

// Railgun\_Trolddom (the successor of Railgun\_Swampshine\_BailOut - avoiding second pattern comparison in BMH2 and pseudo-BMH4), copleft 2016-Aug-19, Kaze.  
// Railgun\_Swampshine\_BailOut, copleft 2016-Aug-10, Kaze.  
// Internet "home" page: <http://www.codeproject.com/Articles/250566/Fastest-strstr-like-function-in-C>  
// My homepage (homeserver, often down): <http://www.sanmayce.com/Railgun/>  
/\*  
!!!!!!!!!!!!!!!!!!!!!!!!!!!! BENCHMARKING GNU's memmem vs Railgun !!!!!!!!!!!!!!!!!!!!!!!!!!!!! [

Add-on: 2016-Aug-22  
  
Two things.

First, the fix from the last time was buggy, my apologies, now fixed, quite embarrassing since it is a simple left/right boundary check. It doesn't affect the speed, it appears as rare pattern hit misses.

Since I don't believe in saying "sorry" but in making things right, here my attempt to further disgrace my amateurish work follows:

Two years ago, I didn't pay due attention to adding 'Swampwalker' heuristic to the Railgun\_Ennearch, I mean, only quick test was done and no real proofing - this was due not to a blunder of mine, nor carelessness, but overconfidence in my ability to write "on the fly". Stupid, indeed, however, when a coder gets momentum in writing simple etudes he starts gaining false confidence of mastering the subject, not good for sure!

Hopefully, other coders will learn to avoid such full of neglect style.

Second, wanted to present the heaviest testbed for search i.e. `memmem()` functions: it benefits the benchmarking (speed in real application) as well as bug-control.

The benchmark is downloadable at my INTERNET drive:  
<https://1drv.ms/u/s!AmwWFXGMZDmEqwljUtnMJrfhosk>

The speed showdown has three facets:

- compares the 64bit code generated from GCC 5.10 versus Intel 15.0 compilers;
- compares four types of datasets - search speed through English texts versus genome ACGT-type data versus binary versus UTF8;
- compares the tweaked Two-way algorithm (implemented by Eric Blake) and adopted by GLIBC as `memmem()` versus my `Railgun_Swampshine`.

Note1: The GLIBC memmem() was taken from latest (2016-08-05) glibc 2.24 tar:

<https://www.gnu.org/software/libc/>

Note2: Eric Blake says that he enhanced the linearity of Two-way by adding some sublinear paths, well, Railgun is all about sublinearity, so feel free to experiment with your own testfiles (worst-case-scenarios), just make such a file feed the compressor with it, then we will see how the LINEAR Two-way behaves versus Railgun Swampshine.

Note3: Just copy-and-paste 'Railgun\_Swampshine' or 'Railgun\_Ennearch' from the benchmark's source.

So the result on Core 2 Q9550s @2.83GHz DDR2 @666MHz / i5-2430M @3.00GHz DDR3 @666MHz:

Searcher	GNU/GLIBC memmem()	Railgun_Swampshine	Railgun_TrollDom
Testfile\Compiler	Intel 15.0   GCC 5.10	Intel 15.0   GCC 5.10	Intel 15.0   GCC 5.10
Size: 27,703 bytes Name: An_Interview_with_Carlos_Castaneda.TXT LATENCY-WISE: Number of 'memmem()' Invocations: 308,062 THROUGHPUT-WISE: Number of Total bytes Traversed: 3,242,492,648	4506/-   5330/14725	13198/-   11581/15171	19105/22449   15493/21642
Size: 2,347,772 bytes Name: Gutenberg_EBook_Don_Quixote_996_(ANSI).txt LATENCY-WISE: Number of 'memmem()' Invocations: 14,316,954 THROUGHPUT-WISE: Number of Total bytes Traversed: 6,663,594,719,173	190/-   226/244	1654/-   1729/1806	1794/1822   1743/1809
Size: 899,425 bytes Name: Gutenberg_EBook_Dokoe_by_Hakucho_Masamune_(Japanese_UTF8).txt LATENCY-WISE: Number of 'memmem()' Invocations: 3,465,806 THROUGHPUT-WISE: Number of Total bytes Traversed: 848,276,034,315	582/-   760/816	3094/-   2898/3088	3255/3289   2915/3322
Size: 4,487,433 bytes Name: Dragonfly_genome_shotgun_sequence_(ACGT_alphabet).fasta LATENCY-WISE: Number of 'memmem()' Invocations: 20,540,375 THROUGHPUT-WISE: Number of Total bytes Traversed: 13,592,530,857,131	104/-   109/116	445/-   458/417	450/411   467/425
Size: 954,035 bytes Name: LAOTZU_wu_wei_(BINARY).pdf LATENCY-WISE: Number of 'memmem()' Invocations: 27,594,933 THROUGHPUT-WISE: Number of Total bytes Traversed: 8,702,455,122,519	99/-   144/144	629/-   580/682	634/807   585/725
Size: 15,583,440 bytes Name: Arabian_Nights_complete.html LATENCY-WISE: Number of 'memmem()' Invocations: 72,313,262 THROUGHPUT-WISE: Number of Total bytes Traversed: 105,631,163,854,099	-/-   -/-	-/-   663/771	675/778   663/757

Note0: Railgun\_Troldom is slightly faster (both with Intel & GCC) than Railgun\_Swampshine, this is mostly due to adding a bitwise BMH order 2 (8KB table overhead instead of 64KB) path - for haystacks <77777 bytes long - the warm-up is faster.

Note1: The numbers represent the rate (bytes/s) at which patterns/needles 4,5,6,8,9,10,12,13,14,16,17,18,24 bytes long are memmmed into 4KB, 256KB, 1MB, 256MB long haystacks.

in fact, these numbers are the compression speed using LZSS and memmem() as matchfinder.

Note2: The Arabian Nights is downloadable at:

NOTE2: The Arabian Nights is downloadable at:  
<https://ebooks.adelaide.edu.au/b/burton/richard/b97b/complete.html>

Note3: On i5-2430M, TW is catching up since this CPU crunches instructions faster while the RAM speed is almost the same, Railgun suffers from the slow RAM fetches - the prefetcher and such suck.

Note4: With a simple English text 'Tales of 1001 Nights', 15,583,440 bytes long, the cumulative size of traversed haystack data is nearly 100TB, 105,631,163,854,099 ~ 1024x4 = 1.099.511.627.776.

Notes: With a simple French text 'Agatha\_Christie\_85-ebooks\_(French)\_TXT.tar', 32,007,168 bytes long, the cumulative size of traversed haystack data is nearly 200TB ~ 234,427,099,834,376.

Just to see how faster is Yann's Zstd in decompression (its level 12 is 377-331 MB/s faster). on Core 2 Q9550s @2.83GHz DDR2 @666MHz:

[illegible]

D:\Nakamichi\_Kintaro+\_source\_executables\_64bit\_(GCC510-vs-Intel150)\_(TW-vs-RG)\_BENCHMARK\Nakamichi\_Kintaro+\_Intel\_15.0\_64bit.exe Agatha\_Christie\_85-ebooks\_(French)\_TXT.tar  
Nakamichi 'Kintaro+', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m2 enforced, mufinesque suggestion by Jim Demsey enforced.

Note1: This compile can handle files up to 1711MB.

Note2: The matchfinder/memmem() is Railgun Trolldom.

```
Current priority class is HIGH_PRIORITY_CLASS.
```

```
Compressing 32007168 bytes ...
```

|; Each rotation means 64KB are encoded; Don

NumberOfFullLiterals (lower-the-better): 164

NumberOfFlushLiteralsHeuristic (bigger-the-better): 184323

Legend: windowSizes: 1/2/3/4=Tiny/Short/Medium/Long

NumberOf(Tiny)Matches[Short]window (4)[2]: 226869

NumberOf(Short)Matches[Short>window (8)[2]: 119810

```
NumberOf(Medium)Matches[Short]window (12)[2]: 71202
```

```
NumberOf(Long)Matches[Short]Window (16)[2]: 31955
```

```
NumberOf(MaxLong)Matches[Short]Window (24)[2]: 7078
NumberOf(MaxLong)Matches[Fixed]Window (5)[2]: 257313
```

```
NumberOf(Tiny)Matches[Medium]Window (5)[3]: 257313
NumberOf(Short)Matches[Medium]Window (0)[2]: 526403
```

NumberOf(Medium)Matches[Medium]Window (9)[3]: 526493  
NumberOf(Medium)Matches[Medium]Window (13)[3]: 3855

NumberOf(Medium)Matches[Medium]Window (13)[3]: 2853  
NumberOf(Long)Matches[Medium]Window (17)[3]: 158873

NumberOf(Long)Matches[Medium]window (17)[3]: 138873  
NumberOf(Max,Long)Matches[Medium]window (34)[3]: 513

```
NumberOf(Medium)Matches[Medium]window (24)[5]: 512
NumberOf(Tiny)Matches[Long]window (6)[4]: 41075
```

NumberOf(Short)Matches[Long]window (10)[4]: 240454

Listing: **Railgun Trolldom**. copyleft 2016-Aug-19. <http://www.sanmayce.com/Railgun/>

page 3 of 28

```

uint32_t PRIMALposition, PRIMALpositionCANDIDATE;
uint32_t PRIMALlength, PRIMALlengthCANDIDATE;
uint32_t j, FoundAtPosition;

// Quadruplet [
//char * pbTargetMax = pbTarget + cbTarget;
//register unsigned long ulHashPattern;
unsigned long ulHashTarget;
//unsigned long count;
unsigned long countSTATIC;
unsigned char SINGLET;
unsigned long Quadruplet2nd;
unsigned long Quadruplet3rd;
unsigned long Quadruplet4th;
unsigned long AdvanceHopperGrass;
// Quadruplet ]

if (cbPattern > cbTarget) return(NULL);

if ( cbPattern<4 ) {
    // SSE2 i.e. 128bit Assembly rules here, Mischa knows best:
    // ...
    pbTarget = pbTarget+cbPattern;
    ulHashPattern = ( (*char *) (pbPattern)<<8 ) + *(pbPattern+(cbPattern-1));
    if ( cbPattern==3 ) {
        for ( ;; ) {
            if ( ulHashPattern == ( (*char *) (pbTarget-3)<<8 ) + *(pbTarget-1) ) {
                if ( (*char *) (pbPattern+1) == (*char *) (pbTarget-2) ) return((pbTarget-3));
            }
            if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) {
                pbTarget++;
                if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) pbTarget++;
            }
            pbTarget++;
            if (pbTarget > pbTargetMax) return(NULL);
        }
    } else {
        for ( ;; ) {
            if ( ulHashPattern == ( (*char *) (pbTarget-2)<<8 ) + *(pbTarget-1) ) return((pbTarget-2));
            if ( (char)(ulHashPattern>>8) != *(pbTarget-1) ) pbTarget++;
            pbTarget++;
            if (pbTarget > pbTargetMax) return(NULL);
        }
    }
} else { //if ( cbPattern<4 )
    if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {

// This is the awesome 'Railgun_Quadruplet', it did outperform EVERYWHERE the fastest strstr (back in old GLIBCes ~2003, by the Dutch hacker Stephen R. van den Berg),
// suitable for short haystacks ~100bytes.
// Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
// char * Railgun_Quadruplet (char * pbTarget, char * pbPattern, unsigned long cbTarget, unsigned long cbPattern)
// ...
// if (cbPattern > cbTarget) return(NULL);
//} else { //if ( cbPattern<4 )
if (cbTarget<777) // This value is arbitrary(don't know how exactly), it ensures(at least must) better performance than 'Boyer_Moore_Horspool'.
{
    pbTarget = pbTarget+cbPattern;
    ulHashPattern = *(unsigned long *) (pbPattern);
    // countSTATIC = cbPattern-1;

    //SINGLET = *(char *) (pbPattern);
    SINGLET = ulHashPattern & 0xFF;
    Quadruplet2nd = SINGLET<<8;
    Quadruplet3rd = SINGLET<<16;
    Quadruplet4th = SINGLET<<24;

    for ( ;; )
    {
        AdvanceHopperGrass = 0;
        ulHashTarget = *(unsigned long *) (pbTarget-cbPattern);

        if ( ulHashPattern == ulHashTarget ) { // Three unnecessary comparisons here, but 'AdvanceHopperGrass' must be calculated - it has a higher priority.
            count = countSTATIC;
            while ( count && (*char *) (pbPattern+1+(countSTATIC-count)) == (*char *) (pbTarget-cbPattern+1+(countSTATIC-count)) ) {
                if ( countSTATIC==AdvanceHopperGrass+count && SINGLET != (*char *) (pbTarget-cbPattern+1+(countSTATIC-count)) ) AdvanceHopperGrass++;
                count--;
            }
            count = cbPattern-1;
            while ( count && (*char *) (pbPattern+(cbPattern-count)) == (*char *) (pbTarget-count) ) {
                if ( cbPattern-1==AdvanceHopperGrass+count && SINGLET != (*char *) (pbTarget-count) ) AdvanceHopperGrass++;
                count--;
            }
            if ( count == 0 ) return((pbTarget-cbPattern));
        } else { // The goal here: to avoid memory accesses by stressing the registers.
            if ( Quadruplet2nd != (ulHashTarget & 0x0000FF00) ) {
                AdvanceHopperGrass++;
                if ( Quadruplet3rd != (ulHashTarget & 0x00FF0000) ) {
                    AdvanceHopperGrass++;
                    if ( Quadruplet4th != (ulHashTarget & 0xFF000000) ) AdvanceHopperGrass++;
                }
            }
        }
    }
}

AdvanceHopperGrass++;

pbTarget = pbTarget + AdvanceHopperGrass;
if (pbTarget > pbTargetMax)
    return(NULL);
}

} else if (cbTarget<77777) { // The warmup/overhead is lowered from 64K down to 8K, however the bitwise additional instructions quickly start hurting the
Listing: Railgun_Troldom, copyleft 2016-Aug-19, http://www.sanmayce.com/Railgun/

```

```

throughput/traversal.
// The below bitwise 0|1 BMH2 gives 1427 bytes/s for 'Don_Quixote' with Intel:
// The below bitwise 0|1 BMH2 gives 1242 bytes/s for 'Don_Quixote' with GCC:
// } else { //if ( cbPattern<4 )
//     if ( cbPattern<=NeedleThreshold2vs4Decumanus ) {
//         // BMH order 2, needle should be >=4:
//         ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
//         for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
//         for (i=0; i < (256*256)>>3; i++) {bm_Horspool_Order2bitwise[i]=0;}
//         for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=1;
//         for (i=0; i < cbPattern-2+1; i++) bm_Horspool_Order2bitwise[(*(unsigned short *) (pbPattern+i))>>3] = bm_Horspool_Order2bitwise[(*(unsigned short *) (pbPattern+i))>>3] | (1<<((*(unsigned short *) (pbPattern+i))&0x7));
//         i=0;
//         while (i <= cbTarget-cbPattern) {
//             Gulliver = 1; // 'Gulliver' is the skip
//             //if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
//                 if ( ( bm_Horspool_Order2bitwise[(*(unsigned short *)&pbTarget[i+cbPattern-1-1])>>3] & (1<<((*(unsigned short *)&pbTarget[i+cbPattern-1-1])&0x7)) ) != 0 ) {
//                     //if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
//                         if ( ( bm_Horspool_Order2bitwise[(*(unsigned short *)&pbTarget[i+cbPattern-1-1-2])>>3] & (1<<((*(unsigned short *)&pbTarget[i+cbPattern-1-1-2])&0x7)) ) == 0 ) Gulliver = cbPattern-(2-1)-2; else {
//                             if ( *(uint32_t *)&pbTarget[i] == ulHashPattern ) { // This fast check ensures not missing a match (for remainder)
//                                 when going under 0 in loop below:
//                                     count = cbPattern-4+1;
//                                     while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i+(count-1)) )
//                                         count = count-4;
//                                     if ( count <= 0 ) return(pbTarget+i);
//                                 }
//                             } else Gulliver = cbPattern-(2-1);
//                             i = i + Gulliver;
//                             //GlobalI++; // Comment it, it is only for stats.
//                         }
//                     }
//                 }
//             }
//             return(NULL);
//         } else { //if ( cbPattern<=NeedleThreshold2vs4Decumanus )
//         } else { //if (cbTarget<777)
//             // BMH order 2, needle should be >=4:
//             ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
//             for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
//             for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=1;
//             i=0;
//             while (i <= cbTarget-cbPattern) {
//                 Gulliver = 1; // 'Gulliver' is the skip
//                 if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
//                     if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
//                         if ( *(uint32_t *)&pbTarget[i] == ulHashPattern ) { // This fast check ensures not missing a match (for remainder)
//                             when going under 0 in loop below:
//                                 count = cbPattern-4+1;
//                                 while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i+(count-1)) )
//                                     count = count-4;
//                                 if ( count <= 0 ) return(pbTarget+i);
//                             }
//                         } else Gulliver = cbPattern-(2-1);
//                         i = i + Gulliver;
//                         //GlobalI++; // Comment it, it is only for stats.
//                     }
//                 }
//                 return(NULL);
//             }
//         }
//     }
// }
// Slower than Swampshine's simple 0|1 segment:
/*
PRIMALlength=0;
for (i=0+1; i < cbPattern-2+1+(1)-(1); i++) { // -(1) because the last BB order 2 has no counterpart(s)
    FoundAtPosition = cbPattern;
    PRIMALpositionCANDIDATE=i;
    while ( PRIMALpositionCANDIDATE <= (FoundAtPosition-1) ) {
        j = PRIMALpositionCANDIDATE + 1;
        while ( j <= (FoundAtPosition-1) ) {
            if ( *(unsigned short *) (pbPattern+PRIMALpositionCANDIDATE-(1)) == *(unsigned short *) (pbPattern+j-(1)) ) FoundAtPosition = j;
            j++;
        }
        PRIMALpositionCANDIDATE++;
    }
    PRIMALlengthCANDIDATE = (FoundAtPosition-1)-i+(2);
    if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=i; PRIMALlength = PRIMALlengthCANDIDATE;}
}
PRIMALlengthCANDIDATE = cbPattern;
cbPattern = PRIMALlength;
pbPattern = pbPattern + (PRIMALposition-1);
if (cbPattern<4) {
    cbPattern = PRIMALlengthCANDIDATE;
    pbPattern = pbPattern - (PRIMALposition-1);
}
if (cbPattern == PRIMALlengthCANDIDATE) {
    // BMH order 2, needle should be >=4:
    ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
    for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
    for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=1;
    i=0;
    while (i <= cbTarget-cbPattern) {
        Gulliver = 1; // 'Gulliver' is the skip
        if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
            if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
                if ( *(uint32_t *)&pbTarget[i] == ulHashPattern ) { // This fast check ensures not missing a match (for remainder)
                    when going under 0 in loop below:
                        count = cbPattern-4+1;
                        while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i+(count-1)) )
                            count = count-4;
                        if ( count <= 0 ) return(pbTarget+i);
                    }
                }
            }
        }
    }
}

```



```

    }
    } else Gulliver = cbPattern-(2-1);
    i = i + Gulliver;
    //GlobalI++; // Comment it, it is only for stats.
}
return(NULL);
} else { //if (cbPattern == PRIMALlengthCANDIDATE) {
// BMH Order 2 [
    ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
    for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]= cbPattern-1;} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
    // The above 'for' gives 1424 bytes/s for 'Don_Quixote' with Intel:
    // The above 'for' gives 1431 bytes/s for 'Don_Quixote' with GCC:
    // The below 'memset' gives 1389 bytes/s for 'Don_Quixote' with Intel:
    // The below 'memset' gives 1432 bytes/s for 'Don_Quixote' with GCC:
    //memset(&bm_Horspool_Order2[0], cbPattern-1, 256*256); // why why? It is 1700:1000 slower than above 'for'?
    for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=i; // Rightmost appearance/position is needed
    i=0;
    while (i <= cbTarget-cbPattern) {
        Gulliver = bm_Horspool_Order2[(unsigned short *) &pbTarget[i+cbPattern-1]];
        if ( Gulliver != cbPattern-1 ) { // CASE #2: if equal means the pair (char order 2) is not found i.e. Gulliver remains intact, skip the
whole pattern and fall back (Order-1) chars i.e. one char for Order 2
            if ( Gulliver == cbPattern-2 ) { // CASE #1: means the pair (char order 2) is found
                if ( *(uint32_t *) &pbTarget[i] == ulHashPattern ) {
                    count = cbPattern-4+1;
                    while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )
                        count = count-4;
                }
            }
        }
        // If we miss to hit then no need to compare the original: Needle
        if ( count <= 0 ) {
            // I have to add out-of-range checks...
            // i-(PRIMALposition-1) >= 0
            // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
            // i-(PRIMALposition-1)+(count-1) >= 0
            // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4

            // "FIX" from 2014-Apr-27:
            // Because (count-1) is negative, above fours are reduced to next twos:
            // i-(PRIMALposition-1)+(count-1) >= 0
            // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
            // The line below is BUGGY:
            //if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
            // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
            //if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
            // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
            //if ( ((signed int)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) {
            if ( *(uint32_t *) &pbTarget[i-(PRIMALposition-1)] == *(uint32_t *) (pbPattern-(PRIMALposition-1)) ) { // This fast check ensures not missing a match (for remainder)
when going under 0 in loop below:
                count = PRIMALlengthCANDIDATE-4+1;
                while ( count > 0 && *(uint32_t *) (pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *) (&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
                    count = count-4;
                if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
            }
        }
    }
}
} else
    Gulliver = 1;
    Gulliver = cbPattern - Gulliver - 2; // CASE #3: the pair is found and not as suffix i.e. rightmost position
    i = i + Gulliver;
    //GlobalI++; // Comment it, it is only for stats.
}
return(NULL);
// BMH Order 2 ]
} //if (cbPattern == PRIMALlengthCANDIDATE) {
*/
/*
So the result on Core 2 Q9550s @2.83GHz:

```

testfile\Searcher	GNU/GLIBC memmem()		Railgun_Swampshine		Railgun_Trolldom	
Compiler	Intel 15.0	GCC 5.10	Intel 15.0	GCC 5.10	Intel 15.0	GCC 5.10
The_Project_Gutenberg_EBook_of_Don_Quixote_996_(ANSI).txt 2,347,772 bytes	190	226	1654	1729	1147	1764
The_Project_Gutenberg_EBook_of_Dokoe_by_Hakucho_Masamune_(Japanese_UTF-8).txt 899,425 bytes	582	760	3094	2898	2410	3036
DragonFly_genome_shotgun_sequence_(ACGT_alphabet).fasta 4,487,433 bytes	104	109	445	458	484	553
LAOTZU_Wu_Wei_(BINARY).pdf 954,035 bytes	99	144	629	580	185	570

Below segment (when compiled with Intel) is very slow, see Railgun\_Trolldom two sub-columns above, compared to GCC:

```

/*
/*
// BMH Order 2 [
    ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
    for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]= (cbPattern-1);} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
    // The above 'for' is translated by Intel as:

//.B5.21::
// 0013f 83 c0 40      add eax, 64
// 00142 66 0f 7f 44 14  movdqa XMMWORD PTR [96+rsp+rdx], xmm0
// 60
// 00148 3d 00 00 01 00  cmp eax, 65536

```

Listing: Railgun\_Trolldom, copyleft 2016-Aug-19, <http://www.sanmayce.com/Railgun/>

```

// 0014d 66 0f 7f 44 14
// 70 movdqa XMMWORD PTR [112+rsp+rdx], xmm0
// 00153 66 0f 7f 84 14
// 80 00 00 00 movdqa XMMWORD PTR [128+rsp+rdx], xmm0
// 0015c 66 0f 7f 84 14
// 90 00 00 00 movdqa XMMWORD PTR [144+rsp+rdx], xmm0
// 00165 89 c2 mov edx, eax
// 00167 72 d6 jb .B5.21
//memset(&bm_Horspool_Order2[0], cbPattern-1, 256*256); // why why? It is 1700:1000 slower than above 'for'!?
// The above 'memset' is translated by Intel as:
// 00127 41 b8 00 00 01
// 00 mov r8d, 65536
// 0012d 44 8b 26 mov r12d, DWORD PTR [rsi]
// 00130 e8 fc ff ff call _intel_fast_memset
// ! The problem is that 256*256, 64KB, is already too much, going bitwise i.e. 8KB is not that better, when 'cbPattern-1' is bigger than 255 - an
unsigned char - then
// we must switch to 0|1 table i.e. present or not. Since we are in 'if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {' branch and
NeedleThreshold2vs4swampLITE, by default, is 19 - it is okay to use 'memset'. !
for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=i; // Rightmost appearance/position is needed
i=0;
while (i <= cbTarget-cbPattern) {
    Gulliver = bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1]];
    if ( Gulliver != cbPattern-1 ) { // CASE #2: if equal means the pair (char order 2) is not found i.e. Gulliver remains intact, skip the
whole pattern and fall back (Order-1) chars i.e. one char for Order 2
        if ( Gulliver == cbPattern-2 ) { // CASE #1: means the pair (char order 2) is found
            if ( *(uint32_t *)&pbTarget[i] == ulHashPattern ) {
                count = cbPattern-4+1;
                while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )
                    count = count-4;
                if ( count <= 0 ) return(pbTarget+i);
            }
            Gulliver = 1;
        } else
            Gulliver = cbPattern - Gulliver - 2; // CASE #3: the pair is found and not as suffix i.e. rightmost position
        i = i + Gulliver;
        //GlobalI++; // Comment it, it is only for stats.
    }
}
return(NULL);
// BMH Order 2 ]
*/
// Above fragment in Assembly:
/*
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140";
; mark_description "-O3 -QxSSE2 -D_N_XMM -D_N_prefetch_4096 -D_N_Branchfull -D_N_HIGH_PRIORITY -FA";
ALIGN 16
.B6.1:: ; Preds .B6.0
    push    rbx ;3435.1
    push    r13 ;3435.1
    push    r15 ;3435.1
    push    rbp ;3435.1
    mov     eax, 65592 ;3435.1
    call    __chkstk ;3435.1
    sub     rsp, 65592 ;3435.1
    cmp     r9d, r8d ;3460.18
    ja      .B6.25 ;3460.18
    ; Prob 28%
    ; LOE rdx rcx rbx rsi rdi r12 r14 r8d r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B6.1
.B6.3::
    mov     r13d, DWORD PTR [rdx] ;3491.33
    lea     ebp, DWORD PTR [-1+r9] ;3492.67
    movzx   eax, bpl ;3492.67
    xor     r10d, r10d ;3492.4
    movd    xmm0, eax ;3492.67
    xor     eax, eax ;3492.4
    punpcklbw xmm0, xmm0 ;3492.67
    punpcklwd xmm0, xmm0 ;3492.67
    punpckldq xmm0, xmm0 ;3492.67
    punpcklq dq xmm0, xmm0 ;3492.67
    ; LOE rdx rcx rbx rsi rdi r10 r12 r14 eax ebp r8d r9d r13d xmm0 xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B6.4 .B6.3
.B6.4::
    add     eax, 64 ;3492.4
    movdqa  XMMWORD PTR [48+rsp+r10], xmm0 ;3492.33
    cmp     eax, 65536 ;3492.4
    movdqa  XMMWORD PTR [64+rsp+r10], xmm0 ;3492.33
    movdqa  XMMWORD PTR [80+rsp+r10], xmm0 ;3492.33
    movdqa  XMMWORD PTR [96+rsp+r10], xmm0 ;3492.33
    mov     r10d, eax ;3492.4
    jb      .B6.4 ;3492.4
    ; Prob 99%
    ; LOE rdx rcx rbx rsi rdi r10 r12 r14 eax ebp r8d r9d r13d xmm0 xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B6.4
.B6.5::
    test    ebp, ebp ;3515.28
    je      .B6.12 ;3515.28
    ; Prob 50%
    ; LOE rdx rcx rbx rsi rdi r12 r14 ebp r8d r9d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B6.5
.B6.6::
    mov     eax, 1 ;3515.4
    lea     r11d, DWORD PTR [-1+r9] ;3515.4
    mov     r15d, r11d ;3515.4
    xor     r10d, r10d ;3515.4
    shr     r15d, 1 ;3515.4
    test    r15d, r15d ;3515.4
    jbe     .B6.10 ;3515.4
    ; Prob 15%
    ; LOE rdx rcx rbx rsi rdi r12 r14 eax ebp r8d r9d r10d r11d r13d r15d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B6.6 .B6.8
.B6.8::
    lea     eax, DWORD PTR [r10+r10] ;3515.36
    movzx   ebx, WORD PTR [rax+rdx] ;3515.75
    mov     BYTE PTR [48+rsp+rbx], al ;3515.36
    lea     eax, DWORD PTR [1+r10+r10] ;3515.36
    inc     r10d ;3515.4
    cmp     r10d, r15d ;3515.4
    movzx   ebx, WORD PTR [rax+rdx] ;3515.75

```

```

mov     BYTE PTR [48+rsp+rbx], al                ;3515.36
jb      .B6.8                                   ;3515.4
; LOE rdx rcx rsi rdi r12 r14 ebp r8d r9d r10d r11d r13d r15d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.8
.B6.9:: lea     eax, DWORD PTR [1+r10+r10]        ;3515.4
; LOE rdx rcx rbx rsi rdi r12 r14 eax ebp r8d r9d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.9 .B6.6
.B6.10:: dec     eax                             ;3515.36
cmp     eax, r11d                               ;3515.4
jae     .B6.12                                  ;3515.4
; Prob 15%
; LOE rax rcx rcx rbx rsi rdi r12 r14 ebp r8d r9d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.10
.B6.11:: movzx   r10d, WORD PTR [rax+rdx]         ;3515.75
mov     BYTE PTR [48+rsp+r10], al                ;3515.36
; LOE rdx rcx rbx rsi rdi r12 r14 ebp r8d r9d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.5 .B6.10 .B6.11
.B6.12:: xor     r10d, r10d                       ;3516.4
lea     r15d, DWORD PTR [-3+r9]                 ;3522.27
movsxd  r15, r15d                               ;3522.7
sub     r8d, r9d                                ;3517.16
lea     r11d, DWORD PTR [-2+r9]                 ;3520.32
; LOE rdx rcx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.12 .B6.24
.B6.13:: lea     eax, DWORD PTR [-2+r9+r10]        ;3518.78
movzx   ebx, WORD PTR [rax+rcx]                 ;3518.55
movzx   eax, BYTE PTR [48+rsp+rbx]              ;3518.16
cmp     eax, ebp                                ;3519.32
je      .B6.24                                  ;3519.32
; Prob 50%
; LOE rdx rcx rsi rdi r12 r14 r15 eax ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.13
.B6.14:: cmp     eax, r11d                       ;3520.32
jne     .B6.23                                  ;3520.32
; Prob 62%
; LOE rdx rcx rsi rdi r12 r14 r15 eax ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.14
.B6.15:: mov     eax, r10d                       ;3521.25
add     rax, rcx                                ;3521.25
r13d, DWORD PTR [rax]                          ;3521.40
je      .B6.17                                  ;3521.40
; Prob 50%
; LOE rax rdx rcx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.26 .B6.15
.B6.16:: mov     eax, 1                         ;3527.6
jmp     .B6.24                                  ;3527.6
; Prob 100%
; LOE rdx rcx rsi rdi r12 r14 r15 eax ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.15
.B6.17:: mov     rbx, r15                       ;3522.7
test    r15, r15                               ;3523.23
jle     .B6.22                                  ;3523.23
; Prob 2%
; LOE rax rdx rcx rbx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.17
.B6.18:: mov     QWORD PTR [32+rsp], rsi          ;
; LOE rax rdx rcx rbx rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.20 .B6.18
.B6.19:: mov     esi, DWORD PTR [-1+rbx+rdx]       ;3523.58
cmp     esi, DWORD PTR [-1+rbx+rax]             ;3523.79
jne     .B6.26                                  ;3523.79
; Prob 20%
; LOE rax rdx rcx rbx rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.19
.B6.20:: add     rbx, -4                          ;3524.22
test    rbx, rbx                               ;3523.23
jg      .B6.19                                  ;3523.23
; Prob 82%
; LOE rax rdx rcx rbx rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.20
.B6.21:: mov     rsi, QWORD PTR [32+rsp]           ;
; LOE rax rbx rsi rdi r12 r14 xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.17 .B6.21
.B6.22:: add     rsp, 65592                       ;3525.32
pop     rbp                                     ;3525.32
pop     r15                                     ;3525.32
pop     r13                                     ;3525.32
pop     rbx                                     ;3525.32
ret                                           ;3525.32
; LOE
; Preds .B6.14
.B6.23:: neg     eax                             ;3529.17
add     eax, r9d                               ;3529.17
add     eax, -2                                ;3529.40
; LOE rdx rcx rsi rdi r12 r14 r15 eax ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.16 .B6.23 .B6.13
.B6.24:: add     r10d, eax                       ;3531.13
cmp     r10d, r8d                             ;3517.25
jbe     .B6.13                                  ;3517.25
; Prob 82%
; LOE rdx rcx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B6.1 .B6.24
.B6.25:: xor     eax, eax                         ;3534.10
add     rsp, 65592                             ;3534.10
pop     rbp                                     ;3534.10
pop     r15                                     ;3534.10
pop     r13                                     ;3534.10
pop     rbx                                     ;3534.10
ret                                           ;3534.10
; LOE
; Preds .B6.19
.B6.26:: mov     rsi, QWORD PTR [32+rsp]           ;
jmp     .B6.16                                  ;
; Prob 100%
;
*/

```

// GCC 5.10; >gcc -O3 -m64 -fomit-frame-pointer

/\*

Railgun\_TroIldom:

pushq%r15

Listing: Railgun\_TroIldom, copyleft 2016-Aug-19, <http://www.sanmayce.com/Railgun/>



```

.seh_pushreg    %r15
movl $65592, %eax
pushq%r14
.seh_pushreg    %r14
pushq%r13
.seh_pushreg    %r13
pushq%r12
.seh_pushreg    %r12
pushq%rbp
.seh_pushreg    %rbp
pushq%rdi
.seh_pushreg    %rdi
pushq%rsi
.seh_pushreg    %rsi
pushq%rbx
.seh_pushreg    %rbx
call ____chkstk_ms
subq %rax, %rsp
.seh_stackalloc 65592
.seh_endprologue
cmpl %r9d, %r8d
movq %rcx, %rbx
movq %rdx, %rdi
movl %r8d, %r12d
movl %r9d, %esi
jb .L118
movl (%rdx), %ebp
leal -1(%r9), %edx
movl $65536, %r8d
leaq 48(%rsp), %rcx
movzbl %dl, %edx
call memset
movl %esi, %r11d
subl $1, %r11d
je .L119
xorl %eax, %eax
.p2align 4,,10
.L113:
movzwl (%rdi,%rax), %edx
movb %al, 48(%rsp,%rdx)
addq $1, %rax
cmpl %eax, %r11d
ja .L113
.L112:
leal -4(%rsi), %r9d
movl %r12d, %r8d
xorl %edx, %edx
leal -3(%rsi), %eax
shrl $2, %r9d
subl %esi, %r8d
leal -2(%rsi), %r10d
movslq %eax, %r14
negq %r9
movl %eax, 44(%rsp)
leaq -1(%r14), %r15
salq $2, %r9
leaq (%rdi,%r14), %r13
jmp .L117
.p2align 4,,10
.L130:
movl %r10d, %eax
subl %ecx, %eax
cmpl %r10d, %ecx
je .L129
.L114:
addl %eax, %edx
cmpl %r8d, %edx
ja .L118
.L117:
leal (%rdx,%r10), %eax
movzwl (%rbx,%rax), %eax
movzbl 48(%rsp,%rax), %ecx
cmpl %r11d, %ecx
jne .L130
movl %r11d, %eax
addl %eax, %edx
cmpl %r8d, %edx
jbe .L117
.L118:
xorl %eax, %eax
jmp .L128
.p2align 4,,10
.L129:
movl %edx, %ecx
movl $1, %eax
leaq (%rbx,%rcx), %r12
cmpl (%r12), %ebp
jne .L114
movl 44(%rsp), %esi
testl %esi, %esi
jle .L124
movl (%r12,%r15), %esi
cmpl %esi, (%rdi,%r15)
jne .L114
addq %r14, %rcx
xorl %eax, %eax
addq %rbx, %rcx
jmp .L116
.p2align 4,,10
.L132:

```

```

movl -5(%r13,%rax), %esi
subq $4, %rax
cmpl -1(%rcx,%rax), %esi
jne .L131
.L116:
cmpq %rax, %r9
jne .L132
.L124:
movq %r12, %rax
.L128:
addq $65592, %rsp
popq %rbx
popq %rsi
popq %rdi
popq %rbp
popq %r12
popq %r13
popq %r14
popq %r15
ret
.p2align 4,,10
.L131:
movl $1, %eax
jmp .L114
.L119:
xorl %r11d, %r11d
jmp .L112
*/
} //if (cbTarget<777)

        } else { // if ( cbPattern<=NeedleThreshold2vs4swampLITE )

// Swampwalker_BAILOUT heuristic order 4 (Needle should be bigger than 4) [
// Needle: 1234567890qwertyuiopasdfghjklzxcv          PRIMALposition=01 PRIMALlength=33 '1234567890qwertyuiopasdfghjklzxcv'
// Needle: vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv    PRIMALposition=29 PRIMALlength=04 'vvvv'
// Needle: vvvvvvvvvvBOOMSHAKALAKAvvvvvvvvvvv         PRIMALposition=08 PRIMALlength=20 'vvvBOOMSHAKALAKAvvvv'
// Needle: Trollland                                   PRIMALposition=01 PRIMALlength=09 'Trollland'
// Needle: Swampwalker                                 PRIMALposition=01 PRIMALlength=11 'Swampwalker'
// Needle: licenselessness                             PRIMALposition=01 PRIMALlength=15 'licenselessness'
// Needle: alfalfa                                     PRIMALposition=02 PRIMALlength=06 'lfa'
// Needle: Sandokan                                  PRIMALposition=01 PRIMALlength=08 'Sandokan'
// Needle: shazamish                                  PRIMALposition=01 PRIMALlength=09 'shazamish'
// Needle: Simplicius Simplicissimus                  PRIMALposition=06 PRIMALlength=20 'icius Simplicissimus'
// Needle: domilliaquadringenquattuorquinquagintillion PRIMALposition=01 PRIMALlength=32 'domilliaquadringenquattuorquinqu'
// Needle: boom-boom                                  PRIMALposition=02 PRIMALlength=08 'oom-boom'
// Needle: vvvvv                                       PRIMALposition=01 PRIMALlength=04 'vvvv'
// Needle: 12345                                       PRIMALposition=01 PRIMALlength=05 '12345'
// Needle: likey-likey                                PRIMALposition=03 PRIMALlength=09 'key-likey'
// Needle: BOOOO00M                                    PRIMALposition=03 PRIMALlength=05 '0000M'
// Needle: aaaaaBOOOO00M                              PRIMALposition=02 PRIMALlength=09 'aaaaBOOO0'
// Needle: BOOOO0Maiaaa                               PRIMALposition=03 PRIMALlength=09 '0000Maiaa'
PRIMALlength=0;
for (i=0+(1); i < cbPattern-((4)-1)+(1)-(1); i++) { // -(1) because the last BB (Building-Block) order 4 has no counterpart(s)
    FoundAtPosition = cbPattern - ((4)-1) + 1;
    PRIMALpositionCANDIDATE=i;
    while ( PRIMALpositionCANDIDATE <= (FoundAtPosition-1) ) {
        j = PRIMALpositionCANDIDATE + 1;
        while ( j <= (FoundAtPosition-1) ) {
            if ( *(uint32_t *) (pbPattern+PRIMALpositionCANDIDATE-(1)) == *(uint32_t *) (pbPattern+j-(1)) ) FoundAtPosition = j;
            j++;
        }
        PRIMALpositionCANDIDATE++;
    }
    PRIMALlengthCANDIDATE = (FoundAtPosition-1)-i+1+((4)-1);
    if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=i; PRIMALlength = PRIMALlengthCANDIDATE;}
    if (cbPattern-i+1 <= PRIMALlength) break;
    if (PRIMALlength > 128) break; // Bail Out for 129[+]
}
// Swampwalker_BAILOUT heuristic order 4 (Needle should be bigger than 4) ]

// Swampwalker_BAILOUT heuristic order 2 (Needle should be bigger than 2) [
// Needle: 1234567890qwertyuiopasdfghjklzxcv          PRIMALposition=01 PRIMALlength=33 '1234567890qwertyuiopasdfghjklzxcv'
// Needle: vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv    PRIMALposition=31 PRIMALlength=02 'vv'
// Needle: vvvvvvvvvvBOOMSHAKALAKAvvvvvvvvvvv         PRIMALposition=09 PRIMALlength=13 'vvBOOMSHAKALA'
// Needle: Trollland                                   PRIMALposition=05 PRIMALlength=05 'lland'
// Needle: Swampwalker                                 PRIMALposition=03 PRIMALlength=09 'ampwalker'
// Needle: licenselessness                             PRIMALposition=01 PRIMALlength=13 'licenselesne'
// Needle: alfalfa                                     PRIMALposition=04 PRIMALlength=04 'alfa'
// Needle: Sandokan                                  PRIMALposition=01 PRIMALlength=07 'Sandoka'
// Needle: shazamish                                  PRIMALposition=02 PRIMALlength=08 'hazamish'
// Needle: Simplicius Simplicissimus                  PRIMALposition=08 PRIMALlength=15 'ius Simplicissi'
// Needle: domilliaquadringenquattuorquinquagintillion PRIMALposition=01 PRIMALlength=19 'domilliaquadrinq'
// Needle: DODO                                         PRIMALposition=02 PRIMALlength=03 'ODO'
// Needle: DODOD                                        PRIMALposition=03 PRIMALlength=03 'DOD'
// Needle: aaadODO                                      PRIMALposition=02 PRIMALlength=05 'aadOD'
// Needle: aaadODOD                                    PRIMALposition=02 PRIMALlength=05 'aadOD'
// Needle: DODOaaa                                     PRIMALposition=02 PRIMALlength=05 'ODOaa'
// Needle: DODOdaaa                                   PRIMALposition=03 PRIMALlength=05 'DODaa'
/*
PRIMALlength=0;
for (i=0+(1); i < cbPattern-2+1+(1)-(1); i++) { // -(1) because the last BB order 2 has no counterpart(s)
    FoundAtPosition = cbPattern;
    PRIMALpositionCANDIDATE=i;
    while ( PRIMALpositionCANDIDATE <= (FoundAtPosition-1) ) {
        j = PRIMALpositionCANDIDATE + 1;
        while ( j <= (FoundAtPosition-1) ) {
            if ( *(unsigned short *) (pbPattern+PRIMALpositionCANDIDATE-(1)) == *(unsigned short *) (pbPattern+j-(1)) ) FoundAtPosition = j;
            j++;
        }
    }
}

```

```

    PRIMALpositionCANDIDATE++;
}
PRIMALlengthCANDIDATE = (FoundAtPosition-1)-i+(2);
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=i; PRIMALlength = PRIMALlengthCANDIDATE;}
}
*/
// Swampwalker_BAILOUT heuristic order 2 (Needle should be bigger than 2) ]

/*
Legend:
'[]' points to BB forming left or right boundary;
'{}' points to BB being searched for;
'()' position of duplicate and new right boundary;

00000000011111111222222222333
12345678901234567890123456789012
Example #1 for Needle: 1234567890qwertyuiopasdfghjklzxcv NewNeedle = '1234567890qwertyuiopasdfghjklzxcv'
Example #2 for Needle: vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv NewNeedle = 'vv'
Example #3 for Needle: vvvvvvvvvvBOOMSHAKALAKAvvvvvvvvv NewNeedle = 'vvBOOMSHAKALA'

PRIMALlength=00; FoundAtPosition=33;
Step 01_00: {[12]34567890qwertyuiopasdfghjklzxc[v?]} ! For position #01 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=01, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
Step 01_01: {[12]}34567890qwertyuiopasdfghjklzxc[v?]} ! Searching for '12', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
Step 01_02: {[12]3}4567890qwertyuiopasdfghjklzxc[v?]} ! Searching for '23', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
...
Step 01_30: [12]34567890qwertyuiopasdfghjklzxc[v?]} ! Searching for 'zx', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
Step 01_31: [12]34567890qwertyuiopasdfghjklzxc[v?]} ! Searching for 'xc', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
Step 02_00: {[123]4567890qwertyuiopasdfghjklzxc[v?]} ! For position #02 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=02, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
Step 02_01: [1[23]4567890qwertyuiopasdfghjklzxc[v?]} ! Searching for '23', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
Step 02_02: 1[2[3]4]567890qwertyuiopasdfghjklzxc[v?]} ! Searching for '34', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
...
Step 02_29: 1[23]4567890qwertyuiopasdfghjklzxc[v?]} ! Searching for 'zx', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
Step 02_30: 1[23]4567890qwertyuiopasdfghjklzxc[v?]} ! Searching for 'xc', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
...
Step 31_00: {[1234567890qwertyuiopasdfghjklzxc[v?]} ! For position #31 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=31, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-31+(2)=03 !
Step 31_01: 1234567890qwertyuiopasdfghjklzxc[v?]} ! Searching for 'xc', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-31+(2)=03 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
Result:
PRIMALposition=01 PRIMALlength=33, NewNeedle = '1234567890qwertyuiopasdfghjklzxcv'

PRIMALlength=00; FoundAtPosition=33;
Step 01_00: {[v]vvvvvvvvvvvvvvvvvvvvvvvvvvvv[v?]} ! For position #01 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=01, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
Step 01_01: {[v(v)]vvvvvvvvvvvvvvvvvvvvvvvvvvvv} ! Searching for 'vv', FoundAtPosition = 02, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(02-1)-01+(2)=02 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
Step 02_00: {[v[v]vvvvvvvvvvvvvvvvvvvvvvvvvvvv[v?]} ! For position #02 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=02, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
Step 02_01: {[v(v)]v[v]vvvvvvvvvvvvvvvvvvvvvvvvvv} ! Searching for 'vv', FoundAtPosition = 03, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(03-1)-02+(2)=02 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
...
Step 31_00: {[vvvvvvvvvvvvvvvvvvvvvvvvvvvv[vv]]v[v?]} ! For position #31 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=31, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-31+(2)=03 !
Step 31_01: vvvvvvvvvvvvvvvvvvvvvvvvvvvvv[v(v)]v ! Searching for 'vv', FoundAtPosition = 32, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(32-1)-31+(2)=02 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
Result:
PRIMALposition=31 PRIMALlength=02, NewNeedle = 'vv'

PRIMALlength=00; FoundAtPosition=33;
Step 01_00: {[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv[v?]} ! For position #01 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=01, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
Step 01_01: {[v(v)]v[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'vv', FoundAtPosition = 02, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(02-1)-01+(2)=02 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
Step 02_00: {[v[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv[v?]} ! For position #02 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=02, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
Step 02_01: v[v(v)]v[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'vv', FoundAtPosition = 03, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(03-1)-02+(2)=02 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
...
Step 09_00: {[vvvvvvvv[v]BOOMSHAKALAKAvvvvvvvvv[v?]} ! For position #09 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=09, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-09+(2)=25 !
Step 09_01: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'vv', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16 !
Step 09_02: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'vB', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16 !
Step 09_03: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'BO', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16 !
Step 09_04: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'OO', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16 !
Step 09_05: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'OM', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16 !
Step 09_06: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'MS', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16 !
Step 09_07: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'SH', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16 !
Step 09_08: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'HA', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16 !
Step 09_09: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'AK', FoundAtPosition = 21, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(21-1)-09+(2)=13 !
Step 09_10: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'KA', FoundAtPosition = 21, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(21-1)-09+(2)=13 !
Step 09_11: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'AL', FoundAtPosition = 21, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(21-1)-09+(2)=13 !
Step 09_12: vvvvvvvv[v]vvvvvvvvvBOOMSHAKALAKAvvvvvvvvv} ! Searching for 'LA', FoundAtPosition = 21, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(21-1)-09+(2)=13 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
...
Step 31_00: {[vvvvvvvv[v]BOOMSHAKALAKAvvvvvvvvv[v?]} ! For position #31 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=31, RightBoundary=FoundAtPosition-1,
the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-31+(2)=03 !
Step 31_01: vvvvvvvvvvBOOMSHAKALAKAvvvvvvvvv[v(v)]v ! Searching for 'vv', FoundAtPosition = 32, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(32-1)-31+(2)=02 !
if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
Result:
PRIMALposition=09 PRIMALlength=13, NewNeedle = 'vvBOOMSHAKALA'
*/

// Here we have 4 or bigger NewNeedle, apply order 2 for pbPattern[i+(PRIMALposition-1)] with length 'PRIMALlength' and compare the pbPattern[i] with length 'cbPattern':
Listing: Railgun_Troldom, copyleft 2016-Aug-19, http://www.sanmayce.com/Railgun/

```

```
PRIMALlengthCANDIDATE = cbPattern;
cbPattern = PRIMALlength;
pbPattern = pbPattern + (PRIMALposition-1);

// Revision 2 commented section [
/*
if (cbPattern-1 <= 255) {
// BMH Order 2 [
    ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
    for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]= cbPattern-1;} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
    for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=i; // Rightmost appearance/position is needed
    i=0;
    while (i <= cbTarget-cbPattern) {
        Gulliver = bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1]];
        if ( Gulliver != cbPattern-1 ) { // CASE #2: if equal means the pair (char order 2) is not found i.e. Gulliver remains intact, skip the whole pattern and fall back (Order-1) chars i.e. one char for Order 2
            if ( Gulliver == cbPattern-2 ) { // CASE #1: means the pair (char order 2) is found
                if ( *(uint32_t *)&pbTarget[i] == ulHashPattern ) {
                    count = cbPattern-4+1;
                    while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i+(count-1)) )
                        count = count-4;
                }
                // If we miss to hit then no need to compare the original: Needle
                if ( count <= 0 ) {
                    // I have to add out-of-range checks...
                    // i-(PRIMALposition-1) >= 0
                    // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
                    // i-(PRIMALposition-1)+(count-1) >= 0
                    // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4

                    // "FIX" from 2014-Apr-27:
                    // Because (count-1) is negative, above fours are reduced to next twos:
                    // i-(PRIMALposition-1)+(count-1) >= 0
                    // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
                    // The line below is BUGGY:
                    //if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
                    // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
                    //if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
                    // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
                    if ( ((signed int)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) {
                        if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *) (pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for remainder)
                            when going under 0 in loop below:
                                count = PRIMALlengthCANDIDATE-4+1;
                                while ( count > 0 && *(uint32_t *) (pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *) (&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
                                    count = count-4;
                                if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
                        }
                    }
                } else
                    Gulliver = 1;
            }
            Gulliver = cbPattern - Gulliver - 2; // CASE #3: the pair is found and not as suffix i.e. rightmost position
            i = i + Gulliver;
            //GlobalI++; // Comment it, it is only for stats.
        }
        return(NULL);
    }
}

// BMH Order 2 ]
} else {
    // BMH order 2, needle should be >=4:
    ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
    for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
    for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=i;
    i=0;
    while (i <= cbTarget-cbPattern) {
        Gulliver = 1; // 'Gulliver' is the skip
        if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1]] != 0 ) {
            if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i+cbPattern-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
                if ( *(uint32_t *)&pbTarget[i] == ulHashPattern ) { // This fast check ensures not missing a match (for remainder)
                    when going under 0 in loop below:
                        count = cbPattern-4+1;
                        while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i+(count-1)) )
                            count = count-4;
                }
                // If we miss to hit then no need to compare the original: Needle
                if ( count <= 0 ) {
                    // I have to add out-of-range checks...
                    // i-(PRIMALposition-1) >= 0
                    // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
                    // i-(PRIMALposition-1)+(count-1) >= 0
                    // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4

                    // "FIX" from 2014-Apr-27:
                    // Because (count-1) is negative, above fours are reduced to next twos:
                    // i-(PRIMALposition-1)+(count-1) >= 0
                    // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
                    // The line below is BUGGY:
                    //if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
                    // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
                    //if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
                    // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
                    if ( ((signed int)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) {
                        if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *) (pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for remainder)
                            when going under 0 in loop below:
                                count = PRIMALlengthCANDIDATE-4+1;
                                while ( count > 0 && *(uint32_t *) (pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *) (&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
                                    count = count-4;
                                if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
                        }
                    }
                }
            }
        }
    }
}
```

```

    }
    } else Gulliver = cbPattern-(2-1);
    i = i + Gulliver;
    //GlobalI++; // Comment it, it is only for stats.
}
return(NULL);
}
*/
// Revision 2 commented section ]

if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {

    // BMH order 2, needle should be >=4:
    ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
    for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
    // Above line is translated by Intel as:

// 0044c 41 b8 00 00 01
// 00      mov r8d, 65536
// 00452 44 89 5c 24 20  mov DWORD PTR [32+rsp], r11d
// 00457 44 89 54 24 60  mov DWORD PTR [96+rsp], r10d
// 0045c e8 fc ff ff ff  call _intel_fast_memset
    for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=1;
    i=0;
    while (i <= cbTarget-cbPattern) {
        Gulliver = 1; // 'Gulliver' is the skip
        if ( bm_Horspool_Order2[(unsigned short *) &pbTarget[i+cbPattern-1-1]] != 0 ) {
            if ( bm_Horspool_Order2[(unsigned short *) &pbTarget[i+cbPattern-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
                if ( *(uint32_t *) &pbTarget[i] == ulHashPattern ) { // This fast check ensures not missing a match (for remainder)
                    when going under 0 in loop below:

                        count = cbPattern-4+1;
                        while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i+(count-1)]) )
                            count = count-4;

                    if (cbPattern != PRIMALlengthCANDIDATE) { // No need of same comparison when Needle and NewNeedle are equal!
                        // If we miss to hit then no need to compare the original: Needle
                        if ( count <= 0 ) {
                            // I have to add out-of-range checks...
                            // i-(PRIMALposition-1) >= 0
                            // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
                            // i-(PRIMALposition-1)+(count-1) >= 0
                            // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4

                        // "FIX" from 2014-Apr-27:
                        // Because (count-1) is negative, above fours are reduced to next twos:
                        // i-(PRIMALposition-1)+(count-1) >= 0
                        // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
                        // The line below is BUGGY:
                        // if ( (i-(PRIMALposition-1)) >= 0 && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
                        // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
                        // if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
                        // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
                        // if ( ((signed int)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+(PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4 ) {
                        // if ( *(uint32_t *) &pbTarget[i-(PRIMALposition-1)] == *(uint32_t *) (pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for remainder)
                        when going under 0 in loop below:
                            count = PRIMALlengthCANDIDATE-4+1;
                            while ( count > 0 && *(uint32_t *) (pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *) (&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
                                count = count-4;
                            if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
                        }
                    }
                }
            } else Gulliver = cbPattern-(2-1);
            i = i + Gulliver;
            //GlobalI++; // Comment it, it is only for stats.
        }
        return(NULL);
    } else { // if ( cbPattern<=NeedleThreshold2vs4swampLITE )

        // BMH pseudo-order 4, needle should be >=8+2:
        ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
        for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
        // In line below we "hash" 4bytes to 2bytes i.e. 16bit table, how to compute TOTAL number of BBs, 'cbPattern - Order + 1' is the number of BBs for text
        // 'cbPattern' bytes long, for example, for cbPattern=11 'fastest fox' and Order=4 we have BBs = 11-4+1=8:
        // "fast"
        // "aste"
        // "stes"
        // "test"
        // "est "
        // "st f"
        // "t fo"
        // " fox"
        // for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( *(unsigned short *) (pbPattern+i+0) + *(unsigned short *) (pbPattern+i+2) ) & ( (1<<16)-1 )]=1;
        // for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( *(uint32_t *) (pbPattern+i+0)>>16)+( *(uint32_t *) (pbPattern+i+0)&0xFFFF ) & ( (1<<16)-1 )]=1;
        // Above line is replaced by next one with better hashing:
        for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( *(uint32_t *) (pbPattern+i+0)>>(16-1))+( *(uint32_t *) (pbPattern+i+0)&0xFFFF ) & ( (1<<16)-1 )]=1;

        i=0;
        while (i <= cbTarget-cbPattern) {
            Gulliver = 1;
            // if ( bm_Horspool_Order2[( *(uint32_t *) &pbTarget[i+cbPattern-1-2]>>16)+( *(uint32_t *) &pbTarget[i+cbPattern-1-2]&0xFFFF ) & ( (1<<16)-1 )] != 0 ) { // DWORD #1
            // Above line is replaced by next one with better hashing:
            if ( bm_Horspool_Order2[( *(uint32_t *) &pbTarget[i+cbPattern-1-2]>>(16-1))+( *(uint32_t *) &pbTarget[i+cbPattern-1-2]&0xFFFF ) & ( (1<<16)-1 )] != 0 ) {

```



```

(1<<16)-1 )) != 0 ) { // DWORD #1
    //if ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )) == 0 ) Gulliver = cbPattern-(2-1)-2-4; else {
    // Above line is replaced in order to strengthen the skip by checking the middle DWORD, if the two DWORDs are 'ab' and 'cd' i.e. [2x][2a][2b][2c][2d] then the middle DWORD is 'bc'.
    // The respective offsets (backwards) are: -10/-8/-6/-4 for 'xa'/'ab'/'bc'/'cd'.
    //if ( ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>16)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) ) & ( (1<<16)-1 )) ) + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )) ) + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>16)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) ) & ( (1<<16)-1 )) ) < 3 ) Gulliver = cbPattern-(2-1)-2-4-2; else {
    // Above line is replaced by next one with better hashing:
    // when using (16-1) right shifting instead of 16 we will have two different pairs (if they are equal), the highest bit being lost do the job especially for ASCII texts with no symbols in range 128-255.
    // Example for genomesque pair TT+TT being shifted by (16-1):
    // T      = 01010100
    // TT     = 01010100 01010100
    // TTT    = 01010100 01010100 01010100
    // TTTT   = 00000000 00000000 01010100 01010100
    // TTTT>>16 = 00000000 00000000 01010100 01010100
    // TTTT>>(16-1) = 00000000 00000000 10101000 10101000 ---- Due to the left shift by 1, the 8th bits of 1st and 2nd bytes are populated - usually they are 0 for English texts & 'ACGT' data.
    //if ( ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>(16-1))+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) ) & ( (1<<16)-1 )) ) + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>(16-1))+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )) ) + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>(16-1))+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) ) & ( (1<<16)-1 )) ) < 3 ) Gulliver = cbPattern-(2-1)-2-4-2; else {
    // 'Maximus' uses branched 'if', again.
    if ( \
        ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6 +1]>>(16-1))+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6 +1]&0xFFFF) ) & ( (1<<16)-1 )) ) == 0 \
        || ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4 +1]>>(16-1))+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4 +1]&0xFFFF) ) & ( (1<<16)-1 )) ) == 0 \
        ) Gulliver = cbPattern-(2-1)-2-4-2 +1; else {
    // Above line is not optimized (several a SHR are used), we have 5 non-overlapping WORDs, or 3 overlapping WORDs, within 4 overlapping DWORDs so:
    // [2x][2a][2b][2c][2d]
    // DWORD #4
    // [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>16) = !SHR to be avoided! <--
    // [2x] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) = -----
    // DWORD #3
    // [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16) = !SHR to be avoided! <--
    // [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = -----
    // DWORD #2
    // [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>16) = !SHR to be avoided! <--
    // [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = -----
    // DWORD #1
    // [2d] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]>>16) = -----
    // [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = -----
    // So in order to remove 3 SHR instructions the equal extractions are:
    // DWORD #4
    // [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = !SHR to be avoided! <--
    // [2x] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) = -----
    // DWORD #3
    // [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = !SHR to be avoided! <--
    // [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = -----
    // DWORD #2
    // [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = !SHR to be avoided! <--
    // [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = -----
    // DWORD #1
    // [2d] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]>>16) = -----
    // [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = -----
    //if ( ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) ) & ( (1<<16)-1 )) ) + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )) ) + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) ) & ( (1<<16)-1 )) ) < 3 ) Gulliver = cbPattern-(2-1)-2-6; else {
    // Since the above Decumanus mumbo-jumbo (3 overlapping lookups vs 2 non-overlapping lookups) is not fast enough we go DuoDecumanus or 3x4:
    // [2y][2x][2a][2b][2c][2d]
    // DWORD #3
    // DWORD #2
    // DWORD #1
    //if ( ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )) ) + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-8]>>16)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-8]&0xFFFF) ) & ( (1<<16)-1 )) ) < 2 ) Gulliver = cbPattern-(2-1)-2-8; else {
    if ( *(uint32_t *)&pbTarget[i] == u1HashPattern ) {
        // Order 4 [
        // Let's try something "outrageous" like comparing with[out] overlap BBs 4bytes long instead of 1 byte back-to-back:
        // Inhere we are using order 4, 'cbPattern - Order + 1' is the number of BBs for text 'cbPattern' bytes long, for example, for cbPattern=11 'fastest fox' and Order=4 we have BBs = 11-4+1=8:
        //0:"fast" if the comparison failed here, 'count' is 1; 'Gulliver' is cbPattern-(4-1)-7
        //1:"aste" if the comparison failed here, 'count' is 2; 'Gulliver' is cbPattern-(4-1)-6
        //2:"stes" if the comparison failed here, 'count' is 3; 'Gulliver' is cbPattern-(4-1)-5
        //3:"test" if the comparison failed here, 'count' is 4; 'Gulliver' is cbPattern-(4-1)-4
        //4:"est " if the comparison failed here, 'count' is 5; 'Gulliver' is cbPattern-(4-1)-3
        //5:"st f" if the comparison failed here, 'count' is 6; 'Gulliver' is cbPattern-(4-1)-2
        //6:"t fo" if the comparison failed here, 'count' is 7; 'Gulliver' is cbPattern-(4-1)-1
        //7:" fox" if the comparison failed here, 'count' is 8; 'Gulliver' is cbPattern-(4-1)
        count = cbPattern-4+1;
        // Below comparison is UNIDIRECTIONAL:
        while ( count > 0 && *(uint32_t *)&pbPattern+count-1 == *(uint32_t *)&pbTarget[i+(count-1)) )
            count = count-4;
    }
    if (cbPattern != PRIMALlengthCANDIDATE) { // No need of same comparison when Needle and NewNeedle are equal!
    // count = cbPattern-4+1 = 23-4+1 = 20
    // boomshakalakaZZZZZZZZZZ 20
    // boomshakalakaZZZZZZZZZZ 20-4
    // boomshakalakaZZZZZZZZZZ 20-8 = 12
    // boomshakalakaZZZZZZZZZZ 20-12 = 8
    // boomshakalakaZZZZZZZZZZ 20-16 = 4
    // If we miss to hit then no need to compare the original: Needle
    if ( count <= 0 ) {

```

```

// I have to add out-of-range checks...
// i-(PRIMALposition-1) >= 0
// &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
// i-(PRIMALposition-1)+(count-1) >= 0
// &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4

// "FIX" from 2014-Apr-27:
// Because (count-1) is negative, above fours are reduced to next twos:
// i-(PRIMALposition-1)+(count-1) >= 0
// &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
// The line below is BUGGY:
// if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
// The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
// if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
// FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
// if ( ((signed int)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+(PRIMALLengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) {
// if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(&pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for remainder)
// when going under 0 in loop below:
//     count = PRIMALLengthCANDIDATE-4+1;
//     while ( count > 0 && *(uint32_t *)(&pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
//         count = count-4;
//     if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
// }
// }
// } else { //if (cbPattern != PRIMALLengthCANDIDATE)
//
//     if ( count <= 0 ) return(pbTarget+i);
//
// }
//
// left AT THE SAME TIME.
//
// Below comparison is Bidirectional. It pays off when needle is 8+++ long:
// for (count = cbPattern-4+1; count > 0; count = count-4) {
//     if ( *(uint32_t *)(&pbPattern+count-1) != *(uint32_t *)(&pbTarget[i]+(count-1)) ) {break;}
//     if ( *(uint32_t *)(&pbPattern+(cbPattern-4+1)-count) != *(uint32_t *)(&pbTarget[i]+(cbPattern-
// 4+1)-count) ) {count = (cbPattern-4+1)-count +(1); break;} // +(1) because two lookups are implemented as one, also no danger of 'count' being 0 because of the fast
// check outwith the 'while': if ( *(uint32_t *)&pbTarget[i] == ulHashPattern)
//     if ( count <= 0 ) return(pbTarget+i);
//     // Checking the order 2 pairs in mismatched DWORD, all the 3:
//     //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1]] == 0 ) Gulliver = count; //
//     //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1]] == 0 ) Gulliver = count+1;
//     //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1+1]] == 0 ) Gulliver =
//     //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1]] +
// bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1]] + bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1+1]] < 3 ) Gulliver = count; // 1 or bigger,
// as it should, THE MIN(count,count+1,count+1+1)
//     // Above compound 'if' guarantees not that Gulliver > 1, an example:
//     // Needle:   fastest tax
//     // Window: ...fastest tax...
//     // After matching 'tax' vs 'tax' and 'fast' vs 'fast' the mismatched DWORD is 'test' vs
// 'tast':
//     // 'tast' when factorized down to order 2 yields: 'ta','as','st' - all the three when summed
// give 1+1+1=3 i.e. Gulliver remains 1.
//     // Roughly speaking, this attempt maybe has its place in worst-case scenarios but not in
// English text and even not in ACGT data, that's why I commented it in original 'Shockeroo'.
//     //if ( bm_Horspool_Order2[ ( *(uint32_t *)&pbTarget[i+count-1]>>16)+(*(uint32_t
// *)&pbTarget[i+count-1]&0xFFFF) ) & ( (1<<16)-1) ] == 0 ) Gulliver = count; // 1 or bigger, as it should
//     // Above line is replaced by next one with better hashing:
//     //if ( bm_Horspool_Order2[ ( *(uint32_t *)&pbTarget[i+count-1]>>(16-1))+*(uint32_t
// *)&pbTarget[i+count-1]&0xFFFF) ) & ( (1<<16)-1) ] == 0 ) Gulliver = count; // 1 or bigger, as it should
//     // Order 4 ]
// }
// }
// } else Gulliver = cbPattern-(2-1)-2; // -2 because we check the 4 rightmost bytes not 2.
// i = i + Gulliver;
// //Globali++; // Comment it, it is only for stats.
// }
// return(NULL);
//
// } // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
// } // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
// } //if ( cbPattern<4 )
// }
//
// For short needles, and mainly haystacks, 'Doublet' is quite effective. Consider it or 'Quadruplet'.
// Fixed version from 2012-Feb-27.
// Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
char * Railgun_Doublet (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern)
{
    char * pbTargetMax = pbTarget + cbTarget;
    register uint32_t ulHashPattern;
    uint32_t ulHashTarget, count, countSTATIC;

    if (cbPattern > cbTarget) return(NULL);

    countSTATIC = cbPattern-2;

    pbTarget = pbTarget+cbPattern;
    ulHashPattern = (*(uint16_t *)(&pbPattern));

    for ( ;; ) {
        if ( ulHashPattern == (*(uint16_t *)(&pbTarget-cbPattern)) ) {
            count = countSTATIC;
            while ( count && *(char *)(&pbPattern+2+(countSTATIC-count)) == *(char *)(&pbTarget-cbPattern+2+(countSTATIC-count)) ) {
                count--;
            }
            if ( count == 0 ) return((pbTarget-cbPattern));
        }
    }
}

```

```

    }
    pbTarget++;
    if (pbTarget > pbTargetMax) return(NULL);
}
}
*/
/*
; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140";
; mark_description "-O3 -QxSSE2 -D_N_XMM -D_N_prefetch_4096 -D_N_Branchfull -D_N_HIGH_PRIORITY -FA";

_TEXT SEGMENT 'CODE'
; COMDAT Railgun_Trolldom
; mark_begin;
ALIGN 16
PUBLIC Railgun_Trolldom
Railgun_Trolldom PROC
; parameter 1: rcx
; parameter 2: rdx
; parameter 3: r8d
; parameter 4: r9d
.B11.1:: ; Preds .B11.0
    push rbx ;3712.1
    push rsi ;3712.1
    push rdi ;3712.1
    push r12 ;3712.1
    push r13 ;3712.1
    push r14 ;3712.1
    push r15 ;3712.1
    push rbp ;3712.1
    mov eax, 65640 ;3712.1
    call __chkstk ;3712.1
    sub rsp, 65640 ;3712.1
    mov ebx, r8d ;3712.1
    mov r13d, ebx ;3713.23
    mov rdi, rcx ;3712.1
    mov esi, r9d ;3712.1
    mov r12, rdx ;3712.1
    cmp esi, ebx ;3738.18
    lea r11, QWORD PTR [r13+rdi] ;3713.23
    ja .B11.9 ;3738.18
    ; Prob 28%
    ; LOE rsi rdi r11 r12 r13 ebx r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B11.1
.B11.2::
    cmp esi, 4 ;3740.17
    jae .B11.18 ;3740.17
    ; Prob 50%
    ; LOE rsi rdi r11 r12 r13 ebx r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B11.2
.B11.3::
    movsx edx, BYTE PTR [r12] ;3744.23
    lea ebp, DWORD PTR [-1+rsi] ;3744.74
    shl edx, 8 ;3744.45
    lea rbx, QWORD PTR [rdi+rsi] ;3743.21
    lea rax, QWORD PTR [-2+rsi+rdi] ;
    lea rcx, QWORD PTR [-3+rsi+rdi] ;
    movsx edi, BYTE PTR [r12+rbp] ;3744.53
    add edx, edi ;3744.53
    mov r8d, edx ;3750.32
    shr r8d, 8 ;3750.32
    movsx rbp, r8b ;3750.32
    cmp esi, 3 ;3745.19
    je .B11.11 ;3745.19
    ; Prob 50%
    ; LOE rax rcx rbx r11 r12 edx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B11.3 .B11.8
.B11.5::
    movsx ecx, BYTE PTR [rax] ;3760.48
    shl ecx, 8 ;3760.53
    movsx esi, BYTE PTR [-1+rbx] ;3760.70
    add ecx, esi ;3760.70
    cmp edx, ecx ;3760.70
    je .B11.41 ;3760.70
    ; Prob 20%
    ; LOE rax rbx r11 edx ebp esi xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B11.5
.B11.6::
    cmp ebp, esi ;3761.48
    je .B11.8 ;3761.48
    ; Prob 50%
    ; LOE rax rbx r11 edx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B11.6
.B11.7::
    inc rax ;3761.53
    inc rbx ;3761.53
    ; LOE rax rbx r11 edx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B11.7 .B11.6
.B11.8::
    inc rbx ;3762.4
    inc rax ;3762.4
    cmp rbx, r11 ;3763.19
    jbe .B11.5 ;3763.19
    ; Prob 80%
    ; LOE rax rbx r11 edx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
    ; Preds .B11.1 .B11.8
.B11.9::
    xor eax, eax ;3763.38
    add rsp, 65640 ;3763.38
    pop rbp ;3763.38
    pop r15 ;3763.38
    pop r14 ;3763.38
    pop r13 ;3763.38
    pop r12 ;3763.38
    pop rdi ;3763.38
    pop rsi ;3763.38
    pop rbx ;3763.38
    ret ;3763.38
    ; LOE
    ; Preds .B11.3 .B11.16
.B11.11::
    movsx edi, BYTE PTR [rcx] ;3747.49
    shl edi, 8 ;3747.54
    movsx esi, BYTE PTR [-1+rbx] ;3747.71
    add edi, esi ;3747.71

```

```

movsx r8d, BYTE PTR [-2+rbx] ;3748.56
cmp edx, edi ;3747.71
jne .B11.13 ; Prob 50% ;3747.71
; LOE rcx rbx r11 r12 edx ebp r8d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.11
.B11.12::
cmp r8b, BYTE PTR [1+r12] ;3748.56
je .B11.185 ; Prob 20% ;3748.56
; LOE rcx rbx r11 r12 edx ebp r8d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.11 .B11.12
.B11.13::
cmp ebp, r8d ;3750.49
je .B11.16 ; Prob 50% ;3750.49
; LOE rcx rbx r11 r12 edx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.13
.B11.14::
inc rbx ;3751.6
inc rcx ;3751.6
cmp bpl, BYTE PTR [-2+rbx] ;3752.50
je .B11.16 ; Prob 50% ;3752.50
; LOE rcx rbx r11 r12 edx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.14
.B11.15::
inc rcx ;3752.55
inc rbx ;3752.55
; LOE rcx rbx r11 r12 edx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.15 .B11.14 .B11.13
.B11.16::
inc rbx ;3754.5
inc rcx ;3754.5
cmp rbx, r11 ;3755.20
jbe .B11.11 ; Prob 80% ;3755.20
; LOE rcx rbx r11 r12 edx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.16
.B11.17::
xor eax, eax ;3755.39
add rsp, 65640 ;3755.39
pop rbp ;3755.39
pop r15 ;3755.39
pop r14 ;3755.39
pop r13 ;3755.39
pop r12 ;3755.39
pop rdi ;3755.39
pop rsi ;3755.39
pop rbx ;3755.39
ret ;3755.39
; LOE
; Preds .B11.2
.B11.18::
cmp esi, 19 ;3766.19
ja .B11.82 ; Prob 50% ;3766.19
; LOE rsi rdi r11 r12 r13 ebx r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.18
.B11.19::
cmp ebx, 777 ;3774.14
jb .B11.67 ; Prob 50% ;3774.14
; LOE rsi rdi r11 r12 ebx xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.19
.B11.20::
mov ebp, DWORD PTR [r12] ;3825.33
cmp ebx, 77777 ;3819.21
jae .B11.45 ; Prob 50% ;3819.21
; LOE rsi rdi r12 ebx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.20
.B11.21::
xor edx, edx ;3827.38
lea rcx, QWORD PTR [32+rsp] ;3827.38
mov r8d, 8192 ;3827.38
call _intel_fast_memset ;3827.38
; LOE rsi rdi r12 ebx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.21
.B11.22::
mov r10d, esi ;3829.30
dec r10d ;3829.30
je .B11.31 ; Prob 50% ;3829.30
; LOE rsi rdi r12 ebx ebp r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.22
.B11.23::
mov r9d, r10d ;3829.4
xor r8d, r8d ;3829.4
shr r9d, 4 ;3829.4
mov eax, 1 ;3829.4
xor edx, edx ;
test r9d, r9d ;3829.4
jbe .B11.27 ; Prob 15% ;3829.4
; LOE rdx rsi rdi r12 eax ebx ebp r8d r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.23 .B11.25
.B11.25::
mov r11d, 1 ;3829.211
mov r15d, 1 ;3829.211
lea r13d, DWORD PTR [1+rdx] ;3829.197
inc r8d ;3829.4
movzx ecx, WORD PTR [rdx+r12] ;3829.197
mov eax, ecx ;3829.166
and ecx, 7 ;3829.211
shl r11d, cl ;3829.211
movzx ecx, WORD PTR [r13+r12] ;3829.151
mov r14d, ecx ;3829.166
and ecx, 7 ;3829.211
lea r13d, DWORD PTR [3+rdx] ;3829.197
shl r15d, cl ;3829.211
shr eax, 3 ;3829.166
lea ecx, DWORD PTR [2+rdx] ;3829.197
shr r14d, 3 ;3829.166
movzx ecx, WORD PTR [rcx+r12] ;3829.151
or BYTE PTR [32+rsp+rax], r11b ;3829.211
mov eax, ecx ;3829.166
shr eax, 3 ;3829.166
and ecx, 7 ;3829.211
mov r11d, 1 ;3829.211
shl r11d, cl ;3829.211
or BYTE PTR [32+rsp+r14], r15b ;3829.211
lea r14d, DWORD PTR [4+rdx] ;3829.197

```

```

movzx ecx, WORD PTR [r13+r12] ;3829.151
or     BYTE PTR [32+rsp+rax], r11b ;3829.211
mov    eax, ecx ;3829.166
shr    eax, 3 ;3829.166
and    ecx, 7 ;3829.211
mov    r11d, 1 ;3829.211
shl    r11d, cl ;3829.211
or     BYTE PTR [32+rsp+rax], r11b ;3829.211
lea    r11d, DWORD PTR [5+rdx] ;3829.197
mov    eax, 1 ;3829.211
movzx  ecx, WORD PTR [r14+r12] ;3829.151
mov    r15d, ecx ;3829.166
and    ecx, 7 ;3829.211
mov    r14d, 1 ;3829.211
shl    eax, cl ;3829.211
movzx  ecx, WORD PTR [r11+r12] ;3829.151
mov    r13d, ecx ;3829.166
and    ecx, 7 ;3829.211
mov    r11d, 1 ;3829.211
shl    r14d, cl ;3829.211
shr    r15d, 3 ;3829.166
lea    ecx, DWORD PTR [6+rdx] ;3829.197
shr    r13d, 3 ;3829.166
movzx  ecx, WORD PTR [rcx+r12] ;3829.151
or     BYTE PTR [32+rsp+r15], al ;3829.211
mov    eax, ecx ;3829.166
shr    eax, 3 ;3829.166
lea    r15d, DWORD PTR [7+rdx] ;3829.197
and    ecx, 7 ;3829.211
shl    r11d, cl ;3829.211
or     BYTE PTR [32+rsp+r13], r14b ;3829.211
lea    r13d, DWORD PTR [8+rdx] ;3829.197
movzx  ecx, WORD PTR [r15+r12] ;3829.151
mov    r15d, 1 ;3829.211
or     BYTE PTR [32+rsp+rax], r11b ;3829.211
mov    eax, ecx ;3829.166
shr    eax, 3 ;3829.166
and    ecx, 7 ;3829.211
mov    r11d, 1 ;3829.211
shl    r11d, cl ;3829.211
or     BYTE PTR [32+rsp+rax], r11b ;3829.211
lea    r11d, DWORD PTR [9+rdx] ;3829.197
mov    eax, 1 ;3829.211
movzx  ecx, WORD PTR [r13+r12] ;3829.151
mov    r14d, ecx ;3829.166
and    ecx, 7 ;3829.211
shl    eax, cl ;3829.211
movzx  ecx, WORD PTR [r11+r12] ;3829.151
mov    r13d, ecx ;3829.166
and    ecx, 7 ;3829.211
mov    r11d, 1 ;3829.211
shl    r15d, cl ;3829.211
shr    r14d, 3 ;3829.166
lea    ecx, DWORD PTR [10+rdx] ;3829.197
shr    r13d, 3 ;3829.166
movzx  ecx, WORD PTR [rcx+r12] ;3829.151
or     BYTE PTR [32+rsp+r14], al ;3829.211
mov    eax, ecx ;3829.166
shr    eax, 3 ;3829.166
lea    r14d, DWORD PTR [11+rdx] ;3829.197
and    ecx, 7 ;3829.211
shl    r11d, cl ;3829.211
or     BYTE PTR [32+rsp+r13], r15b ;3829.211
lea    r13d, DWORD PTR [12+rdx] ;3829.197
movzx  ecx, WORD PTR [r14+r12] ;3829.151
mov    r14d, 1 ;3829.211
or     BYTE PTR [32+rsp+rax], r11b ;3829.211
mov    eax, ecx ;3829.166
shr    eax, 3 ;3829.166
and    ecx, 7 ;3829.211
mov    r11d, 1 ;3829.211
shl    r11d, cl ;3829.211
or     BYTE PTR [32+rsp+rax], r11b ;3829.211
lea    r11d, DWORD PTR [13+rdx] ;3829.197
mov    eax, 1 ;3829.211
movzx  ecx, WORD PTR [r13+r12] ;3829.151
mov    r15d, ecx ;3829.166
and    ecx, 7 ;3829.211
shl    eax, cl ;3829.211
movzx  ecx, WORD PTR [r11+r12] ;3829.151
mov    r13d, ecx ;3829.166
and    ecx, 7 ;3829.211
mov    r11d, 1 ;3829.211
shl    r14d, cl ;3829.211
shr    r15d, 3 ;3829.166
lea    ecx, DWORD PTR [14+rdx] ;3829.197
shr    r13d, 3 ;3829.166
movzx  ecx, WORD PTR [rcx+r12] ;3829.151
or     BYTE PTR [32+rsp+r15], al ;3829.211
mov    eax, ecx ;3829.166
shr    eax, 3 ;3829.166
lea    r15d, DWORD PTR [15+rdx] ;3829.197
and    ecx, 7 ;3829.211
add    edx, 16 ;3829.4
shl    r11d, cl ;3829.211
or     BYTE PTR [32+rsp+r13], r14b ;3829.211
movzx  ecx, WORD PTR [r15+r12] ;3829.151
DB     144 ;3829.211
or     BYTE PTR [32+rsp+rax], r11b ;3829.211
mov    eax, ecx ;3829.166

```



```

shr     eax, 3                                ;3829.166
and     ecx, 7                                ;3829.211
mov     r11d, 1                               ;3829.211
shl     r11d, cl                               ;3829.211
or      BYTE PTR [32+rsp+rax], r11b           ;3829.211
cmp     r8d, r9d                             ;3829.4
jb      .B11.25                               ;3829.4
; Prob 99%
; LOE rdx rsi rdi r12 ebx ebp r8d r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.25
.B11.26::
shl     r8d, 4                                ;3829.38
lea     eax, DWORD PTR [1+r8]                 ;3829.4
; LOE rsi rdi r12 eax ebx ebp r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.26 .B11.23
.B11.27::
dec     eax                                    ;3829.38
mov     edx, eax                              ;3829.4
cmp     eax, r10d                             ;3829.4
jae     .B11.31                               ;3829.4
; Prob 15%
; LOE rdx rsi rdi r12 eax ebx ebp r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.27 .B11.29
.B11.29::
movzx   ecx, WORD PTR [r12+rdx]               ;3829.151
mov     edx, ecx                              ;3829.166
shr     edx, 3                                ;3829.166
and     ecx, 7                                ;3829.211
mov     r8d, 1                                ;3829.211
shl     r8d, cl                               ;3829.211
inc     eax                                    ;3829.4
or      BYTE PTR [32+rsp+rdx], r8b           ;3829.211
mov     edx, eax                              ;3829.4
cmp     eax, r10d                             ;3829.4
jb      .B11.29                               ;3829.4
; Prob 93%
; LOE rdx rsi rdi r12 eax ebx ebp r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.22 .B11.29 .B11.27
.B11.31::
xor     edx, edx                              ;3830.4
lea     r8d, DWORD PTR [-3+rsi]               ;3836.192
movsxd  r9, r8d                               ;3838.8
lea     r10d, DWORD PTR [-1+rsi]              ;3844.33
sub     ebx, esi                              ;3831.16
; LOE rsi rdi r9 r12 edx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.31 .B11.43
.B11.32::
mov     r14d, 1                               ;3834.145
lea     eax, DWORD PTR [-2+rsi+rdx]           ;3834.141
movzx   ecx, WORD PTR [rax+rdi]               ;3834.59
mov     r11d, ecx                             ;3834.87
shr     r11d, 3                               ;3834.87
and     ecx, 7                                ;3834.145
shl     r14d, cl                               ;3834.145
movzx   r13d, BYTE PTR [32+rsp+r11]           ;3834.12
test    r13d, r14d                           ;3834.145
je      .B11.42                               ;3834.156
; Prob 50%
; LOE rsi rdi r9 r12 edx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.32
.B11.33::
mov     r14d, 1                               ;3836.150
lea     eax, DWORD PTR [-4+rsi+rdx]           ;3836.146
movzx   ecx, WORD PTR [rax+rdi]               ;3836.60
mov     r11d, ecx                             ;3836.90
shr     r11d, 3                               ;3836.90
and     ecx, 7                                ;3836.150
shl     r14d, cl                               ;3836.150
movzx   r13d, BYTE PTR [32+rsp+r11]           ;3836.13
test    r13d, r14d                           ;3836.150
jne     .B11.35                               ;3836.161
; Prob 50%
; LOE rsi rdi r9 r12 edx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.33
.B11.34::
mov     ecx, r8d                              ;3836.165
jmp     .B11.43                               ;3836.165
; Prob 100%
; LOE rsi rdi r9 r12 edx ecx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.33
.B11.35::
mov     eax, edx                              ;3837.26
mov     ecx, 1                                ;3832.5
rax, rdi                                     ;3837.26
cmp     ebp, DWORD PTR [rax]                  ;3837.41
jne     .B11.43                               ;3837.41
; Prob 50%
; LOE rax rsi rdi r9 r12 edx ecx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.35
.B11.36::
mov     r11, r9                               ;3838.8
test    r9, r9                               ;3839.24
jle     .B11.41                               ;3839.24
; Prob 2%
; LOE rax rsi rdi r9 r11 r12 edx ecx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.36 .B11.39
.B11.38::
mov     r13d, DWORD PTR [-1+r11+r12]           ;3839.59
cmp     r13d, DWORD PTR [-1+r11+rax]           ;3839.80
jne     .B11.43                               ;3839.80
; Prob 20%
; LOE rax rsi rdi r9 r11 r12 edx ecx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.38
.B11.39::
add     r11, -4                               ;3840.23
test    r11, r11                             ;3839.24
jg      .B11.38                               ;3839.24
; Prob 82%
; LOE rax rsi rdi r9 r11 r12 edx ecx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.5 .B11.36 .B11.39
.B11.41::
add     rsp, 65640                            ;3841.33
pop     rbp                                    ;3841.33
pop     r15                                    ;3841.33
pop     r14                                    ;3841.33
pop     r13                                    ;3841.33
pop     r12                                    ;3841.33
pop     rdi                                    ;3841.33
pop     rsi                                    ;3841.33
pop     rbx                                    ;3841.33
ret                                           ;3841.33
; LOE

```

```

.B11.42::      ; Preds .B11.32
mov     ecx, r10d      ;3844.12
          ; LOE rsi rdi r9 r12 edx ecx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
.B11.43::      ; Preds .B11.38 .B11.34 .B11.35 .B11.42
add     edx, ecx      ;3845.13
cmp     edx, ebx      ;3831.25
jbe     .B11.32      ; Prob 82%
          ; LOE rsi rdi r9 r12 edx ebx ebp r8d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.43
.B11.44::
xor     eax, eax      ;3848.10
add     rsp, 65640     ;3848.10
pop     rbp           ;3848.10
pop     r15           ;3848.10
pop     r14           ;3848.10
pop     r13           ;3848.10
pop     r12           ;3848.10
pop     rdi           ;3848.10
pop     rsi           ;3848.10
pop     rbx           ;3848.10
ret
          ; LOE
          ; Preds .B11.20
.B11.45::
xor     edx, edx      ;3853.33
lea     rcx, QWORD PTR [32+rsp] ;3853.33
mov     r8d, 65536     ;3853.33
call    _intel_fast_memset ;3853.33
          ; LOE rsi rdi r12 ebx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.45
.B11.46::
mov     r9d, esi      ;3854.28
dec     r9d           ;3854.28
je      .B11.53      ; Prob 50%
          ; LOE rsi rdi r12 ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.46
.B11.47::
mov     edx, r9d      ;3854.4
mov     ecx, 1        ;3854.4
shr     edx, 1        ;3854.4
xor     eax, eax      ;3854.4
test    edx, edx      ;3854.4
jbe     .B11.51      ; Prob 15%
          ; LOE rsi rdi r12 eax edx ecx ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.47
.B11.48::
mov     cl, 1         ;3854.36
          ; LOE rsi rdi r12 eax edx ebx ebp r9d cl xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.49 .B11.48
.B11.49::
lea     r8d, DWORD PTR [rax+rax] ;3854.75
lea     r11d, DWORD PTR [1+rax+rax] ;3854.75
inc     eax           ;3854.4
cmp     eax, edx      ;3854.4
movzx   r10d, WORD PTR [r8+r12] ;3854.75
movzx   r13d, WORD PTR [r11+r12] ;3854.75
mov     BYTE PTR [32+rsp+r10], cl ;3854.36
mov     BYTE PTR [32+rsp+r13], cl ;3854.36
jb      .B11.49      ; Prob 64%
          ; LOE rsi rdi r12 eax edx ebx ebp r9d cl xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.49
.B11.50::
lea     ecx, DWORD PTR [1+rax+rax] ;3854.4
          ; LOE rsi rdi r12 ecx ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.50 .B11.47
.B11.51::
dec     ecx           ;3854.36
cmp     ecx, r9d      ;3854.4
jae     .B11.53      ; Prob 15%
          ; LOE rcx rsi rdi r12 ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.51
.B11.52::
movzx   eax, WORD PTR [rcx+r12] ;3854.75
mov     BYTE PTR [32+rsp+rax], 1 ;3854.36
          ; LOE rsi rdi r12 ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.46 .B11.51 .B11.52
.B11.53::
xor     edx, edx      ;3855.4
lea     ecx, DWORD PTR [-3+rsi] ;3859.113
movsxd  r8, ecx       ;3861.8
sub     ebx, esi      ;3856.16
          ; LOE rsi rdi r8 r12 edx ecx ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.53 .B11.65
.B11.54::
lea     eax, DWORD PTR [-2+rsi+rdx] ;3858.72
movzx   r10d, WORD PTR [rax+rdi] ;3858.49
BYTE PTR [32+rsp+r10], 0 ;3858.79
cmp     .B11.64      ; Prob 50%
          ; LOE rsi rdi r8 r12 edx ecx ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.54
.B11.55::
lea     eax, DWORD PTR [-4+rsi+rdx] ;3859.75
movzx   r10d, WORD PTR [rax+rdi] ;3859.50
BYTE PTR [32+rsp+r10], 0 ;3859.82
cmp     .B11.57      ; Prob 50%
          ; LOE rsi rdi r8 r12 edx ecx ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.55
.B11.56::
mov     r10d, ecx     ;3859.86
jmp     .B11.65      ; Prob 100%
          ; LOE rsi rdi r8 r12 edx ecx ebx ebp r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.55
.B11.57::
mov     eax, edx      ;3860.26
mov     r10d, 1       ;3857.5
add     rax, rdi      ;3860.26
cmp     ebp, DWORD PTR [rax] ;3860.41
jne     .B11.65      ; Prob 50%
          ; LOE rax rsi rdi r8 r12 edx ecx ebx ebp r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
          ; Preds .B11.57
.B11.58::
mov     r11, r8       ;3861.8
test    r8, r8        ;3862.24
jle     .B11.63      ; Prob 2%
          ; LOE rax rsi rdi r8 r11 r12 edx ecx ebx ebp r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15

```

```

.B11.60::      ; Preds .B11.58 .B11.61
mov     r13d, DWORD PTR [-1+r11+r12] ;3862.59
cmp     r13d, DWORD PTR [-1+r11+rax] ;3862.80
jne     .B11.65 ; Prob 20% ;3862.80
; LOE rax rsi rdi r8 r11 r12 edx ecx ebx ebp r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.60
.B11.61::
add     r11, -4 ;3863.23
test    r11, r11 ;3862.24
jg      .B11.60 ; Prob 82% ;3862.24
; LOE rax rsi rdi r8 r11 r12 edx ecx ebx ebp r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.58 .B11.61
.B11.63::
add     rsp, 65640 ;3864.33
pop     rbp ;3864.33
pop     r15 ;3864.33
pop     r14 ;3864.33
pop     r13 ;3864.33
pop     r12 ;3864.33
pop     rdi ;3864.33
pop     rsi ;3864.33
pop     rbx ;3864.33
ret     ;3864.33
; LOE
.B11.64::      ; Preds .B11.54
mov     r10d, r9d ;3867.12
; LOE rsi rdi r8 r12 edx ecx ebx ebp r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.60 .B11.56 .B11.57 .B11.64
.B11.65::
add     edx, r10d ;3868.13
cmp     edx, ebx ;3856.25
jbe     .B11.54 ; Prob 82% ;3856.25
; LOE rsi rdi r8 r12 edx ecx ebx ebp r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.65
.B11.66::
xor     eax, eax ;3871.10
add     rsp, 65640 ;3871.10
pop     rbp ;3871.10
pop     r15 ;3871.10
pop     r14 ;3871.10
pop     r13 ;3871.10
pop     r12 ;3871.10
pop     rdi ;3871.10
pop     rsi ;3871.10
pop     rbx ;3871.10
ret     ;3871.10
; LOE
.B11.67::      ; Preds .B11.19
mov     ebx, DWORD PTR [r12] ;3777.43
ecx, DWORD PTR [-1+rsi] ;3797.28
movzx   ebp, bl ;3781.5
mov     edx, esi ;3776.20
mov     r8d, ebp ;3782.30
mov     r9d, ebp ;3783.30
mov     r10d, ebp ;3784.30
shl     r8d, 8 ;3782.30
add     rdi, rdx ;3776.20
shl     r9d, 16 ;3783.30
shl     r10d, 24 ;3784.30
mov     QWORD PTR [65608+rsp], r12 ;3797.28
; LOE rdx rsi rdi r11 ecx ebx ebp r8d r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.80 .B11.67
.B11.68::
mov     rax, rdi ;3789.36
xor     r12d, r12d ;3788.2
sub     rax, rdx ;3789.36
mov     r13d, DWORD PTR [rax] ;3789.45
cmp     ebx, r13d ;3791.31
jne     .B11.77 ; Prob 50% ;3791.31
; LOE rax rdx rsi rdi r11 ecx ebx ebp r8d r9d r10d r12d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.68
.B11.69::
mov     r14d, ecx ;
mov     r13d, ecx ;3797.10
movsxd  r15, ecx ;
neg     r14d ;
neg     r15 ;
add     r14d, esi ;
test    ecx, ecx ;3798.18
je      .B11.76 ; Prob 10% ;3798.18
; LOE rax rdx rsi rdi r11 r14 r15 ecx ebx ebp r8d r9d r10d r12d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.69
.B11.70::
mov     DWORD PTR [65624+rsp], esi ;
mov     QWORD PTR [40+rsp], rdx ;
mov     QWORD PTR [32+rsp], r11 ;
mov     rsi, QWORD PTR [65608+rsp] ;
; LOE rax rsi rdi r14 r15 ecx ebx ebp r8d r9d r10d r12d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.74 .B11.70
.B11.71::
movsx   edx, BYTE PTR [r15+rdi] ;3798.88
cmp     dl, BYTE PTR [r14+rsi] ;3798.88
jne     .B11.169 ; Prob 20% ;3798.88
; LOE rax rsi rdi r15 edx ecx ebx ebp r8d r9d r10d r12d r13d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.71
.B11.72::
lea     r11d, DWORD PTR [r12+r13] ;3799.46
cmp     ecx, r11d ;3799.46
jne     .B11.74 ; Prob 50% ;3799.46
; LOE rax rsi rdi r15 edx ecx ebx ebp r8d r9d r10d r12d r13d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.72
.B11.73::
cmp     ebp, edx ;3799.94
lea     r11d, DWORD PTR [1+r12] ;3799.94
cmovne  r12d, r11d ;3799.94
; LOE rax rsi rdi r15 ecx ebx ebp r8d r9d r10d r12d r13d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.73 .B11.72
.B11.74::
inc     r14d ;3800.16
inc     r15 ;3800.16
dec     r13d ;3800.16

```

```

jne .B11.71 ; Prob 82% ;3798.18
; LOE rax rsi rdi r14 r15 ecx ebx ebp r8d r9d r10d r12d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
.B11.76:: ; Preds .B11.69 .B11.74
add rsp, 65640 ;3802.44
pop rbp ;3802.44
pop r15 ;3802.44
pop r14 ;3802.44
pop r13 ;3802.44
pop r12 ;3802.44
pop rdi ;3802.44
pop rsi ;3802.44
pop rbx ;3802.44
ret ;3802.44

; LOE
.B11.77:: ; Preds .B11.68
mov eax, r13d ;3804.43
and eax, 65280 ;3804.43
cmp r8d, eax ;3804.43
je .B11.80 ; Prob 50% ;3804.43
; LOE rdx rsi rdi r11 ecx ebx ebp r8d r9d r10d r12d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.77
.B11.78::
mov eax, r13d ;3806.48
mov r12d, 1 ;3805.10
and eax, 16711680 ;3806.48
cmp r9d, eax ;3806.48
je .B11.80 ; Prob 50% ;3806.48
; LOE rdx rsi rdi r11 ecx ebx ebp r8d r9d r10d r12d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.78
.B11.79::
and r13d, -16777216 ;3808.53
mov eax, 3 ;3808.67
mov r12d, 2 ;3808.67
cmp r10d, r13d ;3808.67
cmovne r12d, eax ;3808.67
; LOE rdx rsi rdi r11 ecx ebx ebp r8d r9d r10d r12d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.169 .B11.79 .B11.78 .B11.77
.B11.80::
inc r12d ;3813.2
add rdi, r12 ;3815.13
cmp rdi, r11 ;3816.24
jbe .B11.68 ; Prob 80% ;3816.24
; LOE rdx rsi rdi r11 ecx ebx ebp r8d r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.80
.B11.81::
xor eax, eax ;3817.19
add rsp, 65640 ;3817.19
pop rbp ;3817.19
pop r15 ;3817.19
pop r14 ;3817.19
pop r13 ;3817.19
pop r12 ;3817.19
pop rdi ;3817.19
pop rsi ;3817.19
pop rbx ;3817.19
ret ;3817.19

; LOE
.B11.82:: ; Preds .B11.18
xor ebp, ebp ;4360.1
lea ecx, DWORD PTR [-3+rsi] ;4361.29
mov edx, 1 ;4361.6
mov r8d, -1 ;
mov r10d, esi ;
cmp ecx, 1 ;4361.41
jbe .B11.184 ; Prob 10% ;4361.41
; LOE rsi rdi r12 r13 edx ecx ebx ebp r8d r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.82
.B11.83::
mov DWORD PTR [32+rsp], ebx ;4364.54
lea eax, DWORD PTR [-2+rsi] ;4362.42
mov QWORD PTR [40+rsp], rdi ;4364.54
lea r11d, DWORD PTR [-3+rsi] ;4364.54
mov QWORD PTR [48+rsp], r13 ;4364.54
; LOE rsi r12 eax edx ecx ebp r8d r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.96 .B11.83
.B11.84::
mov r14d, edx ;4363.2
mov ebx, eax ;4362.2
cmp edx, r11d ;4364.54
ja .B11.92 ; Prob 10% ;4364.54
; LOE rsi r12 r14 eax edx ecx ebx ebp r8d r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.84
.B11.85::
mov DWORD PTR [65624+rsp], esi ;4365.3
lea edi, DWORD PTR [-3+rsi] ;4366.33
mov r13d, edx ;4365.3
; LOE r12 r14 eax edx ecx ebx ebp edi r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.90 .B11.85
.B11.86::
inc r13d ;4365.33
mov r15d, r13d ;4365.3
cmp r13d, edi ;4366.33
ja .B11.90 ; Prob 10% ;4366.33
; LOE r12 r14 r15 eax edx ecx ebx ebp edi r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.86
.B11.87::
mov esi, DWORD PTR [-1+r14+r12] ;4367.57
; LOE r12 r15 eax edx ecx ebx ebp esi edi r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.88 .B11.87
.B11.88::
lea r14d, DWORD PTR [-1+r15] ;4366.33
cmp esi, DWORD PTR [-1+r15+r12] ;4367.98
jne L26 ; Prob 50% ;4366.33
mov edi, r14d ;4366.33
L26:
jne L27 ; Prob 50% ;4367.98
mov ebx, r15d ;4367.98
L27:
inc r15d ;4368.4
cmp r15d, edi ;4366.33

```

```

jbe .B11.88 ; Prob 82% ;4366.33
; LOE r12 r15 eax edx ecx ebx ebp esi edi r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.88 .B11.86
.B11.90::
mov r14d, r13d ;4370.3
cmp r13d, edi ;4364.54
jbe .B11.86 ; Prob 82% ;4364.54
; LOE r12 r14 eax edx ecx ebx ebp edi r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.90
.B11.91::
mov esi, DWORD PTR [65624+rsp] ;
; LOE r12 eax edx ecx ebx ebp esi r8d r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.91 .B11.84
.B11.92::
lea ebx, DWORD PTR [3+r8+rbx] ;4372.2
cmp ebx, ebp ;4373.31
jb .B11.95 ; Prob 50% ;4373.31
; LOE rsi r12 eax edx ecx ebx ebp r8d r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.92
.B11.93::
mov r9d, edx ;4373.46
mov ebp, ebx ;4373.64
cmp ebx, r10d ;4374.23
jae .B11.97 ; Prob 20% ;4374.23
; LOE rsi r12 eax edx ecx ebx ebp r8d r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.93
.B11.94::
cmp ebx, 128 ;4375.21
ja .B11.97 ; Prob 20% ;4375.21
jmp .B11.96 ; Prob 100% ;4375.21
; LOE rsi r12 eax edx ecx ebp r8d r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.92
.B11.95::
cmp ebp, r10d ;4374.23
jae .B11.97 ; Prob 20% ;4374.23
; LOE rsi r12 eax edx ecx ebp r8d r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.95 .B11.94
.B11.96::
inc edx ;4361.46
dec r10d ;4361.46
dec r8d ;4361.46
cmp edx, ecx ;4361.41
jb .B11.84 ; Prob 82% ;4361.41
; LOE rsi r12 eax edx ecx ebp r8d r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.93 .B11.95 .B11.94 .B11.96
.B11.97::
mov ebx, DWORD PTR [32+rsp] ;
mov rdi, QWORD PTR [40+rsp] ;
mov r13, QWORD PTR [48+rsp] ;
; LOE rbx rsi rdi r12 r13 ebx ebp edi r9d r13d b1 bh dil r13b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.97
.B11.98::
cmp ebp, 19 ;4605.19
lea r14d, DWORD PTR [-1+r9] ;4500.41
lea r15, QWORD PTR [r12+r14] ;4500.13
mov r10d, DWORD PTR [r15] ;4608.33
ja .B11.131 ; Prob 50% ;4605.19
; LOE rbx rsi rdi r12 r13 r14 r15 ebx ebp edi r9d r10d r13d b1 bh dil r13b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.184 .B11.98
.B11.99::
xor edx, edx ;4609.33
lea rcx, QWORD PTR [32+rsp] ;4609.33
mov r8d, 65536 ;4609.33
mov DWORD PTR [65568+rsp], r10d ;4609.33
mov DWORD PTR [65616+rsp], r9d ;4609.33
call _intel_fast_memset ;4609.33
; LOE rsi rdi r13 r14 r15 ebx ebp xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.99
.B11.100::
mov r11d, ebp ;4616.28
mov r9d, DWORD PTR [65616+rsp] ;
dec r11d ;4616.28
mov r10d, DWORD PTR [65568+rsp] ;
je .B11.107 ; Prob 50% ;4616.28
; LOE rsi rdi r9 r10 r13 r14 r15 ebx ebp r9d r10d r11d r9b r10b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.100
.B11.101::
mov eax, 1 ;4616.4
lea ecx, DWORD PTR [-1+rbp] ;4616.4
mov r8d, ecx ;4616.4
xor edx, edx ;4616.4
shr r8d, 1 ;4616.4
test r8d, r8d ;4616.4
jbe .B11.105 ; Prob 15% ;4616.4
; LOE rsi rdi r9 r10 r13 r14 r15 eax edx ecx ebx ebp r8d r9d r10d r11d r9b r10b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.101
.B11.102::
mov al, 1 ;3854.36
; LOE rsi rdi r13 r14 r15 edx ecx ebx ebp r8d r9d r10d r11d al xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.103 .B11.102
.B11.103::
lea r12d, DWORD PTR [rdx+rdx] ;4616.75
movzx r12d, WORD PTR [r12+r15] ;4616.75
mov BYTE PTR [32+rsp+r12], al ;4616.36
lea r12d, DWORD PTR [1+rdx+rdx] ;4616.75
inc edx ;4616.4
cmp edx, r8d ;4616.4
movzx r12d, WORD PTR [r12+r15] ;4616.75
mov BYTE PTR [32+rsp+r12], al ;4616.36
jb .B11.103 ; Prob 64% ;4616.4
; LOE rsi rdi r13 r14 r15 edx ecx ebx ebp r8d r9d r10d r11d al xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.103
.B11.104::
lea eax, DWORD PTR [1+rdx+rdx] ;4616.4
; LOE rsi rdi r13 r14 r15 eax ecx ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.104 .B11.101
.B11.105::
dec eax ;4616.36
cmp eax, ecx ;4616.4
jae .B11.107 ; Prob 15% ;4616.4
; LOE rax rsi rdi r13 r14 r15 ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.105
.B11.106::
movzx edx, WORD PTR [rax+r15] ;4616.75
mov BYTE PTR [32+rsp+rdx], 1 ;4616.36
; LOE rsi rdi r13 r14 r15 ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.100 .B11.105
.B11.107::

```



```

mov     r8, r15                                ;4646.70
lea     eax, DWORD PTR [-4+rsi]                 ;4645.109
lea     ecx, DWORD PTR [-3+rbp]                 ;4621.113
sub     r8, r14                                ;4646.70
add     rax, rdi                                ;4645.54
movsxd  rcx, ecx                                ;4623.8
lea     r12d, DWORD PTR [-3+rsi]                ;4647.36
mov     QWORD PTR [65576+rsp], rcx              ;4623.8
xor     edx, edx                                ;4617.4
movsxd  r12, r12d                               ;4647.4
sub     ebx, ebp                                ;4618.16
mov     QWORD PTR [65584+rsp], rax              ;4647.4
lea     r13, QWORD PTR [-4+r13+rdi]             ;4645.134
mov     QWORD PTR [65608+rsp], r8              ;4647.4
mov     QWORD PTR [65592+rsp], r15             ;4647.4
mov     QWORD PTR [65600+rsp], r14             ;4647.4
; LOE rsi rdi r12 r13 edx ecx ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.107 .B11.129
.B11.108::
lea     eax, DWORD PTR [-2+rbp+rdx]            ;4620.72
movzx   r8d, WORD PTR [rax+rdi]                 ;4620.49
cmp     BYTE PTR [32+rsp+r8], 0                 ;4620.79
je      .B11.128                                ; Prob 50%
; LOE rsi rdi r12 r13 edx ecx ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.108
.B11.109::
lea     eax, DWORD PTR [-4+rbp+rdx]            ;4621.75
movzx   r8d, WORD PTR [rax+rdi]                 ;4621.50
cmp     BYTE PTR [32+rsp+r8], 0                 ;4621.82
jne     .B11.111                                ; Prob 50%
; LOE rsi rdi r12 r13 edx ecx ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.109
.B11.110::
mov     r14d, ecx                               ;4621.86
jmp     .B11.129                                ; Prob 100%
; LOE rsi rdi r12 r13 edx ecx ebx ebp r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.109
.B11.111::
mov     eax, edx                               ;4622.26
mov     r14d, 1                                ;4619.5
lea     r8, QWORD PTR [rdi+rax]                 ;4622.26
cmp     r10d, DWORD PTR [r8]                   ;4622.41
jne     .B11.129                                ; Prob 50%
; LOE rax rsi rdi r8 r12 r13 edx ecx ebx ebp r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.111
.B11.112::
mov     r15, QWORD PTR [65576+rsp]              ;4623.8
test    r15, r15                               ;4624.24
jle     .B11.176                                ; Prob 2%
; LOE rax rsi rdi r8 r12 r13 r15 edx ecx ebx ebp r9d r10d r11d r14d r15d r15b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.112
.B11.113::
mov     DWORD PTR [65616+rsp], r9d              ;
mov     DWORD PTR [65624+rsp], esi              ;
mov     r9, r15                                ;
mov     r15, QWORD PTR [65592+rsp]              ;
; LOE rax rdi r8 r9 r12 r13 r15 edx ecx ebx ebp r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.115 .B11.113
.B11.114::
mov     esi, DWORD PTR [-1+r9+r15]              ;4624.59
cmp     esi, DWORD PTR [-1+r9+r8]              ;4624.80
jne     .B11.175                                ; Prob 20%
; LOE rax rdi r8 r9 r12 r13 r15 edx ecx ebx ebp r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.114
.B11.115::
add     r9, -4                                  ;4625.23
test    r9, r9                                  ;4624.24
jg      .B11.114                                ; Prob 82%
; LOE rax rdi r8 r9 r12 r13 r15 edx ecx ebx ebp r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.115
.B11.116::
mov     esi, DWORD PTR [65624+rsp]              ;
cmp     ebp, esi                                ;4627.19
mov     r9d, DWORD PTR [65616+rsp]              ;
mov     QWORD PTR [65592+rsp], r15              ;
je      .B11.127                                ; Prob 50%
; LOE rax rdi r8 r12 r13 edx ecx ebx ebp esi r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.176 .B11.116
.B11.117::
mov     r8d, edx                               ;4645.21
sub     r8d, r9d                               ;4645.21
inc     r8d                                     ;4645.21
js      .B11.129                                ; Prob 16%
; LOE rax rsi rdi r8 r12 r13 edx ecx ebx ebp r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.117
.B11.118::
mov     r15, QWORD PTR [65584+rsp]              ;4645.54
mov     QWORD PTR [65568+rsp], r8              ;4645.54
add     r8, r15                                ;4645.54
cmp     r13, r8                                ;4645.134
jb      .B11.129                                ; Prob 50%
; LOE rax rsi rdi r12 r13 edx ecx ebx ebp r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.118
.B11.119::
mov     r8, QWORD PTR [65568+rsp]              ;4646.22
add     r8, rdi                                ;4646.22
mov     QWORD PTR [65568+rsp], r8              ;4646.22
mov     r15d, DWORD PTR [r8]                   ;4646.22
mov     r8, QWORD PTR [65608+rsp]              ;4646.96
cmp     r15d, DWORD PTR [r8]                   ;4646.96
jne     .B11.129                                ; Prob 50%
; LOE rax rsi rdi r12 r13 edx ecx ebx ebp r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.119
.B11.120::
mov     r8, r12                                ;4647.4
test    r12, r12                               ;4648.20
jle     .B11.124                                ; Prob 2%
; LOE rax rsi rdi r8 r12 r13 edx ecx ebx ebp r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.120
.B11.121::
mov     DWORD PTR [65616+rsp], r9d              ;
mov     DWORD PTR [65624+rsp], esi              ;
mov     r9, QWORD PTR [65568+rsp]              ;
mov     r15, QWORD PTR [65608+rsp]              ;

```

```

; LOE rax rdi r8 r9 r12 r13 r15 edx ecx ebx ebp r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.123 .B11.121
.B11.122::
mov     esi, DWORD PTR [-1+r8+r15] ;4648.74
cmp     esi, DWORD PTR [-1+r8+r9] ;4648.95
jne     .B11.173 ; Prob 20% ;4648.95
; LOE rax rdi r8 r9 r12 r13 r15 edx ecx ebx ebp r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.122
.B11.123::
add     r8, -4 ;4649.19
test    r8, r8 ;4648.20
jg      .B11.122 ; Prob 82% ;4648.20
; LOE rax rdi r8 r9 r12 r13 r15 edx ecx ebx ebp r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.120 .B11.123
.B11.124::
mov     r14, QWORD PTR [65600+rsp] ;
; LOE rax rdi r14 r14d r14b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.124
.B11.125::
sub     rdi, r14 ;4650.29
add     rax, rdi ;4650.29
add     rsp, 65640 ;4650.56
pop     rbp ;4650.56
pop     r15 ;4650.56
pop     r14 ;4650.56
pop     r13 ;4650.56
pop     r12 ;4650.56
pop     rdi ;4650.56
pop     rsi ;4650.56
pop     rbx ;4650.56
ret     ;4650.56

; LOE
; Preds .B11.176 .B11.116
.B11.127::
mov     rax, r8 ;4655.33
add     rsp, 65640 ;4655.33
pop     rbp ;4655.33
pop     r15 ;4655.33
pop     r14 ;4655.33
pop     r13 ;4655.33
pop     r12 ;4655.33
pop     rdi ;4655.33
pop     rsi ;4655.33
pop     rbx ;4655.33
ret     ;4655.33

; LOE
; Preds .B11.108
.B11.128::
mov     r14d, r11d ;4659.12

; LOE rsi rdi r12 r13 edx ecx ebx ebp r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.175 .B11.173 .B11.110 .B11.119 .B11.118
.B11.129::
; .B11.117 .B11.111 .B11.128
add     edx, r14d ;4660.13
cmp     edx, ebx ;4618.25
jbe     .B11.108 ; Prob 82% ;4618.25
; LOE rsi rdi r12 r13 edx ecx ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.129
.B11.130::
xor     eax, eax ;4663.10
add     rsp, 65640 ;4663.10
pop     rbp ;4663.10
pop     r15 ;4663.10
pop     r14 ;4663.10
pop     r13 ;4663.10
pop     r12 ;4663.10
pop     rdi ;4663.10
pop     rsi ;4663.10
pop     rbx ;4663.10
ret     ;4663.10

; LOE
; Preds .B11.98
.B11.131::
xor     edx, edx ;4669.33
lea     rcx, QWORD PTR [32+rsp] ;4669.33
mov     r8d, 65536 ;4669.33
mov     DWORD PTR [65568+rsp], r10d ;4669.33
mov     DWORD PTR [65616+rsp], r9d ;4669.33
call    _intel_fast_memset ;4669.33
; LOE rbx rsi rdi r12 r13 r14 r15 ebx ebp edi r13d b1 bh di1 r13b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.131
.B11.132::
mov     ecx, ebp ;4682.30
mov     r9d, DWORD PTR [65616+rsp] ;
add     ecx, -3 ;4682.30
mov     r10d, DWORD PTR [65568+rsp] ;
je      .B11.139 ; Prob 50% ;4682.30
; LOE rbx rsi rdi r9 r10 r12 r13 r14 r15 ecx ebx ebp edi r9d r10d r13d b1 bh di1 r9b r10b r13b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13
xmm14 xmm15
; Preds .B11.132
.B11.133::
mov     r8d, ecx ;4682.4
mov     eax, 1 ;4682.4
shr     r8d, 1 ;4682.4
xor     edx, edx ;4682.4
test    r8d, r8d ;4682.4
jbe     .B11.137 ; Prob 15% ;4682.4
; LOE rbx rsi rdi r9 r10 r12 r13 r14 r15 eax edx ecx ebx ebp edi r8d r9d r10d r13d b1 bh di1 r9b r10b r13b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11
xmm12 xmm13 xmm14 xmm15
; Preds .B11.133
.B11.134::
mov     DWORD PTR [65624+rsp], esi ;3854.36
mov     al, 1 ;3854.36
; LOE rdi r12 r13 r14 r15 edx ecx ebx ebp r8d r9d r10d al xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.135 .B11.134
.B11.135::
lea     esi, DWORD PTR [rdx+rdx] ;4682.74
mov     r11d, DWORD PTR [rsi+r15] ;4682.113
mov     esi, r11d ;4682.90
shr     esi, 15 ;4682.90
add     esi, r11d ;4682.128
movzx   r11d, si ;4682.140
lea     esi, DWORD PTR [1+rdx+rdx] ;4682.113

```

```

inc     edx                                ;4682.4
mov     BYTE PTR [32+rsp+r11], al         ;4682.38
mov     r11d, DWORD PTR [rsi+r15]         ;4682.74
mov     esi, r11d                         ;4682.90
shr     esi, 15                           ;4682.90
add     esi, r11d                         ;4682.128
cmp     edx, r8d                          ;4682.4
movzx   r11d, si                          ;4682.140
mov     BYTE PTR [32+rsp+r11], al         ;4682.38
jb      .B11.135                          ; Prob 64%
; LOE rdi r12 r13 r14 r15 edx ecx ebx ebp r8d r9d r10d al xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.135
.B11.136::
mov     esi, DWORD PTR [65624+rsp]        ;
lea     eax, DWORD PTR [1+rdx+rdx]        ;4682.4
; LOE rdi r12 r13 r14 r15 eax ecx ebx ebp esi r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.136 .B11.133
.B11.137::
dec     eax                                ;4682.38
cmp     eax, ecx                          ;4682.4
jae     .B11.139                          ; Prob 15%
; LOE rax rsi rdi r12 r13 r14 r15 ecx ebx ebp r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.137
.B11.138::
mov     edx, DWORD PTR [rax+r15]          ;4682.74
mov     r8d, edx                          ;4682.90
shr     r8d, 15                           ;4682.90
add     r8d, edx                          ;4682.128
movzx   r11d, r8w                         ;4682.140
mov     BYTE PTR [32+rsp+r11], 1          ;4682.38
; LOE rsi rdi r12 r13 r14 r15 ecx ebx ebp r9d r10d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.137 .B11.132 .B11.138
.B11.139::
movsxd  rcx, ecx                          ;4754.8
lea     r8d, DWORD PTR [-4+rsi]           ;4784.109
lea     rax, QWORD PTR [-4+r13+rdi]       ;4784.134
mov     QWORD PTR [65584+rsp], rcx        ;4754.8
lea     r13d, DWORD PTR [-3+rsi]          ;4786.36
movsxd  r13, r13d                         ;4786.4
xor     edx, edx                          ;4683.4
mov     QWORD PTR [65592+rsp], r15         ;4706.42
sub     ebx, ebp                          ;4684.16
mov     QWORD PTR [65600+rsp], r14         ;4706.42
r8, rdi                                    ;4784.54
mov     QWORD PTR [65608+rsp], r12         ;4706.42
lea     r11d, DWORD PTR [-8+rbp]          ;4706.42
; LOE rax rdx rsi rdi r8 r13 ecx ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.139 .B11.162
.B11.140::
lea     r12d, DWORD PTR [-4+rbp+rdx]      ;4688.123
mov     r14d, DWORD PTR [r12+rdi]         ;4688.46
mov     r15d, r14d                       ;4688.75
shr     r15d, 15                          ;4688.75
add     r15d, r14d                       ;4688.126
movzx   r12d, r15w                        ;4688.138
cmp     BYTE PTR [32+rsp+r12], 0          ;4688.156
je      .B11.161                          ; Prob 50%
; LOE rax rdx rsi rdi r8 r13 ecx ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.140
.B11.141::
mov     r12d, 1                           ;4685.5
lea     r14d, DWORD PTR [-9+rbp+rdx]      ;4704.131
mov     r15d, DWORD PTR [r14+rdi]         ;4704.44
mov     r14d, r15d                       ;4704.78
shr     r14d, 15                          ;4704.78
add     r14d, r15d                       ;4704.134
movzx   r15d, r14w                        ;4704.146
cmp     BYTE PTR [32+rsp+r15], 0          ;4704.166
je      .B11.143                          ; Prob 50%
; LOE rax rdx rsi rdi r8 r13 ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.141
.B11.142::
lea     r14d, DWORD PTR [-7+rbp+rdx]      ;4705.134
mov     r15d, DWORD PTR [r14+rdi]         ;4705.47
mov     r14d, r15d                       ;4705.81
shr     r14d, 15                          ;4705.81
add     r14d, r15d                       ;4705.137
movzx   r15d, r14w                        ;4705.149
cmp     BYTE PTR [32+rsp+r15], 0          ;4705.169
jne     .B11.144                          ; Prob 50%
; LOE rax rdx rsi rdi r8 r13 ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.141 .B11.142
.B11.143::
mov     r12d, r11d                       ;4706.8
jmp     .B11.162                          ; Prob 100%
; LOE rax rdx rsi rdi r8 r13 ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.142
.B11.144::
mov     QWORD PTR [65576+rsp], rdx         ;4742.26
lea     r15, QWORD PTR [rdi+rdx]          ;4742.26
mov     r10d, DWORD PTR [r15]             ;4742.41
jne     .B11.162                          ; Prob 50%
; LOE rax rsi rdi r8 r13 r15 edx ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.144
.B11.145::
mov     r14, QWORD PTR [65584+rsp]         ;4754.8
test    r14, r14                         ;4756.24
jle     .B11.181                          ; Prob 2%
; LOE rax rsi rdi r8 r9 r13 r14 r15 edx ecx ebx ebp r9d r10d r11d r12d r14d r14b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.145
.B11.146::
mov     DWORD PTR [65616+rsp], r9d        ;
mov     DWORD PTR [65624+rsp], esi        ;
mov     r9, QWORD PTR [65592+rsp]         ;
; LOE rax rdi r8 r9 r13 r14 r15 edx ecx ebx ebp r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.148 .B11.146
.B11.147::
mov     esi, DWORD PTR [-1+r14+r9]        ;4756.59
cmp     esi, DWORD PTR [-1+r14+r15]       ;4756.80
jne     .B11.180                          ; Prob 20%
; LOE rax rdi r8 r9 r13 r14 r15 edx ecx ebx ebp r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.147
.B11.148::

```

```

add     r14, -4                                ;4757.23
test    r14, r14                              ;4756.24
jg      .B11.147                               ;4756.24
; Prob 82%
; LOE rax rdi r8 r9 r13 r14 r15 edx ecx ebx ebp r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.148
.B11.149::
mov     esi, DWORD PTR [65624+rsp]              ;
cmp     ebp, esi                               ;4759.19
mov     QWORD PTR [65592+rsp], r9                ;
mov     r9d, DWORD PTR [65616+rsp]              ;
je      .B11.160                               ;4759.19
; Prob 50%
; LOE rax rdi r8 r13 r15 edx ecx ebx ebp esi r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.181 .B11.149
.B11.150::
mov     r14d, edx                             ;4784.21
sub     r14d, r9d                             ;4784.21
inc     r14d                                  ;4784.21
js      .B11.162                               ;4784.46
; Prob 16%
; LOE rax rsi rdi r8 r13 r14 edx ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.150
.B11.151::
mov     QWORD PTR [65568+rsp], r14              ;4784.54
lea     r15, QWORD PTR [r8+r14]                ;4784.54
cmp     rax, r15                              ;4784.134
jb      .B11.162                               ;4784.134
; Prob 50%
; LOE rax rsi rdi r8 r13 r14 edx ecx ebx ebp r9d r10d r11d r12d r14b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.151
.B11.152::
add     r14, rdi                             ;4785.22
mov     QWORD PTR [65568+rsp], r14              ;4785.22
mov     r15d, DWORD PTR [r14]                  ;4785.22
mov     r14, QWORD PTR [65608+rsp]              ;4785.96
cmp     r15d, DWORD PTR [r14]                  ;4785.96
jne     .B11.162                               ;4785.96
; Prob 50%
; LOE rax rsi rdi r8 r13 edx ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.152
.B11.153::
mov     r14, r13                             ;4786.4
test    r13, r13                             ;4787.20
jle     .B11.157                               ;4787.20
; Prob 2%
; LOE rax rsi rdi r8 r13 r14 edx ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.153
.B11.154::
mov     DWORD PTR [65616+rsp], r9d              ;
mov     DWORD PTR [65624+rsp], esi              ;
mov     r9, QWORD PTR [65568+rsp]              ;
mov     r15, QWORD PTR [65608+rsp]              ;
; LOE rax rdi r8 r9 r13 r14 r15 edx ecx ebx ebp r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.156 .B11.154
.B11.155::
mov     esi, DWORD PTR [-1+r14+r15]            ;4787.74
cmp     esi, DWORD PTR [-1+r14+r9]             ;4787.95
jne     .B11.178                               ;4787.95
; Prob 20%
; LOE rax rdi r8 r9 r13 r14 r15 edx ecx ebx ebp r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.155
.B11.156::
add     r14, -4                                ;4788.19
test    r14, r14                              ;4787.20
jg      .B11.155                               ;4787.20
; Prob 82%
; LOE rax rdi r8 r9 r13 r14 r15 edx ecx ebx ebp r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.153 .B11.156
.B11.157::
mov     rax, QWORD PTR [65576+rsp]              ;
mov     r14, QWORD PTR [65600+rsp]              ;
; LOE rax rdi r14 eax r14d al ah r14b xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.157
.B11.158::
sub     rdi, r14                             ;4789.29
add     rax, rdi                             ;4789.29
add     rsp, 65640                            ;4789.56
pop     rbp                                  ;4789.56
pop     r15                                  ;4789.56
pop     r14                                  ;4789.56
pop     r13                                  ;4789.56
pop     r12                                  ;4789.56
pop     rdi                                  ;4789.56
pop     rsi                                  ;4789.56
pop     rbx                                  ;4789.56
ret                                           ;4789.56
; LOE
; Preds .B11.181 .B11.149
.B11.160::
mov     rax, r15                             ;4794.33
add     rsp, 65640                            ;4794.33
pop     rbp                                  ;4794.33
pop     r15                                  ;4794.33
pop     r14                                  ;4794.33
pop     r13                                  ;4794.33
pop     r12                                  ;4794.33
pop     rdi                                  ;4794.33
pop     rsi                                  ;4794.33
pop     rbx                                  ;4794.33
ret                                           ;4794.33
; LOE
; Preds .B11.140
.B11.161::
mov     r12d, ecx                             ;4821.12
; LOE rax rdx rsi rdi r8 r13 ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.180 .B11.178 .B11.152 .B11.151 .B11.150
; .B11.144 .B11.143 .B11.161
.B11.162::
add     edx, r12d                             ;4822.13
cmp     edx, ebx                             ;4684.25
jbe     .B11.140                               ;4684.25
; Prob 82%
; LOE rax rdx rsi rdi r8 r13 ecx ebx ebp r9d r10d r11d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.162
.B11.163::
xor     eax, eax                             ;4825.10
add     rsp, 65640                            ;4825.10
pop     rbp                                  ;4825.10
pop     r15                                  ;4825.10
pop     r14                                  ;4825.10
pop     r13                                  ;4825.10
pop     r12                                  ;4825.10

```

```

pop     rdi                     ;4825.10
pop     rsi                     ;4825.10
pop     rbx                     ;4825.10
ret                             ;4825.10

; LOE
.B11.169::
; Preds .B11.71
; Infreq
mov     QWORD PTR [65608+rsp], rsi
mov     rdx, QWORD PTR [40+rsp]
mov     r11, QWORD PTR [32+rsp]
mov     esi, DWORD PTR [65624+rsp]
jmp     .B11.80
; Prob 100%
; LOE rdx rdi r11 ecx ebx ebp esi r8d r9d r10d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.122
; Infreq
.B11.173::
mov     QWORD PTR [65608+rsp], r15
mov     r9d, DWORD PTR [65616+rsp]
mov     esi, DWORD PTR [65624+rsp]
jmp     .B11.129
; Prob 100%
; LOE rdi r12 r13 edx ecx ebx ebp esi r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.114
; Infreq
.B11.175::
mov     QWORD PTR [65592+rsp], r15
mov     r9d, DWORD PTR [65616+rsp]
mov     esi, DWORD PTR [65624+rsp]
jmp     .B11.129
; Prob 100%
; LOE rdi r12 r13 edx ecx ebx ebp esi r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.112
; Infreq
.B11.176::
cmp     ebp, esi
jne     .B11.117
; Prob 50%
jmp     .B11.127
; Prob 100%
; LOE rax rsi rdi r8 r12 r13 edx ecx ebx ebp r9d r10d r11d r14d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.155
; Infreq
.B11.178::
mov     QWORD PTR [65608+rsp], r15
mov     r9d, DWORD PTR [65616+rsp]
mov     esi, DWORD PTR [65624+rsp]
jmp     .B11.162
; Prob 100%
; LOE rax rdi r8 r13 edx ecx ebx ebp esi r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.147
; Infreq
.B11.180::
mov     QWORD PTR [65592+rsp], r9
mov     r9d, DWORD PTR [65616+rsp]
mov     esi, DWORD PTR [65624+rsp]
jmp     .B11.162
; Prob 100%
; LOE rax rdi r8 r13 edx ecx ebx ebp esi r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.145
; Infreq
.B11.181::
cmp     ebp, esi
jne     .B11.150
; Prob 50%
jmp     .B11.160
; Prob 100%
; LOE rax rsi rdi r8 r13 r15 edx ecx ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.82
; Infreq
.B11.184::
lea     r14d, DWORD PTR [-1+r9]
;4500.41
lea     r15, QWORD PTR [r12+r14]
;4500.13
mov     r10d, DWORD PTR [r15]
;4608.33
jmp     .B11.99
; Prob 100%
;4608.33
; LOE rsi rdi r13 r14 r15 ebx ebp r9d r10d r11d r12d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
; Preds .B11.12
; Infreq
.B11.185::
mov     rax, rcx
;3748.78
add     rsp, 65640
;3748.78
pop     rbp
;3748.78
pop     r15
;3748.78
pop     r14
;3748.78
pop     r13
;3748.78
pop     r12
;3748.78
pop     rdi
;3748.78
pop     rsi
;3748.78
pop     rbx
;3748.78
ret
;3748.78
ALIGN 16
; LOE
.B11.187::
; mark_end;
Railgun_Trolldom ENDP
;Railgun_Trolldom ENDS
_TEXT ENDS
*/

```

Railgun\_Trolldom is the successor of Railgun\_Swampshine\_BailOut, which is the successor of Railgun\_Ennearch.