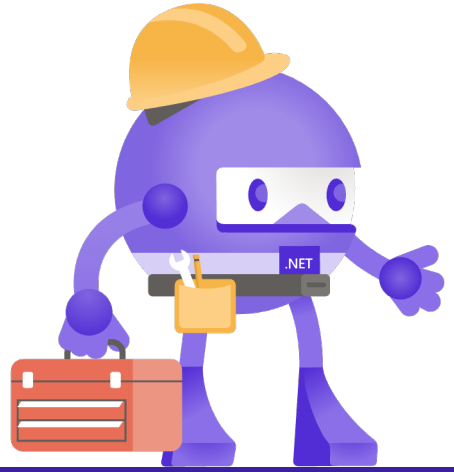


OpenSSL integration in .NET

THE GOOD AND THE CHALLENGING

About me

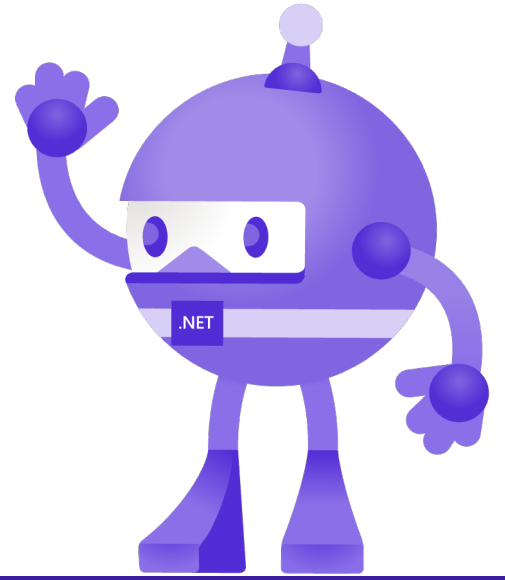
Radek Zikmund

Software Engineer at Microsoft

.NET Networking team

- Current maintainer of System.Net.Security namespace + more
- Contributor to System.Security.Cryptography

rzikm



Agenda

.NET and Linux

OpenSSL vs Schannel

Targeting .NET for Linux distributions

Current technical challenges

.NET and Linux

.NET and Linux

.NET Framework – the original windows-only implementation

.NET Core / .NET – open source, cross platform reimplementation
- dotnet/runtime
- 2016 – first release

.NET relies on platform/OS libraries for cryptography
- Windows – Schannel, Linux – OpenSSL, OSX – Security Framework, …
- Pros
  - Faster security updates, Compliance, FIPS certification
  - Better integration into OS's trust model (certificate stores, etc.)
  - Respecting global system configuration (openssl.cnf)
- Cons
  - Behavioral differences between OS/distro versions
  - Linux: multiple OpenSSL versions to support

OpenSSL vs Schannel

OpenSSL vs Schannel

Most of the .NET API shape has been inherited from .NET Framework

- with Schannel and CAPI as TLS/crypto backends
- Attempt to match the behavior on other platforms (Linux, Mac)

Notable differences

- Architectural difference – storing certificate private keys out-of-process (Schannel) vs in-process (OpenSSL)
  - Mostly matter of documentation
- Single machine store (/etc/ssl/certs) vs multiple stores (My, CA, …)
  - Emulated and stored in ~/.dotnet/corefx/cryptography/x509stores/{StoreName}
- CAPI cert verification routine downloads intermediates and CRLs/OCSP
  - Downloading and caching implemented in C# in PAL layer for Linux for parity

Targeting .NET for Linux

Targeting .NET for Linux

Goal:

- One set of binaries for all Linux distributions
- Enable customers to do the same for their .NET applications self-contained deployments

Challenge: Different linux distros ship with different dependencies version

- glibc, OpenSSL …

Glibc compatibility => build on a system with low glibc version

- .NET 9 (latest release, released in 2024) is built on Ubuntu 16.04 (!!)
- .NET 10 (next LTS release) is built on Ubuntu 18.04

What about OpenSSL?

Targeting .NET for multiple OpenSSL versions

Cannot link statically

- Cons: Security updates, compliance issues (shipping cryptography, FIPS certification)…

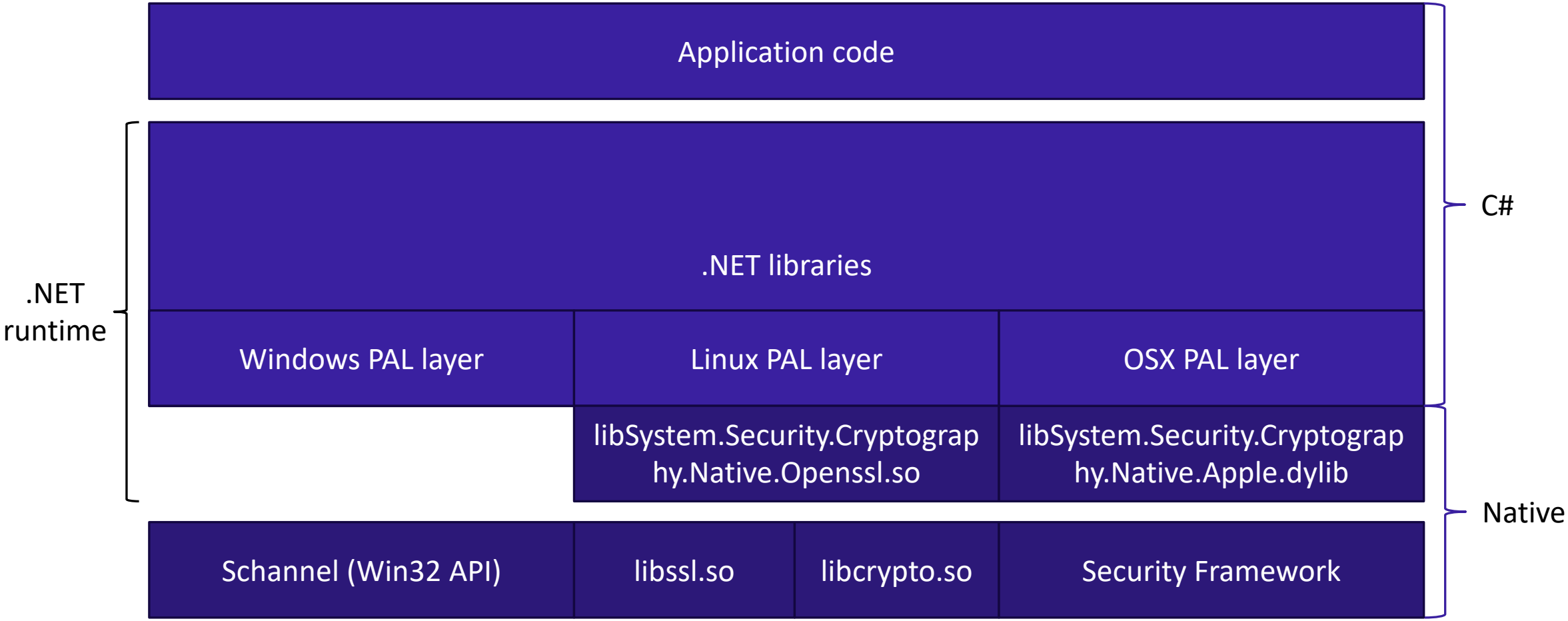
Cannot link dynamically

- Target distro likely ships OpenSSL version

Solution: load and probe for OpenSSL manually on startup

- Intermediate “shim” library that provides unified interface to .NET

OpenSSL integration in .NET



Initialization

src > native > libs > System.Security.Cryptography.Native > C opensslshim.c > ...

```
55  static void DlOpen(const char* libraryName)
56  {
57      void* libsslNew = dlopen(libraryName, RTLD_LAZY);
58
59      // check is someone else has opened and published libssl already
60      if (!pal_atomic_cas_ptr(&libssl, libsslNew, NULL))
61      {
62          dlclose(libsslNew);
63      }
64  }
65
66
67
68
69
70
71
72
73
74

75
76
82
83
84
85
86
87
88
89
90
91
97
98
99
100
101
102
103
104
105
106
107
108
```

src > native > libs > System.Security.Cryptography.Native > C opensslshim.c > ...

```
158  static pthread_once_t g_openLibrary = PTHREAD_ONCE_INIT;
159
160  int OpenLibrary(void)
161  {
162      pthread_once(&g_openLibrary, OpenLibraryOnce);
163
164      if (libssl != NULL)
165      {
166          return 1;
167      }
168      else
169      {
170          return 0;
171      }
172  }
173
174  void InitializeOpenSSLShim(void)
175  {
176      if (!OpenLibrary())
177      {
178          fprintf(stderr, "No usable version of libssl was found\n");
179          abort();
180      }
```

Loading OpenSSL functions

src > native > libs > System.Security.Cryptography.Native > C opensslshim.h > ...

```
306  // List of all functions from the libssl that are used in the System.Security.Cryptography.Native.
307  // Forgetting to add a function here results in build failure with message reporting the function
308  // that needs to be added.
309
310  #define FOR_ALL_OPENSSL_FUNCTIONS \
311      REQUIRED_FUNCTION(a2d_ASN1_OBJECT) \
312      REQUIRED_FUNCTION(ASN1_d2i_bio) \
313      REQUIRED_FUNCTION(ASN1_i2d_bio) \
314      REQUIRED_FUNCTION(ASN1_GENERALIZEDTIME_free) \
315      REQUIRED_FUNCTION(ASN1_INTEGER_get) \
316      REQUIRED_FUNCTION(ASN1_OBJECT_free) \
317      REQUIRED_FUNCTION(ASN1_OCTET_STRING_free) \
318      REQUIRED_FUNCTION(ASN1_OCTET_STRING_new) \
319      REQUIRED_FUNCTION(ASN1_OCTET_STRING_set) \
320      REQUIRED_FUNCTION(ASN1_STRING_dup) \
321      REQUIRED_FUNCTION(ASN1_STRING_free) \
322      REQUIRED_FUNCTION(ASN1_STRING_print_ex) \
323      REQUIRED_FUNCTION(ASN1_TIME_new) \
324      REQUIRED_FUNCTION(ASN1_TIME_set) \
325      FALLBACK_FUNCTION(ASN1_TIME_to_tm) \
326      REQUIRED_FUNCTION(ASN1_TIME_free) \
327      REQUIRED_FUNCTION(BIO_ctrl) \
328      REQUIRED_FUNCTION(BIO_ctrl_pending) \
329      REQUIRED_FUNCTION(BIO_free) \
330      REQUIRED_FUNCTION(BIO_gets) \
331      REQUIRED_FUNCTION(BIO_new) \
332      REQUIRED_FUNCTION(BIO_new_file) \
333      REQUIRED_FUNCTION(BIO_read) \
334      FALLBACK_FUNCTION(BIO_up_ref) \
335      REQUIRED_FUNCTION(BIO_s_mem) \
336      REQUIRED_FUNCTION(BIO_write) \
337      FALLBACK_FUNCTION(BN_abs_is_word) \
338      REQUIRED_FUNCTION(BN_bin2bn) \
339      REQUIRED_FUNCTION(BN_bn2bin) \
340      REQUIRED_FUNCTION(BN_clear_free) \
341      REQUIRED_FUNCTION(BN_cmp) \
342      REQUIRED_FUNCTION(BN_div) \
343      REQUIRED_FUNCTION(BN_dup) \
```

src > native > libs > System.Security.Cryptography.Native > C opensslshim.h > ...

```
864  // Declare pointers to all the used OpenSSL functions
865  #define REQUIRED_FUNCTION(fn) extern TYPEOF(fn)* fn##_ptr;
866  #define REQUIRED_FUNCTION_110(fn) extern TYPEOF(fn)* fn##_ptr;
867  #define LIGHTUP_FUNCTION(fn) extern TYPEOF(fn)* fn##_ptr;
868  #define FALLBACK_FUNCTION(fn) extern TYPEOF(fn)* fn##_ptr;
869  #define RENAMED_FUNCTION(fn,oldfn) extern TYPEOF(fn)* fn##_ptr;
870  #define LEGACY_FUNCTION(fn) extern TYPEOF(fn)* fn##_ptr;
871  FOR_ALL_OPENSSL_FUNCTIONS
872  #undef LEGACY_FUNCTION
873  #undef RENAMED_FUNCTION
874  #undef FALLBACK_FUNCTION
875  #undef LIGHTUP_FUNCTION
876  #undef REQUIRED_FUNCTION_110
877  #undef REQUIRED_FUNCTION
878  #if defined(TARGET_ARM) && defined(TARGET_LINUX)
879  extern TYPEOF(OPENSSL_gmtime)* OPENSSL_gmtime_ptr;
880  #endif
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
```

Loading OpenSSL functions

src > native > libs > System.Security.Cryptography.Native > C opensslshim.c > ...

```
174  void InitializeOpenSSLShim(void)
182      // A function defined in libcrypto.so.1.0.0/libssl.so.1.0.0 that is not defined in
183      // libcrypto.so.1.1.0/libssl.so.1.1.0
184      const void* v1_0_sentinel = dlsym(libssl, "SSL_state");
185
186      // Only permit a single assignment here so that two assemblies both triggering the initializer doesn't cause a
187      // race where the fn_ptr is nullptr, then properly bound, then goes back to nullptr right before being used (then bound again).
188      void* volatile tmp_ptr;
189
190      // Get pointers to all the functions that are needed
191  #define REQUIRED_FUNCTION(fn) \
192      if (!(fn##_ptr = (TYPEOF(fn))(dlsym(libssl, #fn)))) { fprintf(stderr, "Cannot get required symbol " #fn " from libssl\n"); abort(); }
193
```

Fallback functions

Usually used for functions which used to be macros in previous library versions

src > native > libs > System.Security.Cryptography.Native > C apibridge.c > ...

```
10   #ifdef NEED_OPENSSL_1_0
803  unsigned long local_SSL_CTX_set_options(SSL_CTX* ctx, unsigned long options)
804  {
805      // SSL_CTX_ctrl is signed long in and signed long out; but SSL_CTX_set_options,
806      // which was a macro call to SSL_CTX_ctrl in 1.0, is unsigned/unsigned.
807      return (unsigned long)SSL_CTX_ctrl(ctx, SSL_CTRL_OPTIONS, (long)options, NULL);
808  }
809
810  unsigned long local_SSL_set_options(SSL* ssl, unsigned long options)
811  {
812     // SSL_ctrl is signed long in and signed long out; but SSL_set_options,
813     // which was a macro call to SSL_ctrl in 1.0, is unsigned/unsigned.
814     return (unsigned long)SSL_ctrl(ssl, SSL_CTRL_OPTIONS, (long)options, NULL);
815  }
816
817  int local_SSL_session_reused(SSL* ssl)
818  {
819      return (int)SSL_ctrl(ssl, SSL_CTRL_GET_SESSION_REUSED, 0, NULL);
820  }
```

Missing functions

src > native > libs > System.Security.Cryptography.Native > C openssl.c > ...

```
1276
1277  void CryptoNative_RegisterLegacyAlgorithms(void)
1278  {
1279  #ifdef NEED_OPENSSL_3_0
1280      if (API_EXISTS(OSSL_PROVIDER_try_load))
1281      {
1282          OSSL_PROVIDER_try_load(NULL, "legacy", 1);
1283
1284          // Doesn't matter if it succeeded or failed.
1285          ERR_clear_error();
1286      }
1287  #endif
1288  }
1289
1290  int32_t CryptoNative_IsSignatureAlgorithmAvailable(const char* algorithm)
1291  {
1292      int32_t ret = 0;
1293
1294  #if defined(NEED_OPENSSL_3_0) && HAVE_OPENSSL_EVP_PKEY_SIGN_MESSAGE_INIT
1295      if (!API_EXISTS(EVP_PKEY_sign_message_init) ||
1296          !API_EXISTS(EVP_PKEY_verify_message_init))
1297      {
1298          return 0;
1299      }
1300
1301      EVP_SIGNATURE* sigAlg = NULL;
1302
1303      sigAlg = EVP_SIGNATURE_fetch(NULL, algorithm, NULL);
1304      if (sigAlg)
1305      {
1306          ret = 1;
1307          EVP_SIGNATURE_free(sigAlg);
1308      }
1309  #endif
1310
1311      (void)algorithm;
1312      return ret;
1313  }
```

Used for optional features available only in newer OpenSSL versions

Compiling against old OpenSSL headers?

- => need to provide signatures

src > native > libs > System.Security.Cryptography.Native > C osslcompat_30.h > ...

```
118  OSSL_PARAM OSSL_PARAM_construct_end(void);
119  OSSL_PARAM OSSL_PARAM_construct_int(const char *key, int *buf);
120  OSSL_PARAM OSSL_PARAM_construct_int32(const char *key, int32_t *buf);
121  OSSL_PARAM OSSL_PARAM_construct_octet_string(const char *key, void *buf, size_t bsize);
122  OSSL_PARAM OSSL_PARAM_construct_utf8_string(const char *key, char *buf, size_t bsize);
123
124  void OSSL_LIB_CTX_free(OSSL_LIB_CTX*);
125  OSSL_LIB_CTX* OSSL_LIB_CTX_new(void);
126  OSSL_PROVIDER* OSSL_PROVIDER_load(OSSL_LIB_CTX*, const char* name);
127  OSSL_PROVIDER* OSSL_PROVIDER_try_load(OSSL_LIB_CTX*, const char* name, int retain_fallbacks);
128  int OSSL_PROVIDER_unload(OSSL_PROVIDER* prov);
129  int OSSL_STORE_close(OSSL_STORE_CTX* ctx);
130  int OSSL_STORE_eof(OSSL_STORE_CTX* ctx);
131  OSSL_STORE_INFO* OSSL_STORE_load(OSSL_STORE_CTX* ctx);
132  void OSSL_STORE_INFO_free(OSSL_STORE_INFO* info);
133  int OSSL_STORE_INFO_get_type(const OSSL_STORE_INFO* info);
134  EVP_PKEY* OSSL_STORE_INFO_get1_PKEY(const OSSL_STORE_INFO* info);
135  EVP_PKEY* OSSL_STORE_INFO_get1_PUBKEY(const OSSL_STORE_INFO* info);
```

Technical challenges

WHAT COULD GO WRONG?

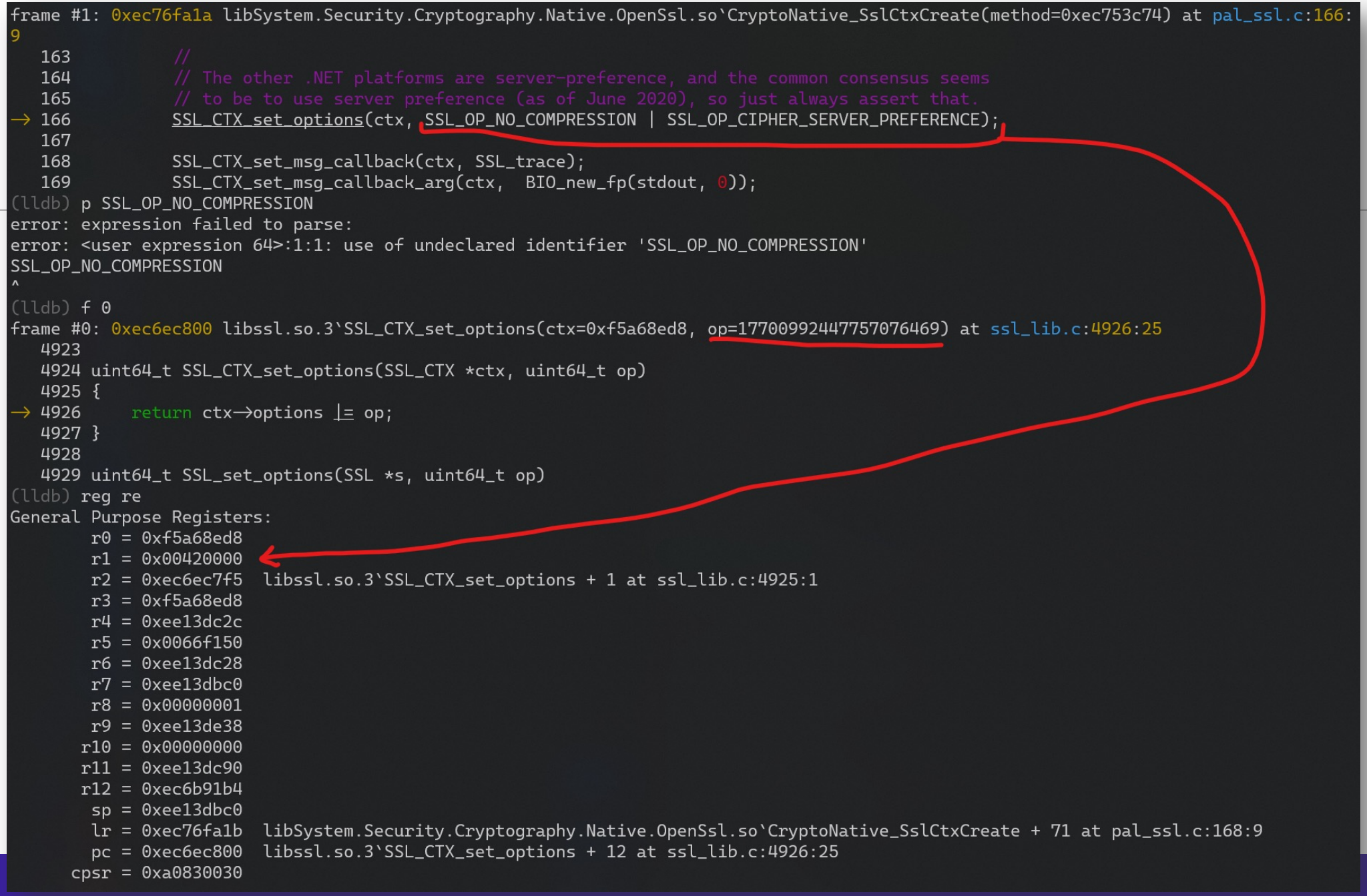
Mysterious bugs

OpenSSL error with Ubuntu 22.04 on Arm32 architecture · Issue #66310 · dotnet/runtime

- “`dotnet build` crashes with `error:0A0000BF:SSL routines::no protocols available`”
- => .NET is unusable on Arm32 Ubuntu 22.04

Root cause:

- parameter to SSL_CTX_set_options changed from 32-bit to 64-bit between OpenSSL 1.1 and 3.0
- .NET was compiled against OpenSSL 1.1 headers, Ubuntu 22.04 has OpenSSL 3.0
- Arm32 calling convention expects 32bit args in registers, 64bit args on stack
- We were passing garbage to OpenSSL


frame #1: 0xec76fa1a libSystem.Security.Cryptography.Native.OpenSsl.so`CryptoNative_SslCtxCreate(method=0xec753c74) at pal_ssl.c:166:9
163 //
164 // The other .NET platforms are server-preference, and the common consensus seems
165 // to be to use server preference (as of June 2020), so just always assert that.
→ 166 SSL_CTX_set_options(ctx, SSL_OP_NO_COMPRESSION | SSL_OP_CIPHER_SERVER_PREFERENCE);
167
168 SSL_CTX_set_msg_callback(ctx, SSL_trace);
169 SSL_CTX_set_msg_callback_arg(ctx, BIO_new_fp(stdout, 0));
(lldb) p SSL_OP_NO_COMPRESSION
error: expression failed to parse:
error: <user expression 64>:1:1: use of undeclared identifier 'SSL_OP_NO_COMPRESSION'
SSL_OP_NO_COMPRESSION
^
(lldb) f 0
frame #0: 0xec6ec800 libssl.so.3`SSL_CTX_set_options(ctx=0xf5a68ed8, op=17700992447757076469) at ssl_lib.c:4926:25
4923
4924 uint64_t SSL_CTX_set_options(SSL_CTX *ctx, uint64_t op)
4925 {
→ 4926 return ctx->options |= op;
4927 }
4928
4929 uint64_t SSL_set_options(SSL *s, uint64_t op)
(lldb) reg re
General Purpose Registers:
r0 = 0xf5a68ed8
r1 = 0x00420000
r2 = 0xec6ec7f5 libssl.so.3`SSL_CTX_set_options + 1 at ssl_lib.c:4925:1
r3 = 0xf5a68ed8
r4 = 0xee13dc2c
r5 = 0x0066f150
r6 = 0xee13dc28
r7 = 0xee13dbc0
r8 = 0x00000001
r9 = 0xee13de38
r10 = 0x00000000
r11 = 0xee13dc90
r12 = 0xec6b91b4
sp = 0xee13dbc0
lr = 0xec76fa1b libSystem.Security.Cryptography.Native.OpenSsl.so`CryptoNative_SslCtxCreate + 71 at pal_ssl.c:168:9
pc = 0xec6ec800 libssl.so.3`SSL_CTX_set_options + 12 at ssl_lib.c:4926:25
cpsr = 0xa0830030

How did we fix it?

src > native > libs > System.Security.Cryptography.Native > C pal_ssl.c > ...

```
43  #ifdef FEATURE_DISTRO_AGNOSTIC_SSL
44  // redirect all SSL_CTX_set_options and SSL_set_options calls via dynamic shims
45  // to work around ABI breaking change between 1.1 and 3.0
46
47  #undef SSL_CTX_set_options
48  #define SSL_CTX_set_options SSL_CTX_set_options_dynamic
49  static uint64_t SSL_CTX_set_options_dynamic(SSL_CTX* ctx, uint64_t options)
50  {
51  #pragma clang diagnostic push
52  #pragma clang diagnostic ignored "-Wcast-function-type"
53      if (API_EXISTS(ERR_new)) // OpenSSL 3.0 sentinel function
54      {
55          // OpenSSL 3.0 and newer, use uint64_t for options
56          uint64_t (*func)(SSL_CTX* ctx, uint64_t op) = (uint64_t(*)(SSL_CTX*, uint64_t))SSL_CTX_set_options_ptr;
57          return func(ctx, options);
58      }
59      else
60      {
61          // OpenSSL 1.1 and earlier, use uint32_t for options
62          uint32_t (*func)(SSL_CTX* ctx, uint32_t op) = (uint32_t(*)(SSL_CTX*, uint32_t))SSL_CTX_set_options_ptr;
63          return func(ctx, (uint32_t)options);
64      }
65  #pragma clang diagnostic pop
66  }
```

Multiple OpenSSL versions in one process

Unsupported in general

MsQuic and QUIC+HTTP/3 support in .NET (since 2022)

- No QUIC-enabling API available in OpenSSL at the time
- MsQuic
  - **Statically** linked against a modified fork of libssl
  - **Dynamically** linked against libcrypto

MsQuic did not load on Ubuntu 22.04 (OpenSSL 1.1 vs 3.0)

- Temporary solution: installing OpenSSL 1.1 libs alongside 3.0 for compatibility
- Long-term solution: OpenSSL 3.0-flavored MsQuic build

Future solution: MsQuic uses OpenSSL 3.5

Current challenges

CRLs and High memory usage

High native memory usage during CRL check for the server not featuring OCSP stapling · Issue #108557 · dotnet/runtime

Scenario – application regularly connecting to an external server with a cert with large (~10MB) CRL

Application consumes GBs of RAM more than expected

Debug tools show no memory leaks

Mall_info dump

```
ÍR"ŸRn ¥"ŸŸ¥"¸Ÿ ƒ"ŸQ:·C"Ÿ.·ı CQ_ ÍPÍR"¸Ÿ CÍRCC "ŸŸ.·"ŸŸPtsc PtsÍHo·:·Ptsı "ŸÍh Ptsç: P⁄ts ģĹH˝ f Ĉ÷‡Pts"ŸŸP"ŸPts}: PtsŨ˝ ũ˘ ĥĈā
éŲũ¥ÕŨ˝/ / ĈāPtsŅ ˘ ÓĈPts˘ ŘŘŲĆ˝ f ĈāPtsOşLĮ[ ĈŅC:·Pts Pts Pts CQ÷‡P"ŸR÷‡Q ÍR÷‡"¸ŸN
ẽ ŨşĈģPtsŲĮĤPtsĤģĈĈPtsÓţ ũőŅ:·Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts ÍP÷‡C¸
ẽ ŨşĈģPtsŲĮĤPtsĤģĈĈPtsĤ˝ ŅĺşĹũPtsşŘŲÓőŅ:·Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts"ŸQ÷‡NC"Ÿ
ẽ ŨşĈģPtsŲĮĤPtsŨŨ˝/ / ĈāPtsģĈĥĹŲũŅ:·Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts C
ª/ ˘ ÓĈPts˘ ŘŘŲĆ˝ f ĈāPtsĹũPtsŨŨ˝/ / ĈāPtsģĈĥĹŲũŅPtsOşLĮ[ ĈŅC:·Pts Pts Ptsı¸ ÍH÷‡"ŸŸN
¯ ŅŨşŘőŅ:·Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts Pts "Ÿ
ª/ ˘ ÓĈPtsĹũPtsĤģĈĈāPtsĤ˝ ŅĺşĹũPtsşŘŲÓőŅPtsOşLĮ[ ĈŅC:·Pts Pts Pts Pts Pts NC"Ÿ¥÷CQ`ŸH
ƒŲĺ ˘ ŘPts˘ ŘŘŲĆ˝ f ĈāPtsŅ ˘ ÓĈPtsOşLĮ[ ĈŅC:·Pts Pts Pts Pts Pts Pts Pts Pts¸ ÷‡Hh ÍR÷‡PCN
ƒŲĺ ˘ ŘPtsĤģĈĈPtsŅ ˘ ÓĈPtsOşLĮ[ ĈŅC:·Pts CQ÷‡P"ŸR÷‡¸ ¸ ÷‡CN"ŸY
ƒĺŲ ¥ÕŲŅĺ ÷‡PtsģĈŘĈ˝ Ņ˝ şŘĈPtsŅ ˘ ÓĈPtsOşLĮ[ ĈŅC:·Pts Pts Pts Pts Pts Pts Pts"ŸC"Ÿ¥÷CQ`ŸH
```

Root cause

Native heap fragmentation

Simple repro

```c
void *allocate_buffers_then_free()
{
    // Allocate an array of NUM_BUFFERS buffers
    char *buffers[NUM_BUFFERS];
    int i;

    for (i = 0; i < NUM_BUFFERS; i++)
    {
        buffers[i] = (char *)malloc(BUFFER_SIZE);

        // Check if malloc was successful
        if (buffers[i] == NULL)
        {
            fprintf(stderr, "Failed to allocate memory for buffer %d\n", i);
            exit(1);
        }

        // Write to buffer in order to assure it's mapped to physical memory
        memset(buffers[i], 0, BUFFER_SIZE);
    }
    printf("Allocated buffers\n");
    dump_mall_info();

    sleep(10);

    // Freeing all the buffers except the last one
    for (i = 0; i < NUM_BUFFERS - 1; i++)
    {
        free(buffers[i]);
    }
    printf("\nFinished freeing buffers\n");
    dump_mall_info();
}
```

```c
#define NUM_BUFFERS 1000000
#define BUFFER_SIZE 1024 // 1KB
// 1000000 buffers * 1KB = ~1GB

void dump_mall_info(void)
{
    struct mallinfo2 mi = mallinfo2();
    printf("Total allocated space: %ld bytes\n", mi.uordblks);
    printf("Total free space: %ld bytes\n", mi.fordblks);
    printf("Total releasable space: %ld bytes\n", mi.keepcost);
}
```

```
❯ gcc ./main.c; ./a.out
Allocated buffers
Total allocated space: 1040000656 bytes
Total free space: 117104 bytes
Total releasable space: 117104 bytes

Finished freeing buffers
Total allocated space: 8976 bytes
Total free space: 1040108784 bytes
Total releasable space: 117104 bytes
```

Root cause

Native heap fragmentation

Large CRL => lots of small objects allocated in X509_CRL object => lots of memory ends up in malloc free lists

.NET async programming model – logical task can move between threads

- Memory associated with one TLS connection is allocated from different threads
- => distributed across different malloc arenas (up to 8x{CPU core count})

Some small allocations keeps malloc from coalescing and releasing free memory

Solution?

No good solution, only workarounds with downsides

Disable CRL checking
- May go against customer's company policies

Lower MALLOC_MAX_ARENA
- Increases lock contention

Replace malloc implementation
- Complicates app deployment

Regularly call malloc_trim()
- Does not work in all cases

Implement internal cache for X509_CRL objects
- Introduces complexity

Q&A