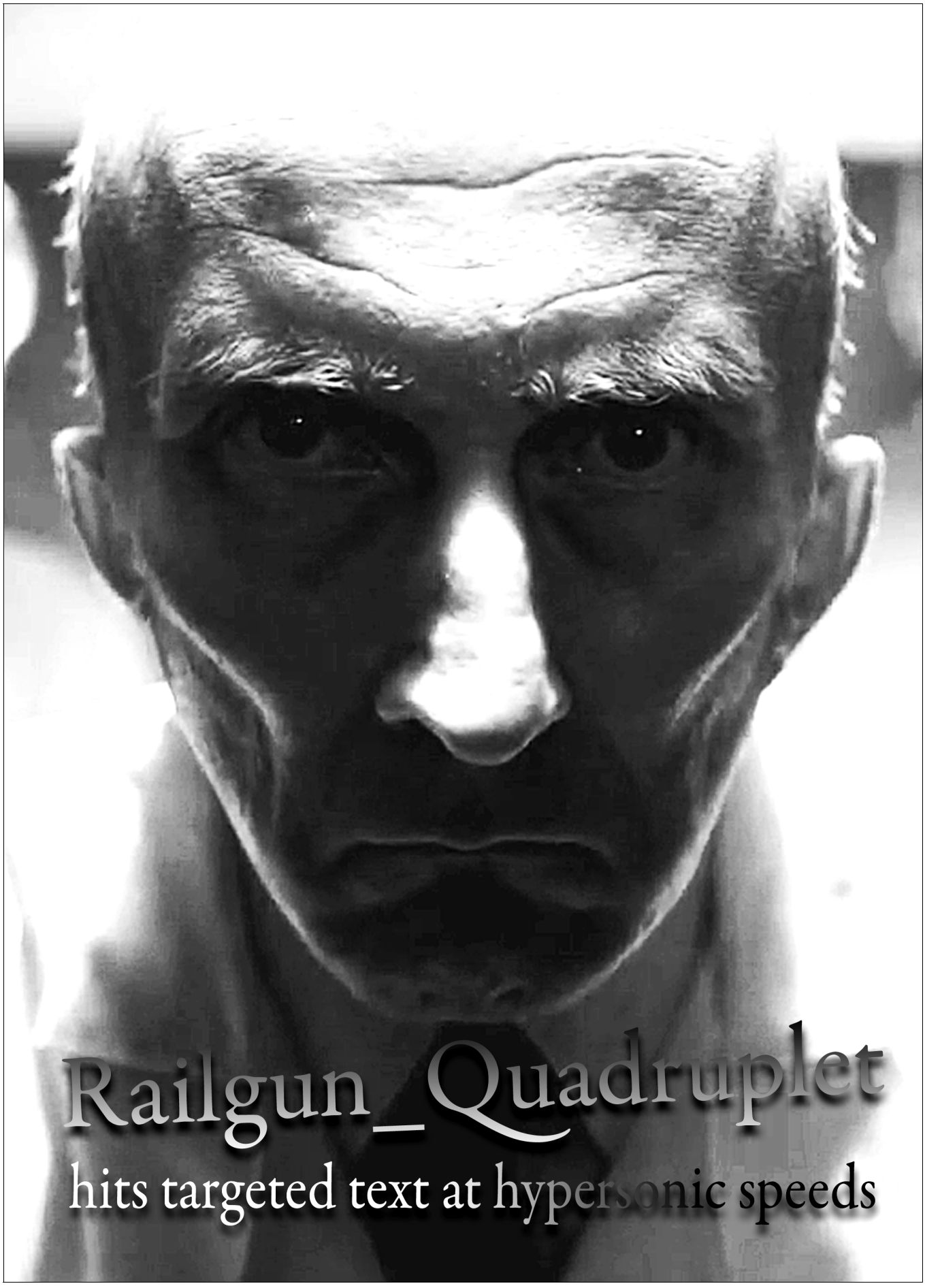
Railgun_Sekireigan



鶺鴒眼

Sekireigan

FASTEST MEMMEM

```
001 // ''Sekirei''|鶺鴒| are superpowered beings with a different code similar to a human's.
002 // wagtail -  Any of various chiefly Old World birds of the family Motacillidae, having a slender body with a long tail that constantly wags.
003 // wag - A humorous or droll person; a wit.
004 // wag -
005 // v. intr.
006 // 1. To move briskly and repeatedly from side to side, to and fro, or up and down.
007 // 2. To move rapidly in talking. Used of the tongue.
008 // 3. To walk with a clumsy sway; waddle.
009 // 4. Archaic: To be on one's way; depart.
010 // v. tr.
011 // To move (a body part) rapidly from side to side or up and down, as in playfulness, agreement, admonition, or chatter.
012 // /HERITAGE/
013 // ''Kusano''|草野|Sekirei #108, commonly referred to as &amp;quot;Kuu-chan&amp;quot; or &amp;quot;Ku&amp;quot;, is the youngest of Minato's Sekirei, and is also known as the &amp;quot;Green Girl&amp;quot;|緑の少女|Midori no Shōjo by other Sekirei. At the beginning of the story, she was hiding in a botanical garden after being traumatized when Mikogami attempted to forcibly wing her. Kusano communicated with Minato telepathically and led him through the garden until he found her. Kusano refers to Minato as ''Onii-chan'' (big brother), and is the most attached to him. She does not like fighting or quarreling and she can be seen stopping them when they start. She is also very impressionable and often copies Musubi, Tsukiumi and Kazehana's mannerisms. She VERY much wants to be Minato's wife when she grows up, and is highly possessive of him at times, even biting her fellow sekirei when they start to get too close to him.
014 // Apart from its unusual [[plumage]] pattern and habitat, the Forest Wagtail differs from its ''Motacilla'' relatives in its strange habit of swaying its tail from side to side, not wagging it up and down like other wagtails. The Japanese name ''Jokofury-sekirei'' (=sideways-swinging wagtail) ) is based on this habit.
015 // ''Kazehana''|風花|Sekirei #03 is a tall and extremely buxom Sekirei whose first encounter with Minato was to stumble into his bed in a drunken stupor.
016 // Kazehana displays an extremely relaxed personality, preferring to spend most of her time lounging and drinking ''[[sake]]'', and often becomes giddy when discussing matters of love. However, when she becomes serious she displays a lot of power and lets nobody (with the exception of Minaka, Miya, and Minato) talk down to her.
017 // Kazehana has the ability to control and manipulate wind, which also grants her a limited ability of flight.
018 // Her known attacks is the |Flower Whirlwind|花旋風|Hana Senpū ...
019 // ... who teaches him the Sekireigan (Eye of the Sekirei). With this he is able to move extremely fast and attack from many angles simultaneously.
020 // His sword techniques are unnamed, but he uses the Sekireigan, which was taught to him by Anri. Yukimura can utilize the Sekireigan to access extreme speed.
021 // /Wikipedia/
022 // Version: This is Railgun_Sekireigan revision 2, copyleft 2013-Oct-16, Kaze
023 // Purpose: This is optimized strstr-like (memmem in fact) function for short needles up to 255 bytes ... and beyond.
024 // Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
025 #define _rotl_KAZE(x, n) (((x) &amp;lt;&amp;lt; (n)) | ((x) &amp;gt;&amp;gt; (32-(n))))
026 #define HaystackThresholdSekirei 961 // Quadruplet works up to this value, if bigger then BMH2 takes over.
027 #define NeedleThresholdBIGSekirei 12+40 // Should be bigger than 'HasherezadeOrder'. BMH2 works up to this value (inclusive).
028 #define HashTableSizeSekirei 17-1 // In fact the real size is -3, because it is BITwise, when 17-3=14 it means 16KB, (17-1)-3=13 it means 8KB.
029 char * Railgun_Sekireigan_Bari (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern)
030 {
031       char * pbTargetMax = pbTarget + cbTarget;
032       register uint32_t ulHashPattern;
033       register uint32_t ulHashTarget;
034       signed long count;
035       signed long countSTATIC;
036
037       unsigned char SINGLET;
038       uint32_t Quadruplet2nd;
039       uint32_t Quadruplet3rd;
040       uint32_t Quadruplet4th;
041
042       uint32_t AdvanceHopperGrass;
043
044       long i; //BMH needed
```

```
045	int a, j;
046	unsigned char bm_Horspool_Order2[256*256]; // BMHSS(Elsiane) needed, 'char' limits patterns to 255, if 'long' then table becomes 256KB, grrr.
047	uint32_t Gulliver; // or unsigned char or unsigned short
048
049	unsigned char bm_Hasherezade_HASH[1&lt;&lt;(HashTableSizeSekirei-3)];
050	uint32_t hash32;
051	uint32_t hash32B;
052	uint32_t hash32C;
053
054	if (cbPattern &gt; cbTarget) return(NULL);
055
056	if ( cbPattern&lt;4 ) {
057
058		pbTarget = pbTarget+cbPattern;
059		ulHashPattern = ( (*(char *)(pbPattern))&lt;&lt;8 ) + *(pbPattern+(cbPattern-1));
060		if ( cbPattern==3 ) {
061			for ( ;; ) {
062				if ( ulHashPattern == ( (*(char *)(pbTarget-3))&lt;&lt;8 ) + *(pbTarget-1) ) {
063					if ( *(char *)(pbPattern+1) == *(char *)(pbTarget-2) ) return((pbTarget-3));
064				}
065				if ( (char)(ulHashPattern&gt;&gt;8) != *(pbTarget-2) ) {
066					pbTarget++;
067					if ( (char)(ulHashPattern&gt;&gt;8) != *(pbTarget-2) ) pbTarget++;
068				}
069				pbTarget++;
070				if (pbTarget &gt; pbTargetMax) return(NULL);
071			}
072		} else {
073		}
074		for ( ;; ) {
075			if ( ulHashPattern == ( (*(char *)(pbTarget-2))&lt;&lt;8 ) + *(pbTarget-1) ) return((pbTarget-2));
076			if ( (char)(ulHashPattern&gt;&gt;8) != *(pbTarget-1) ) pbTarget++;
077			pbTarget++;
078			if (pbTarget &gt; pbTargetMax) return(NULL);
079		}
080
081	} else {
082		if (cbTarget&lt;HaystackThresholdSekirei) { // This value is arbitrary (don't know how exactly), it ensures (at least must) better performance than 'Boyer_Moore_Horspool'.
083
084		pbTarget = pbTarget+cbPattern;
085		ulHashPattern = *(uint32_t *)(pbPattern);
086		SINGLET = ulHashPattern &amp; 0xFF;
087		Quadruplet2nd = SINGLET&lt;&lt;8;
088		Quadruplet3rd = SINGLET&lt;&lt;16;
089		Quadruplet4th = SINGLET&lt;&lt;24;
090		for ( ;; ) {
091			AdvanceHopperGrass = 0;
092			ulHashTarget = *(uint32_t *)(pbTarget-cbPattern);
093			if ( ulHashPattern == ulHashTarget ) { // Three unnecessary comparisons here, but 'AdvanceHopperGrass' must be calculated - it has a higher priority.
094				count = cbPattern-1;
095				while ( count &amp;&amp; *(char *)(pbPattern+(cbPattern-count)) == *(char *)(pbTarget-count) ) {
096					if ( cbPattern-1==AdvanceHopperGrass+count &amp;&amp; SINGLET != *(char *)(pbTarget-count) ) AdvanceHopperGrass++;
097					count--;
098				}
099				if ( count == 0) return((pbTarget-cbPattern));
100			} else { // The goal here: to avoid memory accesses by stressing the registers.
101				if ( Quadruplet2nd != (ulHashTarget &amp; 0x0000FF00) ) {
102					AdvanceHopperGrass++;
103					if ( Quadruplet3rd != (ulHashTarget &amp; 0x00FF0000) ) {
104						AdvanceHopperGrass++;
105						if ( Quadruplet4th != (ulHashTarget &amp; 0xFF000000) ) AdvanceHopperGrass++;
106					}
107				}
108			}
109			AdvanceHopperGrass++;
110			pbTarget = pbTarget + AdvanceHopperGrass;
111			if (pbTarget &gt; pbTargetMax) return(NULL);
112		}
113
114		} else { //if (cbTarget&lt;HaystackThresholdSekirei)
115
116	if ( cbPattern&lt;=NeedleThresholdBIGSekirei ) {
117
118			countSTATIC = cbPattern-2-2;
119			ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
120			//ulHashTarget = *(unsigned short *)(pbPattern+cbPattern-1-1); // Last two bytes
121			i=0;
122			//for (a=0; a &lt; 256*256; a++) {bm_Horspool_Order2[a]= cbPattern-1;} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
123			//for (j=0; j &lt; cbPattern-1; j++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+j)]=j; // Rightmost appearance/position is needed
124			for (a=0; a &lt; 256*256; a++) {bm_Horspool_Order2[a]=0;}
125			for (j=0; j &lt; cbPattern-1; j++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+j)]=1;
126			while (i &lt;= cbTarget-cbPattern) {
```

```
127                                 Gulliver = 1;
128                                 if ( bm_Horspool_Order2[*(unsigned short *)&amp;pbTarget[i+cbPattern-1-1]] != 0 ) {
129                                         //if ( Gulliver == 1 ) { // Means the Building-Block order 2 is found somewhere i.e. NO MAXIMUM SKIP
130                                                 if ( *(uint32_t *)&amp;pbTarget[i] == ulHashPattern) {
131                                                         count = countSTATIC; // Last two chars already matched, to be fixed with -2
132                                                         while ( count !=0 &amp;&amp; *(char *)(pbPattern+(countSTATIC-count)+4) == *(char *)(&amp;pbTarget[i]+(countSTATIC-count)+4) )
133                                                                 count--;
134                                                         if ( count == 0) return(pbTarget+i);
135                                                 }
136                                         //}
137                                         // Trying to strengthen the Skip performance... here ONLY one additional lookup, for better/longer jumps more such lookups, unrolled to be added.
138                                         if ( bm_Horspool_Order2[*(unsigned short *)&amp;pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2;
139                                 } else Gulliver = cbPattern-(2-1);
140                                 i = i + Gulliver;
141                                 //GlobalI++; // Comment it, it is only for stats.
142                         }
143                         return(NULL);
144
145       } else { // if ( cbPattern&lt;=NeedleThresholdBIGSekirei )
146 // MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() [
147                         countSTATIC = cbPattern-2-2;
148
149                         ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
150                         //ulHashTarget = *(unsigned short *)(pbPattern+cbPattern-1-1); // Last two bytes
151
152                         i=0;
153
154                         for (a=0; a &lt; 1&lt;&lt;(HashTableSizeSekirei-3); a++) {bm_Hasherezade_HASH[a]= 0;} // to-do: 'memset' if not optimized
155                         // cbPattern - Order + 1 i.e. number of BBs for 11 'fastest fox' 11-8+1=4: 'fastest ', 'astest f', 'stest fo', 'test fox'
156                         for (j=0; j &lt; cbPattern-12+1; j++) {
157                                 hash32 = (2166136261 ^ *(uint32_t *)(pbPattern+j+0)) * 709607;
158                                 hash32B = (2166136261 ^ *(uint32_t *)(pbPattern+j+4)) * 709607;
159                                 hash32C = (2166136261 ^ *(uint32_t *)(pbPattern+j+8)) * 709607;
160                                 hash32 = (hash32 ^ _rotl_KAZE(hash32C,5) ) * 709607;
161                                 hash32 = (hash32 ^ _rotl_KAZE(hash32B,5) ) * 709607;
162                                 hash32 = ( hash32 ^ (hash32 &gt;&gt; 16) ) &amp; ( (1&lt;&lt;(HashTableSizeSekirei))-1 );
163                                 bm_Hasherezade_HASH[hash32&gt;&gt;3]= bm_Hasherezade_HASH[hash32&gt;&gt;3] | (1&lt;&lt;(hash32&amp;0x7));
164                         }
165
166                         while (i &lt;= cbTarget-cbPattern) {
167                                 Gulliver = 1; // Assume minimal jump as initial value.
168                                 // The goal: to jump when the rightmost 8bytes (Order 8 Horspool) of window do not look like any of Needle prefixes i.e. are not to be found. This maximum jump equals cbPattern-(Order-1) or 11-(8-1)=4 for 'fastest fox' - a small one but for Needle 31 bytes the jump equals 31-(8-1)=24
169                                 //GlobalHashSectionExecution++; // Comment it, it is only for stats.
170                                         hash32 = (2166136261 ^ *(uint32_t *)(pbTarget+i+cbPattern-12+0)) * 709607;
171                                         hash32B = (2166136261 ^ *(uint32_t *)(pbTarget+i+cbPattern-12+4)) * 709607;
172                                         hash32C = (2166136261 ^ *(uint32_t *)(pbTarget+i+cbPattern-12+8)) * 709607;
173                                         hash32 = (hash32 ^ _rotl_KAZE(hash32C,5) ) * 709607;
174                                         hash32 = (hash32 ^ _rotl_KAZE(hash32B,5) ) * 709607;
175                                         hash32 = ( hash32 ^ (hash32 &gt;&gt; 16) ) &amp; ( (1&lt;&lt;(HashTableSizeSekirei))-1 );
176                                         if ( (bm_Hasherezade_HASH[hash32&gt;&gt;3] &amp; (1&lt;&lt;(hash32&amp;0x7))) ==0 )
177                                                 Gulliver = cbPattern-(12-1);
178
179                                 if ( Gulliver == 1 ) { // Means the Building-Block order 8/12 is found somewhere i.e. NO MAXIMUM SKIP
180                                         if ( *(uint32_t *)&amp;pbTarget[i] == ulHashPattern) {
181                                                 count = countSTATIC;
182                                                 while ( count !=0 &amp;&amp; *(char *)(pbPattern+(countSTATIC-count)+4) == *(char *)(&amp;pbTarget[i]+(countSTATIC-count)+4) )
183                                                         count--;
184                                                 if ( count == 0) return(pbTarget+i);
185                                         }
186                                 }
187                                         i = i + Gulliver;
188                                 //GlobalI++; // Comment it, it is only for stats.
189                         } // while (i &lt;= cbTarget-cbPattern)
190                         return(NULL);
191 // MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() MEMMEM() ]
192       } // if ( cbPattern&lt;=NeedleThresholdBIGSekirei )
193               } //if (cbTarget&lt;HaystackThresholdSekirei)
194       } //if ( cbPattern&lt;4 )
195 }
```

; Main loop of Railgun_Sekireigan_Bari_bit's BMH order 2 section [

```
;                 while (i <= cbTarget-cbPattern) {
;                         Gulliver = 1;
;                         //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
;                         if ( ( bm_Horspool_Order2[(*(unsigned short *)&pbTarget[i+cbPattern-1-1])>>3] & (1<<((*(unsigned short *)&pbTarget[i+cbPattern-1-1])&0x7)) ) != 0 ) {
;                                 //if ( Gulliver == 1 ) { // Means the Building-Block order 2 is found somewhere i.e. NO MAXIMUM SKIP
;                                         if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) {
;                                                 count = countSTATIC; // Last two chars already matched, to be fixed with -2
```

```
;                                                       while ( count !=0 && *(char *)(pbPattern+(countSTATIC-count)+4) == *(char
   *)(&pbTarget[i]+(countSTATIC-count)+4) )
;                                                               count--;
;                                                       if ( count == 0) return(pbTarget+i);
;                                               }
;                                       //}
;                                       // Trying to strengthen the Skip performance... here ONLY one additional lookup, for better/longer jumps more
   such lookups, unrolled to be added.
;                                       //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-
   2;
;                                       if ( ( bm_Horspool_Order2[(*(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 Gulliver = cbPattern-(2-1);
;                               i = i + Gulliver;
;                               //GlobalI++; // Comment it, it is only for stats.
;                       }
;                       return(NULL);

; mark_description "Intel(R) C++ Compiler XE for applications running on IA-32, Version 12.1.1.258 Build 20111011";
; mark_description "-O3 -FAcs -Festrstr_SHORT-SHOWDOWN_LV_WIKI_32bit_IntelV12";

.B3.29:
  001f9 0f b7 4c 02 fe   movzx ecx, WORD PTR [-2+edx+eax]
  001fe 8b e9            mov ebp, ecx
  00200 c1 ed 03         shr ebp, 3
  00203 83 e1 07         and ecx, 7
  00206 0f b6 1c 2c      movzx ebx, BYTE PTR [esp+ebp]
  0020a bd 01 00 00 00   mov ebp, 1
  0020f d3 e5            shl ebp, cl
  00211 85 dd            test ebx, ebp
  00213 74 79            je .B3.38
.B3.30:
  00215 3b 3c 06         cmp edi, DWORD PTR [esi+eax]
  00218 75 4b            jne .B3.37
.B3.31:
  0021a 8b 8c 24 10 20
        00 00            mov ecx, DWORD PTR [8208+esp]
  00221 8b e9            mov ebp, ecx
  00223 f7 dd            neg ebp
  00225 85 c9            test ecx, ecx
  00227 0f 84 52 04 00
        00               je .B3.82
.B3.32:
  0022d 8b 9c 24 08 20
        00 00            mov ebx, DWORD PTR [8200+esp]
  00234 8b f1            mov esi, ecx
  00236 8b bc 24 04 20
        00 00            mov edi, DWORD PTR [8196+esp]
  0023d 03 d8            add ebx, eax
.B3.33:
  0023f 8a 4c 3d 04      mov cl, BYTE PTR [4+ebp+edi]
  00243 3a 4c 1d 04      cmp cl, BYTE PTR [4+ebp+ebx]
  00247 0f 85 1b 04 00
        00               jne .B3.81
.B3.34:
  0024d 45               inc ebp
  0024e 4e               dec esi
  0024f 75 ee            jne .B3.33
.B3.35:
  00251 8b b4 24 34 20
        00 00            mov esi, DWORD PTR [8244+esp]
.B3.36:
  00258 03 c6            add eax, esi
  0025a 81 c4 20 20 00
        00               add esp, 8224
  00260 5d               pop ebp
  00261 5b               pop ebx
  00262 5f               pop edi
  00263 5e               pop esi
  00264 c3               ret
.B3.37:
  00265 0f b7 4c 02 fc   movzx ecx, WORD PTR [-4+edx+eax]
  0026a 8b e9            mov ebp, ecx
  0026c c1 ed 03         shr ebp, 3
  0026f 83 e1 07         and ecx, 7
  00272 0f b6 1c 2c      movzx ebx, BYTE PTR [esp+ebp]
  00276 bd 01 00 00 00   mov ebp, 1
  0027b d3 e5            shl ebp, cl
  0027d b9 01 00 00 00   mov ecx, 1
  00282 85 dd            test ebx, ebp
  00284 0f 44 8c 24 0c
        20 00 00         cmove ecx, DWORD PTR [8204+esp]
  0028c eb 07            jmp .B3.39
.B3.38:
  0028e 8b 8c 24 18 20
        00 00            mov ecx, DWORD PTR [8216+esp]
.B3.39:
  00295 03 c1            add eax, ecx
```

```
  00297 3b 84 24 1c 20
        00 00           cmp eax, DWORD PTR [8220+esp]
  0029e 0f 86 55 ff ff
        ff              jbe .B3.29
.B3.40:
  002a4 33 c0           xor eax, eax
  002a6 81 c4 20 20 00
        00              add esp, 8224
  002ac 5d              pop ebp
  002ad 5b              pop ebx
  002ae 5f              pop edi
  002af 5e              pop esi
  002b0 c3              ret

; The loop is 0029e - 001f9 + 6 = 171 bytes

; Haystack: Wikipedia 50MB (results on i7 3930K @4.5GHz):
; Needle 7 chars:
; Railgun_Sekireigan_Bari_bit performance: 3657KB/clock  !!! Intel /O3 !!!
; Railgun_Sekireigan_Bari_bit performance: 2844KB/clock  !!! Microsoft /Ox !!!
; Boyer_Moore-Horspool performance: 1765KB/clock
; Needle 13 chars:
; Railgun_Sekireigan_Bari_bit performance: 5688KB/clock  !!! Intel /O3 !!!
; Railgun_Sekireigan_Bari_bit performance: 4654KB/clock  !!! Microsoft /Ox !!!
; Boyer_Moore-Horspool performance: 3011KB/clock
; Needle 21 chars:
; Railgun_Sekireigan_Bari_bit performance: 7314KB/clock  !!! Intel /O3 !!!
; Railgun_Sekireigan_Bari_bit performance: 6400KB/clock  !!! Microsoft /Ox !!!
; Boyer_Moore-Horspool performance: 3657KB/clock
; Needle 28 chars:
; Railgun_Sekireigan_Bari_bit performance: 7314KB/clock  !!! Intel /O3 !!!
; Railgun_Sekireigan_Bari_bit performance: 6400KB/clock  !!! Microsoft /Ox !!!
; Boyer_Moore-Horspool performance: 4266KB/clock
; Needle 37 chars:
; Railgun_Sekireigan_Bari_bit performance: 7314KB/clock  !!! Intel /O3 !!!
; Railgun_Sekireigan_Bari_bit performance: 7314KB/clock  !!! Microsoft /Ox !!!
; Boyer_Moore-Horspool performance: 4654KB/clock
; Needle 51 chars:
; Railgun_Sekireigan_Bari_bit performance: 8533KB/clock  !!! Intel /O3 !!!
; Railgun_Sekireigan_Bari_bit performance: 8533KB/clock  !!! Microsoft /Ox !!!
; Boyer_Moore-Horspool performance: 5120KB/clock

; Total non-register accesses in mainloop: Intel C++ Compiler XE for applications running on IA-32, Version 12.1.1.258 makes next 14 memory
    accesses:
  001f9 0f b7 4c 02 fe   movzx ecx, WORD PTR [-2+edx+eax]
  00206 0f b6 1c 2c      movzx ebx, BYTE PTR [esp+ebp]
  00215 3b 3c 06         cmp edi, DWORD PTR [esi+eax]
        00 00            mov ecx, DWORD PTR [8208+esp]
        00 00            mov ebx, DWORD PTR [8200+esp]
        00 00            mov edi, DWORD PTR [8196+esp]
  0023f 8a 4c 3d 04      mov cl, BYTE PTR [4+ebp+edi]
  00243 3a 4c 1d 04      cmp cl, BYTE PTR [4+ebp+ebx]
        00 00            mov esi, DWORD PTR [8244+esp]
  00265 0f b7 4c 02 fc   movzx ecx, WORD PTR [-4+edx+eax]
  00272 0f b6 1c 2c      movzx ebx, BYTE PTR [esp+ebp]
        20 00 00         cmove ecx, DWORD PTR [8204+esp]
        00 00            mov ecx, DWORD PTR [8216+esp]
        00 00            cmp eax, DWORD PTR [8220+esp]

; Total non-register accesses in mainloop: Microsoft Optimizing Compiler Version 16.00.30319.01 makes next 14+10 (!!!!!!!!!!) memory accesses:
  03b82  0f b7 44 37 fe      movzx     eax, WORD PTR [edi+esi-2]
  03b96  8a 44 04 38         mov       al, BYTE PTR _bm_Horspool_Order2$[esp+eax+8252]
    00 00 00         mov       DWORD PTR _Gulliver$[esp+8252], 1
  03ba6  8b 4c 24 20         mov       ecx, DWORD PTR _ulHashPattern$[esp+8252]
  03baa  39 0c 1e   cmp      DWORD PTR [esi+ebx], ecx
  03baf  8d 45 fc   lea      eax, DWORD PTR [ebp-4]
  03bb6  8b 74 24 24         mov       esi, DWORD PTR _pbPattern$GSCopy$[esp+8252]
  03bba  8b 54 24 14         mov       edx, DWORD PTR _i$[esp+8252]
  03bc1  8d 7c 1a 04         lea       edi, DWORD PTR [edx+ebx+4]
  03bc5  8a 0e               mov       cl, BYTE PTR [esi]
  03bc7  3a 0f               cmp       cl, BYTE PTR [edi]
  03bd0  8b 74 24 14         mov       esi, DWORD PTR _i$[esp+8252]
  03bd4  8d 04 1e   lea      eax, DWORD PTR [esi+ebx]
  03be0  8b 74 24 14         mov       esi, DWORD PTR _i$[esp+8252]
  03be4  8d 3c 2b   lea      edi, DWORD PTR [ebx+ebp]
  03be7  0f b7 44 37 fc      movzx     eax, WORD PTR [edi+esi-4]
  03bfb  8a 44 04 38         mov       al, BYTE PTR _bm_Horspool_Order2$[esp+eax+8252]
  03c03  8d 4d fd   lea      ecx, DWORD PTR [ebp-3]
  03c06  89 4c 24 10         mov       DWORD PTR _Gulliver$[esp+8252], ecx
  03c0c  8d 45 ff   lea      eax, DWORD PTR [ebp-1]
  03c0f  89 44 24 10         mov       DWORD PTR _Gulliver$[esp+8252], eax
  03c13  03 74 24 10         add       esi, DWORD PTR _Gulliver$[esp+8252]
  03c17  89 74 24 14         mov       DWORD PTR _i$[esp+8252], esi
  03c1b  3b 74 24 1c         cmp       esi, DWORD PTR tv772[esp+8252]
```

; Main loop of Railgun_Sekireigan_Bari_bit's BMH order 2 section]

www.sanmayce.com **РГ/ϱγ** home!

Railgun_Sekireigan_Bari