAM: Practical Memory Aliasing Attacks on Trusted Execution Environments", in IEEE Symposium on Security and Privacy, 2025. [Online]. Available at: <https://badram.eu/>
 - [6] J. D. Meulemeester, D. Oswald, I. Verbauwhede, and J. V. Bulck, "Battering RAM: Low-Cost Interposer Attacks on Confidential Computing via Dynamic Memory Aliasing", in IEEE Symposium on Security and Privacy, 2026. [Online]. Available at: <https://batteringram.eu/>
 - [7] P. Pessl and others, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks", in USENIX Security Symposium, 2016. [Online]. Available at: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl>
 - [8] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "DRAMDig: A Knowledge-Assisted Tool to Uncover DRAM Address Mapping", in Design Automation Conference, 2020. [Online]. Available at: <https://arxiv.org/abs/2004.02354>
 - [9] M. Wi, S. Baek, S. Park, M. Erez, and J. H. Ahn, "Sudoku: Decomposing DRAM Address Mapping into Component Functions". [Online]. Available at: <https://arxiv.org/abs/2506.15918>
 - [10] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks", in IEEE Symposium on Security and Privacy, 2019. [Online]. Available at: https://download.vusec.net/papers/eccploit_sp19.pdf
 - [11] J. Corbet, "Clarifying memory management with page folios". [Online]. Available at: <https://lwn.net/Articles/849538/>
 - [12] Linux Kernel Community, "The ext4 Filesystem: On-Disk Data Structures". [Online]. Available at: <https://www.kernel.org/doc/html/v6.12/filesystems/ext4/index.html>
 - [13] J. A. Halderman and others, "Lest We Remember: Cold Boot Attacks on Encryption Keys", in 17th USENIX Security Symposium, 2008. [Online]. Available at: https://www.usenix.org/legacy/events/sec08/tech/full_papers/halderman/halderman.pdf
 - [14] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-Level Cache Side-Channel Attacks are Practical", in IEEE Symposium on Security and Privacy, 2015. [Online]. Available at: <https://yuval.yarom.org/pdfs/LiuYGH15.pdf>
 - [15] C. Maurice, N. L. Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters", in RAID, 2015. doi: 10.1007/978-3-319-26362-5_3.
 - [16] P. Vila, B. Köpf, and J. F. Morales, "Theory and Practice of Finding Eviction Sets", in IEEE Symposium on Security and Privacy, 2019. doi: 10.1109/SP.2019.00042.
 - [17] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: Cross-Cores Cache Covert Channel", in DIMVA, 2015. doi: 10.1007/978-3-319-20550-2_3.
 - [18] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory Access Scheduling", in ISCA, 2000. [Online]. Available at: https://groups.csail.mit.edu/cag/PSRG/papers/rixner_isca27.pdf
 - [19] Linux Kernel Community, "Linux ext4 inode implementation source". [Online]. Available at: <https://github.com/torvalds/linux/blob/v6.12/fs/ext4/inode.c>
 - [20] Linux Kernel Community, "ext4 Inodes". [Online]. Available at: <https://www.kernel.org/doc/html/v6.12/filesystems/ext4/inodes.html>
 - [21] Linux Kernel Community, "ext4 Metadata Checksums". [Online]. Available at: <https://www.kernel.org/doc/html/v6.12/filesystems/ext4/checksums.html>
 - [22] Intel Corporation, "Intel Total Memory Encryption White Paper". [Online]. Available at: <https://www.intel.com/content/dam/www/central-libraries/us/en/documents/white-paper-intel-tme.pdf>
 - [23] P.-C. Cheng and others, "Intel TDX Demystified: A Top-Down Approach". [Online]. Available at: <https://arxiv.org/abs/2303.15540>
 - [24] S. F. Yitbarek, M. T. Aga, R. Das, and T. Austin, "Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors", in HPCA, 2017. doi: 10.1109/HPCA.2017.10.
 - [25] Siemens EDA, "Using Data Bus Inversion to Mitigate Simultaneously Switching Noise". [Online]. Available at: <https://blogs.sw.siemens.com/electronic-systems-design/2015/06/24/using-data-bus-inversion-to-mitigate-simultaneously-switching-noise/>

- [26] Micron Technology, “DDR4 SDRAM Datasheet”. [Online]. Available at: <https://www.micron.com/products/memory/dram-components/ddr4-sdram>
- [27] G. Henke, “Attack on DDR4 Memory Scrambling”, Master's thesis, 2020. [Online]. Available at: https://www.its.uni-luebeck.de/fileadmin/files/theses/MA_GillianHenke_MemoryScrambling.pdf
- [28] Linux Kernel Community, “Examining Process Page Tables: pagemap”. [Online]. Available at: <https://www.kernel.org/doc/html/v6.12/admin-guide/mm/pagemap.html>
- [29] JEDEC Solid State Technology Association, “JESD79-4B: DDR4 SDRAM Standard”. [Online]. Available at: <https://www.jedec.org/standards-documents/docs/jesd79-4a>
- [30] Advanced Micro Devices, “AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More”. [Online]. Available at: <https://docs.amd.com/v/u/en-US/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more>
- [31] M. Li, Y. Zhang, H. Wang, K. Li, 와/과 Y. Cheng, “CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel”, in 30th USENIX Security Symposium, 2021. [Online]. Available at: <https://www.usenix.org/conference/usenixsecurity21/presentation/li-mengyuan>
- [32] Linux Kernel Community, “XFS Self Describing Metadata”. [Online]. Available at: <https://www.kernel.org/doc/html/v6.12/filesystems/xfs/xfs-self-describing-metadata.html>
- [33] S. Gueron, “A Memory Encryption Engine Suitable for General Purpose Processors”. [Online]. Available at: <https://eprint.iacr.org/2016/204>
- [34] M. Li, L. Wilke, J. Wichelmann, T. Eisenbarth, R. Teodorescu, 와/과 Y. Zhang, “A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP”, in IEEE Symposium on Security and Privacy, 2022. doi: 10.1109/SP46214.2022.9833768.