```
**********************************************************
    8432 Mon Jun 23 22:27:38 2014
new/usr/src/grub/grub-0.97/stage2/zfs_lz4.c
4936 lz4 could theoretically overflow a pointer with a certain input
**********************************************************
______unchanged_portion_omitted_

#define A64(x)  (((U64_S *)(x))->v)
#define A32(x)  (((U32_S *)(x))->v)
#define A16(x)  (((U16_S *)(x))->v)

/*
 * Constants
 */
#define MINMATCH 4

#define COPYLENGTH 8
#define LASTLITERALS 5

#define ML_BITS 4
#define ML_MASK ((1U<<ML_BITS)-1)
#define RUN_BITS (8-ML_BITS)
#define RUN_MASK ((1U<<RUN_BITS)-1)

/*
 * Architecture-specific macros
 */
#if LZ4_ARCH64
#define STEPSIZE 8
#define UARCH U64
#define AARCH A64
#define LZ4_COPYSTEP(s, d)      A64(d) = A64(s); d += 8; s += 8;
#define LZ4_COPYPACKET(s, d)    LZ4_COPYSTEP(s, d)
#define LZ4_SECURECOPY(s, d, e) if (d < e) LZ4_WILDCOPY(s, d, e)
#define HTYPE U32
#define INITBASE(base)          const BYTE* const base = ip
#else
#define STEPSIZE 4
#define UARCH U32
#define AARCH A32
#define LZ4_COPYSTEP(s, d)      A32(d) = A32(s); d += 4; s += 4;
#define LZ4_COPYPACKET(s, d)    LZ4_COPYSTEP(s, d); LZ4_COPYSTEP(s, d);
#define LZ4_SECURECOPY          LZ4_WILDCOPY
#define HTYPE const BYTE*
#define INITBASE(base)          const int base = 0
#endif

#if (defined(LZ4_BIG_ENDIAN) && !defined(BIG_ENDIAN_NATIVE_BUT_INCOMPATIBLE))
#define LZ4_READ_LITTLEENDIAN_16(d, s, p) \
        { U16 v = A16(p); v = lz4_bswap16(v); d = (s) - v; }
#define LZ4_WRITE_LITTLEENDIAN_16(p, i) \
        { U16 v = (U16)(i); v = lz4_bswap16(v); A16(p) = v; p += 2; }
#else
#define LZ4_READ_LITTLEENDIAN_16(d, s, p) { d = (s) - A16(p); }
#define LZ4_WRITE_LITTLEENDIAN_16(p, v)  { A16(p) = v; p += 2; }
#endif

/* Macros */
#define LZ4_WILDCOPY(s, d, e) do { LZ4_COPYPACKET(s, d) } while (d < e);

/* Decompression functions */

static int
LZ4_uncompress_unknownOutputSize(const char *source,
    char *dest, int isize, int maxOutputSize)
{
```



```
        /* Local Variables */
        const BYTE *restrict ip = (const BYTE *) source;
        const BYTE *const iend = ip + isize;
        const BYTE *restrict ref;

        BYTE *restrict op = (BYTE *) dest;
        BYTE *const oend = op + maxOutputSize;
        BYTE *cpy;

        size_t dec[] = { 0, 3, 2, 3, 0, 0, 0, 0 };

        /* Main Loop */
        while (ip < iend) {
                BYTE token;
                int length;

                /* get runlength */
                token = *ip++;
                if ((length = (token >> ML_BITS)) == RUN_MASK) {
                        int s = 255;
                        while ((ip < iend) && (s == 255)) {
                                s = *ip++;
                                length += s;
                        }
                }
                /* copy literals */
                cpy = op + length;
                /* CORNER-CASE: cpy might overflow. */
                if (cpy < op)
                        goto _output_error;     /* cpy was overflowed, bail! */
                if ((cpy > oend - COPYLENGTH) ||
                    (ip + length > iend - COPYLENGTH)) {
                        if (cpy > oend)
                                /*
                                 * Error: request to write beyond destination
                                 * buffer.
                                 */
                                goto _output_error;
                        if (ip + length > iend)
                                /*
                                 * Error : request to read beyond source
                                 * buffer.
                                 */
                                goto _output_error;
                        memcpy(op, ip, length);
                        op += length;
                        ip += length;
                        if (ip < iend)
                                /* Error : LZ4 format violation */
                                goto _output_error;
                        /* Necessarily EOF, due to parsing restrictions. */
                        break;
                }
                LZ4_WILDCOPY(ip, op, cpy);
                ip -= (op - cpy);
                op = cpy;

                /* get offset */
                LZ4_READ_LITTLEENDIAN_16(ref, cpy, ip);
                ip += 2;
                if (ref < (BYTE * const) dest)
                        /*
                         * Error: offset creates reference outside of
                         * destination buffer.
                         */
                        goto _output_error;
```

```
257                 /* get matchlength */
258                 if ((length = (token & ML_MASK)) == ML_MASK) {
259                         while (ip < iend) {
260                                 int s = *ip++;
261                                 length += s;
262                                 if (s == 255)
263                                         continue;
264                                 break;
265                         }
266                 }
267                 /* copy repeated sequence */
268                 if unlikely(op - ref < STEPSIZE) {
269 #if LZ4_ARCH64
270                         size_t dec2table[] = { 0, 0, 0, -1, 0, 1, 2, 3 };
271                         size_t dec2 = dec2table[op - ref];
272 #else
273                         const int dec2 = 0;
274 #endif
275                         *op++ = *ref++;
276                         *op++ = *ref++;
277                         *op++ = *ref++;
278                         *op++ = *ref++;
279                         ref -= dec[op - ref];
280                         A32(op) = A32(ref);
281                         op += STEPSIZE - 4;
282                         ref -= dec2;
283                 } else {
284                         LZ4_COPYSTEP(ref, op);
285                 }
286                 cpy = op + length - (STEPSIZE - 4);
287                 if (cpy > oend - COPYLENGTH) {
288                         if (cpy > oend)
289                                 /*
290                                  * Error: request to write outside of
291                                  * destination buffer.
292                                  */
293                                 goto _output_error;
294                         LZ4_SECURECOPY(ref, op, (oend - COPYLENGTH));
295                         while (op < cpy)
296                                 *op++ = *ref++;
297                         op = cpy;
298                         if (op == oend)
299                                 /*
300                                  * Check EOF (should never happen, since last
301                                  * 5 bytes are supposed to be literals).
302                                  */
303                                 break;
304                         continue;
305                 }
306                 LZ4_SECURECOPY(ref, op, cpy);
307                 op = cpy;       /* correction */
308         }

310         /* end of decoding */
311         return (int)(((char *)op) - dest);

313         /* write overflow error detected */
314         _output_error:
315         return (int)(-(((char *)ip) - source));
316 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   30163 Mon Jun 23 22:27:39 2014
new/usr/src/uts/common/fs/zfs/lz4.c
4936 lz4 could theoretically overflow a pointer with a certain input
**********************************************************
______unchanged_portion_omitted_

 920 /* Decompression functions */

 922 /*
 923  * Note: The decoding functions real_LZ4_uncompress() and
 924  *      LZ4_uncompress_unknownOutputSize() are safe against "buffer overflow"
 925  *      attack type. They will never write nor read outside of the provided
 926  *      output buffers. LZ4_uncompress_unknownOutputSize() also insures that
 927  *      it will never read outside of the input buffer. A corrupted input
 928  *      will produce an error result, a negative int, indicating the position
 929  *      of the error within input stream.
 930  */

 932 static int
 933 real_LZ4_uncompress(const char *source, char *dest, int osize)
 934 {
 935         /* Local Variables */
 936         const BYTE *restrict ip = (const BYTE *) source;
 937         const BYTE *ref;

 939         BYTE *op = (BYTE *) dest;
 940         BYTE *const oend = op + osize;
 941         BYTE *cpy;

 943         unsigned token;

 945         size_t length;
 946         size_t dec32table[] = {0, 3, 2, 3, 0, 0, 0, 0};
 947 #if LZ4_ARCH64
 948         size_t dec64table[] = {0, 0, 0, (size_t)-1, 0, 1, 2, 3};
 949 #endif

 951         /* Main Loop */
 952         for (;;) {
 953                 /* get runlength */
 954                 token = *ip++;
 955                 if ((length = (token >> ML_BITS)) == RUN_MASK) {
 956                         size_t len;
 957                         for (; (len = *ip++) == 255; length += 255) {
 958                         }
 959                         length += len;
 960                 }
 961                 /* copy literals */
 962                 cpy = op + length;
 963                 /* CORNER-CASE: cpy might overflow. */
 964                 if (cpy < op)
 965                         goto _output_error;     /* cpy was overflowed, bail! */
 966                 if unlikely(cpy > oend - COPYLENGTH) {
 967                         if (cpy != oend)
 968                                 /* Error: we must necessarily stand at EOF */
 969                                 goto _output_error;
 970                         (void) memcpy(op, ip, length);
 971                         ip += length;
 972                         break;  /* EOF */
 973                         }
 974                 LZ4_WILDCOPY(ip, op, cpy);
 975                 ip -= (op - cpy);
 976                 op = cpy;

 978                 /* get offset */
```



```
 979                 LZ4_READ_LITTLEENDIAN_16(ref, cpy, ip);
 980                 ip += 2;
 981                 if unlikely(ref < (BYTE * const) dest)
 982                         /*
 983                          * Error: offset create reference outside destination
 984                          * buffer
 985                          */
 986                         goto _output_error;

 988                 /* get matchlength */
 989                 if ((length = (token & ML_MASK)) == ML_MASK) {
 990                         for (; *ip == 255; length += 255) {
 991                                 ip++;
 992                         }
 993                         length += *ip++;
 994                 }
 995                 /* copy repeated sequence */
 996                 if unlikely(op - ref < STEPSIZE) {
 997 #if LZ4_ARCH64
 998                         size_t dec64 = dec64table[op-ref];
 999 #else
1000                         const int dec64 = 0;
1001 #endif
1002                         op[0] = ref[0];
1003                         op[1] = ref[1];
1004                         op[2] = ref[2];
1005                         op[3] = ref[3];
1006                         op += 4;
1007                         ref += 4;
1008                         ref -= dec32table[op-ref];
1009                         A32(op) = A32(ref);
1010                         op += STEPSIZE - 4;
1011                         ref -= dec64;
1012                 } else {
1013                         LZ4_COPYSTEP(ref, op);
1014                 }
1015                 cpy = op + length - (STEPSIZE - 4);
1016                 if (cpy > oend - COPYLENGTH) {
1017                         if (cpy > oend)
1018                                 /*
1019                                  * Error: request to write beyond destination
1020                                  * buffer
1021                                  */
1022                                 goto _output_error;
1023                         LZ4_SECURECOPY(ref, op, (oend - COPYLENGTH));
1024                         while (op < cpy)
1025                                 *op++ = *ref++;
1026                         op = cpy;
1027                         if (op == oend)
1028                                 /*
1029                                  * Check EOF (should never happen, since last
1030                                  * 5 bytes are supposed to be literals)
1031                                  */
1032                                 goto _output_error;
1033                         continue;
1034                 }
1035                 LZ4_SECURECOPY(ref, op, cpy);
1036                 op = cpy;       /* correction */
1037         }

1039         /* end of decoding */
1040         return (int)(((char *)ip) - source);

1042         /* write overflow error detected */
1043         _output_error:
1044         return (int)(-(((char *)ip) - source));
```

```
1045 }

1047 static int
1048 LZ4_uncompress_unknownOutputSize(const char *source, char *dest, int isize,
1049     int maxOutputSize)
1050 {
1051         /* Local Variables */
1052         const BYTE *restrict ip = (const BYTE *) source;
1053         const BYTE *const iend = ip + isize;
1054         const BYTE *ref;

1056         BYTE *op = (BYTE *) dest;
1057         BYTE *const oend = op + maxOutputSize;
1058         BYTE *cpy;

1060         size_t dec32table[] = {0, 3, 2, 3, 0, 0, 0, 0};
1061 #if LZ4_ARCH64
1062         size_t dec64table[] = {0, 0, 0, (size_t)-1, 0, 1, 2, 3};
1063 #endif

1065         /* Main Loop */
1066         while (ip < iend) {
1067                 unsigned token;
1068                 size_t length;

1070                 /* get runlength */
1071                 token = *ip++;
1072                 if ((length = (token >> ML_BITS)) == RUN_MASK) {
1073                         int s = 255;
1074                         while ((ip < iend) && (s == 255)) {
1075                                 s = *ip++;
1076                                 length += s;
1077                         }
1078                 }
1079                 /* copy literals */
1080                 cpy = op + length;
1081                 /* CORNER-CASE: cpy might overflow. */
1082                 if (cpy < op)
1083                         goto _output_error;     /* cpy was overflowed, bail! */
1084                 if ((cpy > oend - COPYLENGTH) ||
1085                     (ip + length > iend - COPYLENGTH)) {
1086                         if (cpy > oend)
1087                                 /* Error: writes beyond output buffer */
1088                                 goto _output_error;
1089                         if (ip + length != iend)
1090                                 /*
1091                                  * Error: LZ4 format requires to consume all
1092                                  * input at this stage
1093                                  */
1094                                 goto _output_error;
1095                         (void) memcpy(op, ip, length);
1096                         op += length;
1097                         /* Necessarily EOF, due to parsing restrictions */
1098                         break;
1099                 }
1100                 LZ4_WILDCOPY(ip, op, cpy);
1101                 ip -= (op - cpy);
1102                 op = cpy;

1104                 /* get offset */
1105                 LZ4_READ_LITTLEENDIAN_16(ref, cpy, ip);
1106                 ip += 2;
1107                 if (ref < (BYTE * const) dest)
1108                         /*
1109                          * Error: offset creates reference outside of
1110                          * destination buffer
```



```
1111                          */
1112                         goto _output_error;

1114                 /* get matchlength */
1115                 if ((length = (token & ML_MASK)) == ML_MASK) {
1116                         while (ip < iend) {
1117                                 int s = *ip++;
1118                                 length += s;
1119                                 if (s == 255)
1120                                         continue;
1121                                 break;
1122                         }
1123                 }
1124                 /* copy repeated sequence */
1125                 if unlikely(op - ref < STEPSIZE) {
1126 #if LZ4_ARCH64
1127                         size_t dec64 = dec64table[op-ref];
1128 #else
1129                         const int dec64 = 0;
1130 #endif
1131                         op[0] = ref[0];
1132                         op[1] = ref[1];
1133                         op[2] = ref[2];
1134                         op[3] = ref[3];
1135                         op += 4;
1136                         ref += 4;
1137                         ref -= dec32table[op-ref];
1138                         A32(op) = A32(ref);
1139                         op += STEPSIZE - 4;
1140                         ref -= dec64;
1141                 } else {
1142                         LZ4_COPYSTEP(ref, op);
1143                 }
1144                 cpy = op + length - (STEPSIZE - 4);
1145                 if (cpy > oend - COPYLENGTH) {
1146                         if (cpy > oend)
1147                                 /*
1148                                  * Error: request to write outside of
1149                                  * destination buffer
1150                                  */
1151                                 goto _output_error;
1152                         LZ4_SECURECOPY(ref, op, (oend - COPYLENGTH));
1153                         while (op < cpy)
1154                                 *op++ = *ref++;
1155                         op = cpy;
1156                         if (op == oend)
1157                                 /*
1158                                  * Check EOF (should never happen, since
1159                                  * last 5 bytes are supposed to be literals)
1160                                  */
1161                                 goto _output_error;
1162                         continue;
1163                 }
1164                 LZ4_SECURECOPY(ref, op, cpy);
1165                 op = cpy;       /* correction */
1166         }

1168         /* end of decoding */
1169         return (int)(((char *)op) - dest);

1171         /* write overflow error detected */
1172         _output_error:
1173         return (int)(-(((char *)ip) - source));
1174 }
```

_____***unchanged_portion_omitted_***