원문 : https://digital-forensics.sans.org/blog/2011/03/14/digital-forensics-understanding-ext4-part-2-timestamps

Setting tue Stage

Ext4 파일 시스템에서 다른 테스트 파일을 설정해보자.

|  |
| --- |
| # echo Time for knowledge >testfile  # touch -a -t 211101231917.42 testfile  # touch -m -t 204005160308.19 testfile |

atime과 mtime 값을 명시적으로 보기 위해 root 권한에서 touch 명령어를 사용할 것이다. 이는 몇몇 흥미로운 타임스탬프 값을 확인하기 위해서이다. 그렇지 않으면 모든 타임스탬프가 새 파일을 만든 순간으로 설정될 것이다. 여기서 한가지 주의해야 할 점은 atime과 mtime 둘 다 먼 미래의 날짜로 설정하고 있다는 것인데, 오래된 Unix 파일 시스템에서는 일반적으로 32비트 date rollover 이슈(또는 "2038년 문제"로 불리는 것)로 인해 문제가 발생할 것이다.

자, 이제 표준 linux stat 명령의 출력값과 Sleuthkit의 istat의 출력값, debugfs의 stat 버전을 비교해보자:

|  |
| --- |
| # stat testfile   File: `testfile'   Size: 19            Blocks: 8          IO Block: 4096   regular file  Device: fc03h/64515d    Inode: 6554914     Links: 1  Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)  Access: 2111-01-23 19:17:42.000000000 -0800  Modify: 2040-05-16 03:08:19.000000000 -0700  Change: 2011-03-12 07:36:13.872411014 -0800  # istat /dev/mapper/RD-home 6554914  inode: 6554914  Allocated  Group: 800  [ . . . ]  Inode Times:  Accessed:          Tue Dec 17 12:49:26 1974  File Modified:     Wed May 16 03:08:19 2040  Inode Modified:   Sat Mar 12 07:36:13 2011  [ . . . ]  # debugfs -R 'stat <6554914>' /dev/mapper/RD-home  [ . . . ]   ctime: 0x4d7b92ed:cfffbe18 -- Sat Mar 12 07:36:13 2011   atime: 0x0954b156:00000001 -- Tue Dec 17 12:49:26 1974   mtime: 0x845e5913:00000000 -- Wed May 16 03:08:19 2040  crtime: 0x4d7b92e4:148af06c -- Sat Mar 12 07:36:04 2011  [ . . . ] |

타임스탬프에 더 쉽게 집중할 수 있게 출력값을 편집했다. 보다시피, 세가지 명령들은 기본적으로 mtime과 ctime 값에 동의한다. 그러나 stat 명령어는 2111의 atime 값을 정확히 해석하지만, istat과 debugfs stat는 모두 그것을 해석하는 데 문제가 있다.

너는 아마 Ext4의 다른 새로운 기능도 발견했을 수 있다:

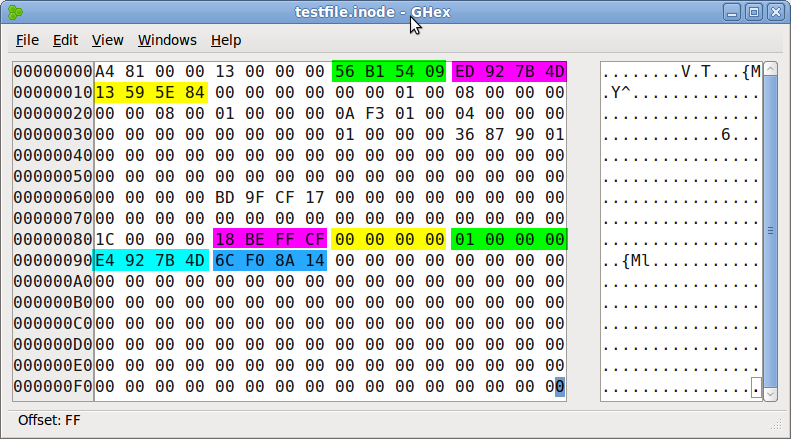
* Stat와 debugfs stat 명령어는 소수 타임스탬프 값을 지원한다. 이제 Ext4는 파일 타임스탬프에서 ns「nanosecond」 해상도를 지원한다.
* Debugfs stat 출력값에 "crtime" (create time) 타임스탬프가 있다. Ext4는 마침내 NTFS같은 "born on" 시간 (btime)을 지원한다. 디스크 볼륨 내에 파일이 생성되었을 때를 위한 타임스탬프를 지원하기 위해서이다.
* 그리고 stat과 touch 둘 다 적어도 먼 미래의 날짜는 완벽하게 받아들일 수 있을 것으로 보인다. Ext4가 2038년 문제를 고친 것 같다.

Eet4는 64비트 타임스탬프 필드로 이동함으로써 이 기능들의 많은 부분을 제공했다. 이는 ns 타임스탬프를 지원하기에 충분한 해상도를 제공하며, 여전히 몇 개의 비트가 남아 있어 미래까지 날짜를 지정할 수 있다.

Show Me the Bits

이전 편에서 언급했듯이 Ext4 개발자들은 Ext2/Ext3의 inode 레이아웃과의 역호환성을 유지하기 위해 매우 열심히 노력했다. 그러나 64비트 타임스탬프와 완전히 새로운 파일 생성 타임스탬프는 분명히 이 목표를 어렵게 한다. Ext4 개발자들은 이 문제를 새로운 256비트 Ext4 inode의 상위 128비트에 추가하여 해결했다.

Hex editor 보기 ( 정확한 inode의 파일 카빙 프로세스는 Part 1을 참조 ):



inode의 시작을 0부터 계산하면, 다음은 다양한 강조 표시된 필드에 대한 바이트 오프셋과 의미이다:

|  |
| --- |
| Bytes 8 - 11: Access time seconds 12 - 15: Change time seconds 16 - 19: Modification time seconds  132 - 135: Change time "extra" 136 - 139: Modification time "extra" 140 - 143: Access time "extra" 144 - 147: Create time seconds 148 - 151: Create time "extra" |

Inode 시작 근처의 표준 MAC 시간 값은 기본적으로 변경되지 않는다. 1970년 1월 1일(unix "epoch"라는 이름) 이후 초를 저장한다. 그러나 Ext4는 이러한 값을 요청된 원래 Unix 파일 시스템 규격으로 서명된 값이 아니라 서명되지 않은 정수로 처리한다. 여분의 비트를 사용하면 실제로 "2038년 문제"를 "2106년 문제"로 바꿀 수 있다.

실제로 2040년도의 mtime 값으로 이것을 볼 수 있다. inode가 사용하는 리틀 엔디안 형식에서 변환하여, 16진수 mtime 값은 0x845E5913 또는 십진수 2220775699-32비트 서명 값에서 rollover를 유발하는-이다. (※ You can actually see this in action with our mtime value from the year 2040. Converted from the little-endian format that the inode uses, our hex mtime value is 0x845E5913, or 2220775699 decimal- a number that would normally cause roll-over in a 32-bit signed value. 제대로 된 해석이 아닌 것 같아 원문 첨부) 이 십진수 값을 작은 쉘 magic과 GNU date 명령으로 사람이 읽을 수 있는 날짜로 변환할 수 있다:

|  |
| --- |
| # **date -d @2220775699** Wed May 16 03:08:19 PDT 2040 |

아까의 stat 결과값과 비교해보면 정확한 값을 계산한 것을 알 수 있다.

But Wait! There's More!

그러나, inode의 상위 128바이트에는 각 MAC 타임스탬프에 대해 "extra" 32비트가 있다. 여기에 해당 색상으로 표시했기 때문에 매칭하기 쉽다. "extra" 필드를 각 타임스탬프의 ns부분으로 생각할 수 있다. 하지만 실제로는 ns 해상도를 나타내기 위해 30비트만 있으면 된다. 그래서 "extra" 장면의 상위 30비트는 ns이지만, 나머지 2비트는 실제로 inode의 초기 부분의 seconds field를 확장하는 데 사용된다.

헷갈린다면 2111년의 atime 값을 예로 들어보자. 표준 atime bytes 8-11을 살펴보면 16진수 값이 0x0954B156 또는 156545366이 될 것이다. 이를 날짜 문자열로 변환하면:

|  |
| --- |
| # **date -d @156545366** Tue Dec 17 12:49:26 PST 1974 |

이제 istat과 debugfs stat이 잘못 출력하고 있는 곳을 볼 수 있다. 그들은 inode 초기 때부터 표준 32비트 atime 값을 변환하고 있을 뿐이다.

하지만 우리가 "extra" atime 필드에서 두 비트를 낮은 순서로 더하면 어떻게 되는지 봐라. "extra" atime 필드의 low-order 바이트는 2진수로 0x01 또는 "00000001"이다. 그래서 low-order 2비트는 "01"이다. 우리는 표준 atime 값 앞에 이 추가 비트를 미리 붙인다. 그래서 실제 atime는 0x010954B156으로, 10진수 4451512662이다. 이제 우리는 그것을 인간이 읽을 수 있는 스트링으로 변환한다:

|  |
| --- |
| # **date -d @4451512662** Fri Jan 23 19:17:42 PST 2111 |

빙고! 우리의 touch 명령과 stat 출력값에 매칭되는 정확한 날짜가 있다.

extra 2비트로 나타낼 수 있는 가장 큰 값은 0x03FFFFFFFF, 즉 십진수 17179869이다. 이로써 GMT의 날짜는 2514-05-30 01:53:03이며 이는 확실히 우리의 날짜 rollover 문제를 훨씬 더 멀리 미뤄주고 있다. 하지만 우리가 실제로 ns 파일 타임스탬프 해상도가 필요한지는 확실하지 않다. 나는 Ext4 개발자들이 덜 정밀하게 선택해서 우리의 타임스탬프의 수명을 늘리기 위해 나에게 2-6비트를 더 줬으면 좋겠다. 반면에, 2514년까지 이 중 어느 것도 아마 중요하지 않을 것이다. (※ I wish the EXT4 developers had chosen less precision and given me another 2-6 bits to extend the lifetime of our timestamps. On the other hand, by 2514 none of this will probably matter anyway.)

Fractional Seconds

각 타임스탬프의 "extra" 비트의 재미있는 packing은 여러분이 각 타임스탬프의 ns 구성요소에 관심이 있을 때 상황을 더 복잡하게 만든다. 새 생성 시간 필드를 예로 들어보자. 생성 시간 초와 생성 시간 "extra" 필드에 필요한 64비트는 새로운 대형 Ext4 inode의 상단 부분에 보관된다. 그것들은 다른 타임스탬프의 "extra" 필드 바로 뒤에 나타나며 위의 그림에서 두 가지 색조의 파란색으로 강조된다.

생성 시간 "extra"의 16진수 값은 0x148이다. AF06C, 또는 344649836. 그러나 low-order 2비트는 ns를 세는 데 사용되지 않는다. 우리는 그 비트들을 버리고 모든 것을 2비트씩 바로 옮겨야 한다. 이것은 4로 나누는 것과 같다. 따라서 우리의 실제 ns 값은 344649836 / 4 = 86162459이다.

우리는 변경 시간 "extra" 필드에 대해서도 같은 일을 할 수 있다. 16진수 값 0xCFFFBE18는 십진수로 3489644056이다. 4로 나누면 872411014가 나온다. 글의 앞부분에 나온 stat 결과값을 다시 참조하면, 계산된 값이 ctime에 대해 보고된 fractional seconds와 일치한다는 것을 알 수 있다.

Wrapping Up For Now

이 시점에서 우리는 대부분의 사용자에게 영향을 미치는 주요 Ext4 inode 변경사항, 즉 extent와 새로운 타임스탬프 형식을 다루었다. 그러나 제 1부의 extent에 대한 논의는 표준 Ext3 inode에서 사용 가능한 4개 이상의 extent 구조가 실제로 필요할 때 발생하는 문제를 건너뛰었다. 그것은 이 시리즈의 다음 편에서 주제가 될 것이다.