INCLUDE SECURITY

Security Assessment and Fuzzer Improvement for Libtorrent

Sponsored by Mozilla's Secure Open Source Fund

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Background

IncludeSec is an application security assessment focused consultancy founded by application security veterans and Defcon CTF winners in 2010. The team has delivered 1,000+ security assessments for 200+ clients in 30+ programming languages; privately reporting tens of thousands of security concerns to our clients to-date. The team works primarily with technology-oriented clients in the Silicon Valley, San Francisco, and New York City metro areas. We welcome all clients who seek an expert team.

Scope and Methodology

IncludeSec performed a security assessment of the open source codebases for libtorrent as part of Mozilla's Secure Open Source program which allows for Free and Open Source Software(FOSS) to be assessed for security with the overall goal of improving the security posture of the Internet and the FOSS ecosystem. For this project the IncludeSec assessment team was sponsored to execute a 20 day effort spanning from October 13th – November 9th, 2020. The team focused the allocated time on remote attack surface areas primarily. With development and utilization of fuzzing harnesses (libfuzzer), manual code analysis, automated static code analysis, automated fuzzing harness/stub generation, protocol analysis and dynamic analysis all being employed as needed. While there was progress that was made utilizing the technique of automated fuzzing harness generation which satisfied the requirements of the original SOW. The team envisioned an "above and beyond project" and commenced creating a full port of FuzzGen for use with libtorrent on Linux. This was not able to be completed in the allotted time and was indeed beyond the scope of the original SOW. Full details were provided on the progress of this effort for other researchers or the libtorrent team to continue (see Appendices A3 and A4.) The assessment team invites Mozilla and other COTS vendors to sponsor additional work in this regard with the IncludeSec team to finish the FuzzGen porting and framework implementation for libtorrent and other open source projects.

IncludeSec thanks Arvid Norberg from libtorrent for his assistance during the assessment process. Additionally, IncludeSec would like to thank Tom Ritter and the entire Mozilla team for defining this project and sponsoring work to improve the security of Open Source Software.

Assessment Objectives

The objective of this assessment was to identify potential security vulnerabilities within the libtorrent library as well as implement the fuzzer to be function level instead of socket level as much as possible. The team assigned a qualitative risk ranking to each finding. IncludeSec also provided remediation steps which **libtorrent** could implement to secure its applications and systems.

Findings Overview

IncludeSec identified 10 areas of improvement in the code base. Of these, seven are suggested to be addressed immediately and could pose a security or reliability risk. An additional three findings are informational in nature and recommended as future tactical and strategic improvements to minimize the chances of future security problems arising in the applications during future development.

Next Steps

IncludeSec advises libtorrent to remediate as many findings as possible in a prioritized manner and make systemic changes to the Software Development Life Cycle (SDLC) to prevent further vulnerabilities from being introduced into future release cycles. This report can be used by libtorrent as a basis for any SDLC changes. IncludeSec welcomes the opportunity to assist Mozilla or libtorrent in improving the libtorrent SDLC in future engagements by providing security assessments of additional products.

ASSESSMENT RESULTS

At the conclusion of the assessment, Include Security categorized findings into two general groups. The first group “Security and Reliability Findings” comprise of crash cases, design flaws, memory copy stack/heap corruptions, cryptography concerns, compiler concerns, integer overflow/underflows, and memory leaks. The second group “Informational findings future proofing and defense in depth” consist of recommended improvements and risk remediation tactics to prevent future security vulnerabilities as active development continues on the applications.

Within each group the findings are ordered in a prioritized manner with the top issue being presented as the most important in terms of prioritization in the opinion of the security assessment team. The groupings and ordering below are guidelines that IncludeSec believes reflect best practices in the security industry and may differ from what the application author’s perceived risk or prioritization may be. It is common and encouraged that all open-source projects align prioritization based on user security and safety after receipt of these results.

The findings below are listed by a short name (e.g., F1, F2, F3, I1, I2) and a finding title which may reference one or more components in brackets for quick references. Each finding includes: Description, Recommended Remediation, and References as appropriate.

SECURITY AND RELIABILITY FINDINGS

F1: Server-Side Request Forgery (SSRF)

Description:

A Server-Side Request Forgery (SSRF) issue was discovered in the libtorrent library. SSRF issues occur when a user can supply a hostname or IP address to the server which will cause the server to make a request to that host. Attackers can use SSRF vulnerabilities to attack or probe internal network services which are available to the server (but not available externally on the Internet) to attack other services on the Internet or cause requests from the server to be made into an attacker-controlled server enabling the attacker to control the response.

The SSRF exists in the magnet and torrent file functionality, which can make outgoing HTTP or UDP requests to a user-supplied host.

Steps to Reproduce
1. Set up libtorrent to be compiled (see libtorrent compile script setup in Appendix section).
2. Compile examples:

```
cd examples/
b2 clang -j$(nproc)
```

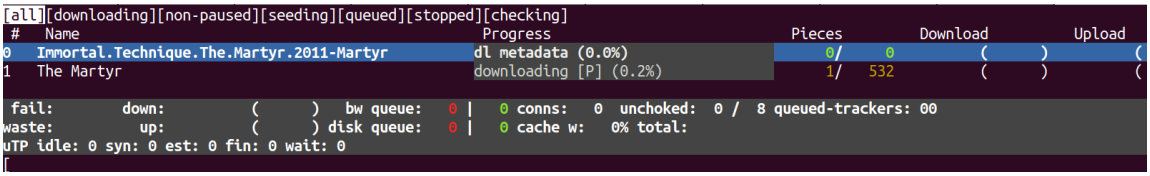
3. Go to **https://thepirattebay.org** in a web browser.
4. Search for Immortal Technique.

5. Click on a torrent that torrents The Martyr album. (Note that “The Martyr” album was released by Immortal Technique/Viper Records free of charge in 2011 and is legal to download)
6. Right click the **GET THIS TORRENT** link and copy the magnet URI.
7. Substitute the UDP tracker domain names with **127.0.0.1**.
8. Execute Wireshark and capture traffic on **localhost**.
9. Execute the **client_test** with the new magnet URI.

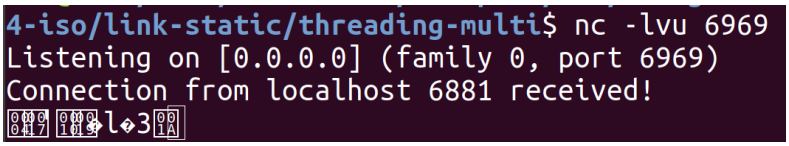
```
./client_test
'magnet:?xt=urn:btih:254DC05696CB2375AE763F565CC48A8C6592A5FD&dn=Immortal.Technique.The.Martyr.2011-
Martyr&tr=udp%3A%2F%2F127.0.0.1%3A6969%2Fannounce&tr=udp%3A%2F%2Flocalhost%3A2850%2Fannounce&tr=udp%3A%2F%2
Flocalhost%3A2920%2Fannounce&tr=udp%3A%2F%2F127.0.0.1%3A1337&tr=udp%3A%2F%2F127.0.0.1%3A6969%2Fannounce'
```

10. Notice that a request was sent to **localhost**.

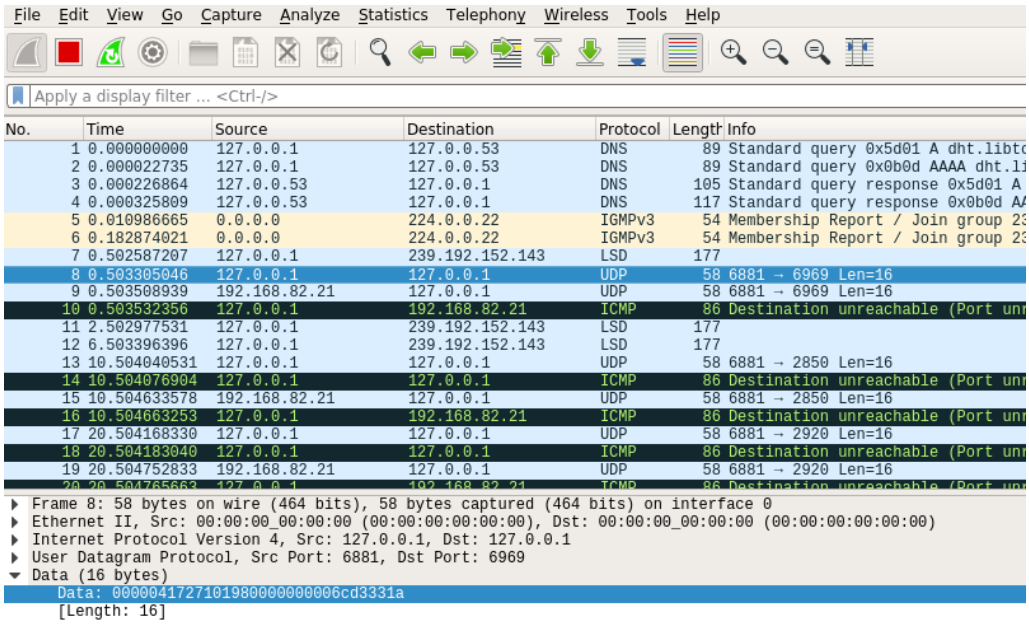
The following is a result of the execution of the client_test libtorrent wrapper:



The following is a screenshot of a netcat listener receiving a UDP connection on localhost:



The following is a wireshark screenshot of localhost request:



Recommended Remediation:

Whenever possible, do not trust user-controllable IPs or URLs when web or network requests need to be made by the client to other services. The code should not be allowed to make requests to localhost or internal network hosts, even through redirects from external hosts. If user-controllable URLs must be requested, then sanitizing them in a manner similar to the SafeUrl library is recommended (see the link in the reference section).

References:

SafeUrl Libraries
Paranoid Request
The Martyr Album

F2: Compile Options Can Remove Assert Security Validation

Description:

The **libtorrent** library conducts several security validations utilizing asserts. These asserts are optional and can be removed with certain compile option flags. Removing some of these assert security checks could make certain memory corruption conditions more easily exploitable.

Affected Location

- **include/libtorrent/config.hpp**:485

The following example that contains **TORRENT_ASSERT** security checks can be found in **libtorrent/src/utp_stream.cpp**.

```
998 std::size_t utp_stream::read_some(bool const clear_buffers)
 999 {
1000         if (m_impl->m_receive_buffer_size == 0)
1001         {
1002                 if (clear_buffers)
1003                 {
1004                         m_impl->m_read_buffer_size = 0;
1005                         m_impl->m_read_buffer.clear();
1006                 }
1007                 return 0;
1008         }
1009
1010         auto target = m_impl->m_read_buffer.begin();
1011
1012         std::size_t ret = 0;
1013
1014         int pop_packets = 0;
1015         for (auto i = m_impl->m_receive_buffer.begin()
1016                 , end(m_impl->m_receive_buffer.end()); i != end;)
1017         {
```

```
1018                if (target == m_impl->m_read_buffer.end())
1019                {
1020                        UTP_LOGV("  No more target buffers: %d bytes left in buffer\n"
1021                                , m_impl->m_receive_buffer_size);
1022                        TORRENT_ASSERT(m_impl->m_read_buffer.empty());
1023                        break;
1024                }
1025
1026 #if TORRENT_USE_INVARIANT_CHECKS
1027                m_impl->check_receive_buffers();
1028 #endif
1029
1030                packet* p = i->get();
1031                int to_copy = std::min(p->size - p->header_size, aux::numeric_cast<int>(target->len));
1032                TORRENT_ASSERT(to_copy >= 0);
1033                std::memcpy(target->buf, p->buf + p->header_size, std::size_t(to_copy));
1034                ret += std::size_t(to_copy);
1035                target->buf = static_cast<char*>(target->buf) + to_copy;
1036                TORRENT_ASSERT(target->len >= std::size_t(to_copy));
1037                target->len -= std::size_t(to_copy);
1038                m_impl->m_receive_buffer_size -= to_copy;
1039                TORRENT_ASSERT(m_impl->m_read_buffer_size >= to_copy);
1040                m_impl->m_read_buffer_size -= to_copy;
1041                p->header_size += std::uint16_t(to_copy);
1042                if (target->len == 0) target = m_impl->m_read_buffer.erase(target);
```

Notice that several **TORRENT_ASSERT** calls check the length of data and verify that **to_copy** is not negative. After compiling libtorrent with default options the following Ghidra decompiled code shows that the **ASSERT** checks were not included.

```
iVar4 = (uint)*(ushort *)(lVar1 + 10) - (uint)*(ushort *)(lVar1 + 0xc);
      iVar5 = (int)ppvVar2[1];
      if (iVar4 <= (int)ppvVar2[1]) {
        iVar5 = iVar4;
      }
      memcpy(*ppvVar2,(void *)(lVar1 + 0xf + (ulong)*(ushort *)(lVar1 + 0xc)),(long)iVar5);
    }
```

The **include/libtorrent/config.hpp** defines the configuration for **TORRENT_ASSERTS**:

```
481 // debug builds have asserts enabled by default, release
482 // builds have asserts if they are explicitly enabled by
483 // the release_asserts macro.
484 #ifndef TORRENT_USE_ASSERTS
485 #define TORRENT_USE_ASSERTS 0
486 #endif // TORRENT_USE_ASSERTS
```

Notice that the default configuration for production builds is to turn **TORRENT_ASSERTS** off.

Recommended Remediation:

If the current code patterns and practices are to remain; the assessment team recommends creating a new assert macro that is always enabled regardless of compile flag options. If a change is welcomed, the team recommends replacing certain ASSERTs with error handling code that can not be configured which could also help mitigate issues.

References:

C Macro Assert

F3: Confidential and Security Relevant Information Stored in Logs

Description:

Security relevant information, specifically, encryption key information is being logged by the **libtorrent** library. Applications utilizing the libtorrent library have the option to enable or disable logging at compile time and can output logs to a specific location on the filesystem. Anyone with access to the logs, including access to log backups, etc. would be able to retrieve the encryption key information.

Affected Locations

- **libtorrent/src/bt_peer_connection.cpp**:590-593
- **libtorrent/src/bt_peer_connection.cpp**:2824-2830

Encryption key information is being logged in the **bt_peer_connection::write_pe3_sync()** function if the **TORRENT_DISABLE_LOGGING** configuration is not set:

BT Peer Connection Encryption Key Logging

```
587 #ifndef TORRENT_DISABLE_LOGGING
588                 if (should_log(peer_log_alert::info))
589                 {
590                         peer_log(peer_log_alert::info, "ENCRYPTION"
591                                 , "writing synchash %s secret: %s"
592                                 , aux::to_hex(sync_hash).c_str()
593                                 , aux::to_hex(secret).c_str());
594                 }
595 #endif
```

Notice that the secret is logged by the **peer_log()** function on line 593.

Recommended Remediation:

The assessment team recommends not logging key information. The logging section of the code could be removed or generic information could be logged instead of key information.

References:

Information Exposure Through Log Files

F4: Pseudo Random Number Generator Is Vulnerable to Prediction Attack

Description:

An attacker may be able to predict future values by monitoring the deterministic random number generator.

Currently the libtorrent application produces pseudo random bytes for use in Diffie Helman key generation, elliptic curve diffie helman key generation, pe_crypto (custom encryption), cookie values, padding, universal plug and play (UPNP) port selection, and connection IDs using a pseudo random number generator (PRNG) that does not provide high entropy for cryptographic operations (Mersenne Twister 19937 generator).

Exploiting this situation would involve observing network traffic until the attacker is able to determine the internal state of the pseudo random number generator, after which further pseudo random numbers would be predictable. An attacker could utilize this information to decrypt encrypted data or predict bittorrent protocol information that could aid an attacker in a network based attack.

Affected Locations

- libtorrent/src/random.cpp:127,138,150,155,160

The **random_bytes()** function is being used to generate key material and is being called from **bt_peer_connection.cpp** file:

bt_peer_connection.cpp

```
 503                 char msg[dh_key_len + 512];
 504                 char* ptr = msg;
 505                 int const buf_size = int(dh_key_len) + pad_size;
 506
 507                 std::array<char, dh_key_len> const local_key = export_key(m_dh_key_exchange-
>get_local_key());
 508                 std::memcpy(ptr, local_key.data(), dh_key_len);
 509                 ptr += dh_key_len;
 510
 511                 aux::random_bytes({ptr, pad_size});
 512                 send_buffer({msg, buf_size});
```

random.cpp

```
80 namespace aux {
81
82                 std::mt19937& random_engine()
83                 {
84 #ifdef TORRENT_BUILD_SIMULATOR
85                         // make sure random numbers are deterministic. Seed with a fixed number
86                         static std::mt19937 rng(0x82daf973);
87 #else
88
89 #if TORRENT_BROKEN_RANDOM_DEVICE
```

```
90                          struct {
91                                  std::uint32_t operator()() const
92                                  {
93                                          static std::atomic<std::uint32_t>
seed{static_cast<std::uint32_t>(duration_cast<microseconds>(
94
std::chrono::high_resolution_clock::now().time_since_epoch()).count())};
95                                          return seed++;
96                                  }
97                          } dev;
98 #else
99                          static std::random_device dev;
100 #endif
...
110                 void random_bytes(span<char> buffer)
111                 {
112 #ifdef TORRENT_BUILD_SIMULATOR
113                         // simulator
114
115                         std::generate(buffer.begin(), buffer.end(), [] { return char(random(0xff)); });
116
117 #elif TORRENT_USE_CNG
118                         aux::cng_gen_random(buffer);
119 #elif TORRENT_USE_CRYPTOAPI
120                         // windows
121
122                         aux::crypt_gen_random(buffer);
123
124 #elif TORRENT_USE_DEV_RANDOM
125                         // /dev/random
126
127                         static dev_random dev;
128                         dev.read(buffer);
129
130 #elif defined TORRENT_USE_LIBCRYPTO
131
132 #if defined TORRENT_USE_WOLFSSL
133 // wolfSSL uses wc_RNG_GenerateBlock as the internal function for the
134 // openssl compatibility layer. This function API does not support
135 // an arbitrary buffer size (openssl does), it is limited by the
136 // constant RNG_MAX_BLOCK_LEN.
137 // TODO: improve calling RAND_bytes multiple times, using fallback for now
138                         std::generate(buffer.begin(), buffer.end(), [] { return char(random(0xff)); });
139 #else // TORRENT_USE_WOLFSSL
140                         // openssl
141
142                         int r = RAND_bytes(reinterpret_cast<unsigned char*>(buffer.data())
143                                 , int(buffer.size()));
144                         if (r != 1) aux::throw_ex<system_error>(errors::no_entropy);
145 #endif
146
147 #else
148                         // fallback
149
150                         std::generate(buffer.begin(), buffer.end(), [] { return char(random(0xff)); });
...
155         std::uint32_t random(std::uint32_t const max)
156         {
157 #ifdef BOOST_NO_CXX11_THREAD_LOCAL
158                 std::lock_guard<std::mutex> l(rng_mutex);
159 #endif
160                 return std::uniform_int_distribution<std::uint32_t>(0, max)(aux::random_engine());
161         }
```

Notice that the **random_bytes()** function calls **random()** in several locations which leverages the Mersenne Twister 19937 algorithm (e.g. std::mt19937) which is considered a PRNG and not a cryptographically security PRNG (CSPRNG). Once an attacker can gather several generated numbers, then they can predict future values. See the references section for more information on predictability and cryptographic weaknesses. If the **TORRENT_USE_CRYPTOAPI** (windows only) is enabled or if **TORRENT_USE_CNG** is enabled, then the **random()** function would not be called by **random_bytes()** and the compiled instance of the library would not be affected.

The following are additional locations of where the **random_bytes()** function is called:

- kademlia/node_id.cpp:159: aux::random_bytes(r);
- kademlia/ed25519.cpp:43: aux::random_bytes(seed);
- web_peer_connection.cpp:144: aux::random_bytes(pid);
- http_seed_connection.cpp:83: aux::random_bytes(pid);
- bt_peer_connection.cpp:550: aux::random_bytes({ptr, pad_size});
- bt_peer_connection.cpp:697: aux::random_bytes(write_buf.first(pad_size));
- natpmp.cpp:364: aux::random_bytes(inonce);
- pe_crypto.cpp:89: aux::random_bytes({reinterpret_cast<char*>(random_key.data())

The **node::node()** and **node::new_write_key()** functions in **libtorrent/src/kademlia/node.cpp** call the **random()** function to generate a secret:

```
116 node::node(aux::listen_socket_handle const& sock, socket_manager* sock_man
 117         , aux::session_settings const& settings
 118         , node_id const& nid
 119         , dht_observer* observer
 120         , counters& cnt
 121         , get_foreign_node_t get_foreign_node
 122         , dht_storage_interface& storage)
 123         : m_settings(settings)
 124         , m_id(calculate_node_id(nid, sock))
 125         , m_table(m_id, aux::is_v4(sock.get_local_endpoint()) ? udp::v4() : udp::v6(), 8, settings,
observer)
126         , m_rpc(m_id, m_settings, m_table, sock, sock_man, observer)
127         , m_sock(sock)
128         , m_sock_man(sock_man)
129         , m_get_foreign_node(std::move(get_foreign_node))
130         , m_observer(observer)
131         , m_protocol(map_protocol_to_descriptor(aux::is_v4(sock.get_local_endpoint()) ? udp::v4() :
udp::v6()))
132         , m_last_tracker_tick(aux::time_now())
133         , m_last_self_refresh(min_time())
134         , m_counters(cnt)
135         , m_storage(storage)
136 {
137         m_secret[0] = random(0xffffffff);
138         m_secret[1] = random(0xffffffff);
139 }
...
253 void node::new_write_key()
254 {
255         m_secret[1] = m_secret[0];
```

```
256         m_secret[0] = random(0xffffffff);
257 }
```

Notice that the **random()** function is called and **node::new_write_key()** utilizes one **random()** call.

The **random()** function from libtorrent/src/random.cpp is called directly in several locations as well. The following is a subset of those calls:

- kademlia/node_id.cpp:143: if (secret == 0) secret = random(0xfffffffe) + 1;
- kademlia/node_id.cpp:145: std::uint32_t const rand = random(0xffffffff);
- kademlia/node_id.cpp:194: return generate_id_impl(ip, random(0xffffffff));
- kademlia/rpc_manager.cpp:472: auto const tid = static_cast<std::uint16_t>(random(0x7fff));
- lsd.cpp:81: , m_cookie((random(0x7fffffff) ^ std::uintptr_t(this)) & 0x7fffffff)
- piece_picker.cpp:1037: int(random(aux::numeric_cast<std::uint32_t>(static_cast<int>(range.second – range.first) – 1)))
- generate_peer_id.cpp:52: url_random(span<char>(ret).subspan(std::ptrdiff_t(print.length())));
- udp_tracker_connection.cpp:428: std::uint32_t const new_tid = random(0xfffffffe) + 1;
- ut_metadata.cpp:448: m_request_limit = now + seconds(20 + random(50));
- bt_peer_connection.cpp:651: int const pad_size = int(random(512));
- torrent.cpp:9947: int const pick = int(random(aux::numeric_cast<std::uint32_t>(rarest_pieces.end_index() – 1)));
- utp_socket_manager.cpp:337: send_id = std::uint16_t(random(0xffff));
- session_impl.cpp:1964: ? int(random(63000) + 2000)
- session_impl.cpp:5045: match = with_gateways[random(std::uint32_t(with_gateways.size() – 1))];
- ip_voter.cpp:128: if (random(1)) return maybe_rotate();
- utp_stream.cpp:2624: m_seq_nr = std::uint16_t(random(0xffff));
- smart_ban.cpp:76: , m_salt(random(0xffffffff))
- peer_list.cpp:328: int round_robin = aux::numeric_cast<int>(random(std::uint32_t(m_peers.size() – 1)));
- upnp.cpp:1449: m.external_port = 40000 + int(random(10000));

Steps to Demonstrate MT19937 Weaknesses

1. Install g++ and mersenne-twister-predictor

```
sudo apt-get install g++
sudo pip install mersenne-twister-predictor
```

2. Download and compile MT19937 C++ PRNG utilization code similar to libtorrent random.cpp implementation from the Appendix

3. Compile poc_generate_mt19937.cpp

```
g++ poc_generate_mt19937.cpp
```

4. Run binary to generate 1000 random numbers and put output to a file

```
./a.out > 1000_rand_numbers.txt
```

5. Output the first 624 random numbers to a file

```
head -n 624 1000_rand_numbers.txt > first_624_numbers.txt
```

6. Output the last 376 to a file

```
tail -n 376 rand_numbers.txt > last_376_numbers.txt
```

7. Utilize mt19937predict to reverse the mt19937 output values, set the internal state of the mt19937 algorithm for future prediction and output the next 376 predicted values to a file

```
cat first_624_numbers.txt | mt19937predict | head -n 376 > next_predicted_376.txt
```

8. Verify that the predicted values were accurate

```
diff next_predicted_376.txt last_376_numbers.txt
```

Recommended Remediation:

Consider using a stronger pseudo random number generator (e.g. CSPRNG) for all compile options (e.g. TORRENT_USE_WOLFSSL and default "//fallback" else case shown above). Utilizing a stronger pseudo random number generator can make it more difficult for an attacker to passively defeat encryption in transit produced by the libtorrent library.

On Linux getrandom() or /dev/urandom would provide stronger alternatives than MT19937.

References:

Wikipedia: Cryptographically secure pseudorandom number generator
Mersenne Twister 19937 generator
Mersenee Twister Security
Attacking a Random Number Generator
Cracking Phobos UUID
getrandom
Breaking PHPs MT_RAND() With 2 Values and No Bruteforce
random
Identifying Security Vulnerabilities in C Programming
/dev/random Wiki
Myths About /dev/urandom
Mersenne Twister Predictor

F5: Potential Null Pointer Dereferences lead to Program Crashes

Description:

The assessment team identified potential null pointer dereference vulnerabilities in the **libtorrent** library. A null-pointer dereference takes place when a pointer with a value of NULL is used as though it points to a valid memory area. This will cause the program to access an invalid memory address and usually results in a process termination (i.e. crash.)

Affected Locations

- **libtorrent/src/mmap_disk_io.cpp**:715
- **libtorrent/src/mmap_disk_io.cpp**:793
- **libtorrent/src/mmap_disk_io.cpp**:807
- **libtorrent/src/mmap_disk_io.cpp**:819
- **libtorrent/src/mmap_disk_io.cpp**:851
- **libtorrent/src/mmap_disk_io.cpp**:866
- **libtorrent/src/mmap_disk_io.cpp**:905
- **libtorrent/src/mmap_disk_io.cpp**:924

The following code in **mmap_disk_io.cpp** allocates a job and then sets values for the job. However, if the job returns a **nullptr**, then a null pointer dereference could occur:

(src/mmap_disk_io.cpp)

```
712                aux::disk_io_job* j = m_job_pool.allocate_job(aux::job_action_t::read);
713                j->storage = m_torrents[storage]->shared_from_this();
714                j->piece = r.piece;
715                j->d.io.offset = r.start;
716                j->d.io.buffer_size = std::uint16_t(r.length);
717                j->flags = flags;
718                j->callback = std::move(handler);
719
720                if (j->storage->is_blocked(j))
721                {
722                        // this means the job was queued up inside storage
723                        m_stats_counters.inc_stats_counter(counters::blocked_disk_jobs);
724                        DLOG("blocked job: %s (torrent: %d total: %d)\n"
725                                , job_name(j->action), j->storage ? j->storage->num_blocked() : 0
726                                , int(m_stats_counters[counters::blocked_disk_jobs]));
727                }
728                else
729                {
730                        add_job(j);
731                }
```

(src/disk_job_pool.cpp)

```
53         disk_io_job* disk_job_pool::allocate_job(job_action_t const type)
54         {
55                 std::unique_lock<std::mutex> l(m_job_mutex);
```

```
56                 void* storage = m_job_pool.malloc();
57                 m_job_pool.set_next_size(100);
58                 if (storage == nullptr) return nullptr;
59                 ++m_jobs_in_use;
60                 if (type == job_action_t::read) ++m_read_jobs;
61                 else if (type == job_action_t::write) ++m_write_jobs;
62                 l.unlock();
63                 TORRENT_ASSERT(storage);
64
65                 auto ptr = new (storage) disk_io_job;
66                 ptr->action = type;
67 #if TORRENT_USE_ASSERTS
68                 ptr->in_use = true;
69 #endif
70                 return ptr;
71         }
```

Notice that if **m_job_pool.malloc()** returns a null pointer then the function will return **nullptr**. Therefore, in the parent function **j** has the potential to be equal to **nullptr** when it is dereferenced.

Recommended Remediation:

The assessment team recommends checking that the value is not **nullptr** before dereferencing. Checking that the pointer is not **nullptr** before dereferencing it could help mitigate against crashes or invalid memory access.

References:

CWE-476 NULL Pointer Dereference
OWASP Null Pointer Dereference
nullptr in cpp
Difference between null and nullptr

F6: Integer Overflow in bdecode

Description:

The assessment team identified an integer overflow while fuzzing. While it is likely that this issue is already known by the **libtorrent** development team, the security assessment team is documenting it here for completeness. **The parse_int()** function in the bdecode feature can lead to an integer overflow. Integer overflow conditions can lead to misinterpretations of data and length calculations, which can result in potential crashes or memory corruption.

Affected Location

- **libtorrent/src/bdecode.cpp**:171

The following source code of bdecode shows that **val** is multiplied by 10 each iteration and could potentially overflow if **val** is negative. There is a check for the overflow but the check could be improved.

```
char const* parse_int(char const* start, char const* end, char delimiter
        , std::int64_t& val, bdecode_errors::error_code_enum& ec)
{
        while (start < end && *start != delimiter)
        {
                if (!numeric(*start))
                {
                        ec = bdecode_errors::expected_digit;
                        return start;
                }
                if (val > std::numeric_limits<std::int64_t>::max() / 10)
                {
                        ec = bdecode_errors::overflow;
                        return start;
                }
                val *= 10;
                int digit = *start - '0';
                if (val > std::numeric_limits<std::int64_t>::max() - digit)
                {
                        ec = bdecode_errors::overflow;
                        return start;
                }
                val += digit;
                ++start;
        }
        return start;
}
```

Steps to Reproduce (PoC File)

1. Download and compile the libtorrent library on Ubtunu (see the automated build script for more information on this).

2. Compile the examples:

```
cd examples
b2 clang -j$(nproc)
```

3. Output a test case to a file to reproduce this issue:

```
echo -n
'MTE1NTAxMDA2NzAwMTAwMzY2MzYxNjMzM8f///////8xMzY2MzM2MzY2MzMy////////////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAA==' | base64 -d > parse_int-poc
```

4. Run the **dump_bdecode** example on the test case:

```
./dump_bdecode parse_int-poc
```

Notice that an integer overflow was detected.

Steps to Reproduce (Fuzzing)

1. Compile and setup libtorrent library for fuzzing (see Appendix section for fuzzing setup)

2. Run the libtorrent fuzzer suite.

```
cd libtorrent/fuzzers/
./run.sh
```

3. Notice that undefined behavior sanitizer detects an integer overflow at line 171 in **src/bdecode.cpp**:

```
../src/bdecode.cpp:171:8: runtime error: signed integer overflow: -5764607523034234880 * 10 cannot be
represented in type 'long'
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior ../src/bdecode.cpp:171:8 in
MS: 3 EraseBytes-InsertRepeatedBytes-CMP- DE: "\x01\x00\x00\x00"-; base unit:
05e3d82de42944f2d82b079743f55f19b06d71ca
0x30,0x36,0x37,0x0,0x0,0x0,0x1,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,
0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,
067\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
artifact_prefix='./parse_int-'; Test unit written to ./parse_int-crash-
ff7a0e7c251522fdf17f95e6c84161fed8565bcd
Base64:
MDY3AAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Additionally, see the **Appendix – Integer Overfow bdecode Test Script** for more information on integer limits and the bdecode integer overflow conditions.

Recommended Remediation:

The assessment team recommends checking additional integer overflow and signed value conditions and checking for the condition where **val** or **digit** are negative.

In general, it is best practice to utilize fixed-width unsigned integer datatypes where the size of the datatype is the same across compilers and architectures (e.g. uint32_t). The **val** variable is set to a fixed width signed 64-bit integer, however the **digit** variable is set to a signed integer.

References:

Integer Overflow
C++ Integer Types

F7: Magnet URIs Allow IDNA Domain Names

Description:

Internationalized Domain Names in Applications (IDNA), also known as punycode, are a way to represent domain names that include characters outside of the ASCII range (e.g. UTF-8 characters). Magnet URIs can provide domain names of trackers and punycode domain names are accepted in the parsing of magent URIs by the **libtorrent** library. These type of characters could lead to homograph (or homoglyph) attacks if the client application displayed the UTF-8 IDNA representation within the graphical user interface. For example, a torrent application could display a magent URI in the GUI and present the URL in a way that could deceive the user into thinking that a tracker is a trusted domain. When the torrent application leverages the libtorrent library to perform network requests the libtorrent library would not block this request. It could be argued that it is the responsibility of the

client application to protect against attacks like this but it is best practice to prohibit these types of URIs.

Steps to Reproduce

1. Download and setup libtorrent build environment (see Appendix for build steps)

2. Compile the examples directory:

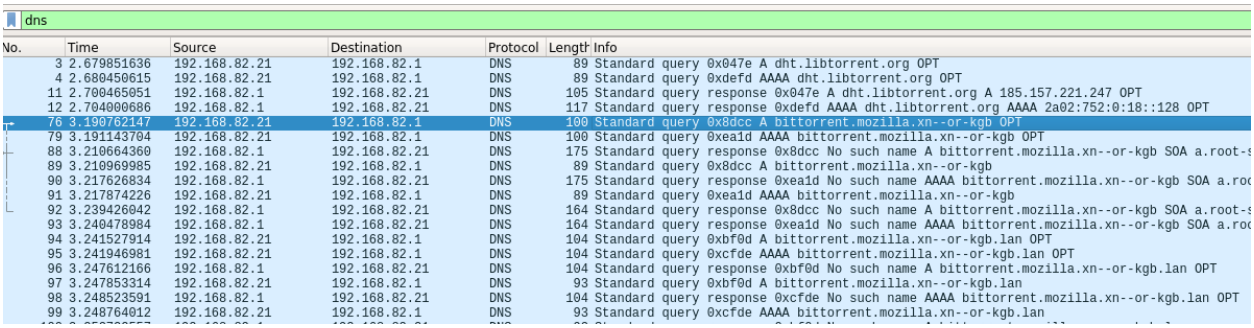
```
cd examples
b2 cxxstd=14 -j$(nproc)
```

3. Run Wireshark and capture traffic on the network interface.

4. Run the **client_test** application with a punycode domain:

```
./client_test 'magnet:?xt=urn:btih:BF6C336ADE3D01A5B78BA58D9FAF078260F53701&dn=Immortal%20Technique%20-
%20The%20Martyr-2011-MIXFIEND&tr=udp%3A%2F%2Fbittorrent.mozilla.xn--or-
kgb%3A6969%2Fannounce&tr=udp%3A%2F%2Fbittorrent.mozilla.xn--or-
kgb%3A2850%2Fannounce&tr=udp%3A%2F%2Fbittorrent.mozilla.xn--or-
kgb%3A2920%2Fannounce&tr=udp%3A%2F%2Fbittorrent.mozilla.xn--or-
kgb%3A1337&tr=udp%3A%2F%2Fbittorrent.mozilla.xn--or-kgb%3A6969%2Fannounce'
```

5. Notice that the client and library process the domain name and make a DNS request.



Recommended Remediation:

The assessment team recommends not allowing punycode (IDNA) domain names and throwing an error if these characters are detected within the domain name in a magnet URI. If this feature is desired, then consider adding additional documentation could be provided for developers utilizing the libtorrent library and suggest not displaying the UTF-8 representation of the characters.

It is recommended that software tools that are leveraged within the torrent ecosystem (e.g. Firefox) disable support for IDNA by default (i.e. set network.IDN_show_punycode to true by default). This would help mitigate against attacks that subvert HTTPS.

References:

IDN Homograph Attack
Out of Character Use of Punycode and Homoglyph Attacks to Obfuscate URLs for Phishing
Irongeek Homoglyph Attack Generator
IDN Phishing
Apple.com Punycode Example

INFORMATIONAL FINDINGS FUTURE PROOFING AND DEFENSE IN DEPTH

I1: Additional Documentation and Automation

Description:

The **libtorrent** documentation provides build information as well as information about fuzzing that could be improved to make the project more approachable to outside contribution and security review. While the documentation on building and fuzzing far exceeds many other open-source repositories, the assessment team has the suggestions below for improvement.

The current build documentation describes several methods for building the **libtorrent** project on several different platforms (e.g. Linux, Windows, Mac). The **libtorrent** build documentation discusses the different compile options and some of the commands involved. It does not cover in detail the operating system dependencies. The build documentation also switches between different operating systems and does not give all the commands necessary for Linux.

Recommended Remediation:

The assessment team recommends creating a list of commands for each operating system (e.g. Ubuntu, Mac, Windows) and documenting the **libtorrent** build process separately for each operating system from start to finish assuming default installations of each OS. In addition, creating a Dockerfile that can be utilized for building libtorrent or fuzzing libtorrent would be helpful. There is currently already a libtorrent Dockerfile and build script that can be found on the OSSFuzz repository that could be adapted and incorporated into the **libtorrent** project.

The assessment team created a build script for Ubuntu and added it to Appendix sections of this report.

References:

Libtorrent Building
Libtorrent OSSFuzz

I2: Automated Fuzzer Generation

Description:

The assessment team spent some time trying to automate the development of fuzzer stubs by leveraging the FuzzGen toolset. The assessment team made progress getting FuzzGen to work with the **libtorrent** library but within the predefined project timeline was not able to fully generate the fuzzing stubs using FuzzGen in the time allotted for the assessment. Additional time would be necessary to utilize FuzzGen on the libtorrent library and the team recommends Mozilla or other organizations

sponsor further work towards that goal. However, information about the setup is provided below and may hopefully help future developers or security researchers auditing libtorrent or similar libraries. Note that the documentation below is a best effort to document the commands executed but might not be an exact representation of the execution flow during testing.

Steps to Reproduce

0. Install dependencies:

```
sudo apt-get install git gcc g++ cmake clang libssl-dev

#bear dependencies
apt-get install python cmake pkg-config
apt-get install libfmt-dev libspdlog-dev nlohmann-json3-dev
apt-get install libgrpc++-dev protobuf-compiler-grpc libssl-dev
```

1. Download **LLVM-7** source code:

```
wget https://github.com/llvm/llvm-project/releases/download/llvmorg-7.1.0/llvm-7.1.0.src.tar.xz
tar xf 'llvm-7.1.0.src.tar.xz'
mv 'llvm-7.1.0.src' LLVM
```

2. Download FuzzGen:

```
git clone https://github.com/HexHive/FuzzGen.git
```

3. Download libtorrent:

```
git clone --recurse-submodules https://github.com/arvidn/libtorrent.git
```

4. Download boost source code:

```
wget https://dl.bintray.com/boostorg/release/1.74.0/source/boost_1_74_0.tar.gz
tar xzf xzf boost_1_74_0.tar.gz
```

5. Download Bear (latest version is required):

```
git clone https://github.com/rizsotto/Bear.git
```

6. Compile Bear (**https://github.com/rizsotto/Bear/blob/master/INSTALL.md**):":https://github.com/rizsotto/Bear/blob/master/INSTALL.md*):

```
cd Bear
mkdir build
cd build
cmake -DENABLE_UNIT_TESTS=OFF -DENABLE_FUNC_TESTS=OFF ../
make -j$(nproc)
cd ../../
```

7. Compile boost with Bear tools to get **compile_commands.json** database (note the two-step approach is utilized to convert the new **llvm compile_commands.json** format to the old LLVM format for LLVM 7 support):

```
$PWD/boost_1_74_0/bootstrap.sh -with-toolset=clang
$PWD/Bear/build/stage/bin/intercept --output commands.json -- $PWD/boost_1_74_0/b2 toolset=clang
cxxflags="-save-temps -S -emit-llvm -m64"
sudo $PWD/boost_1_74_0/b2 install
sudo ln -s $PWD/boost_1_74_0/b2 /usr/local/bin/b2

#create a file named config.json with the following contents in it
{
  "compilation": {
  },
  "output": {
    "content": {
      "include_only_existing_source": true
    },
    "format": {
      "command_as_array": false,
      "drop_output_field": false
    }
  }
}

$PWD/Bear/build/stage/bin/citnames --input commands.json --ouput compile_commands.json --config config.json
```

8. Compile libtorrent with llvm IR output options:

```
echo 'using clang : 6 : clang++-6.0 ;' >> ~/user-config.jam
cd libtorrent
echo "export BOOST_ROOT=$PWD/" >> ~/.bashrc
echo "export BOOST_BUILD_PATH=$PWD/tools/build/" >> ~/.bashrc
export BOOST_ROOT=$PWD/
export BOOST_BUILD_PATH=$PWD/tools/build/
mkdir build
cd build
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=ON -cflags='cxxstd=14 -save-temps -S -emit-llvm -m64'
make -j$(nproc)
```

9. Compile libtorrent examples with llvm IR output options:

```
cd ../examples/
mkdir build
cd build
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=ON -cflags='cxxstd=14 -save-temps -S -emit-llvm -m64'
make -j$(nproc)
```

10. Compile FuzzGen preprocessor and LLVM:

```
cp -r FuzzGen/src/preprocessor/ LLVM/tools/clang/tools/fuzzgen-preprocessor/
echo 'add_clang_subdirectory(fuzzgen-preprocessor)' >> LLVM/tools/clang/tools/CMakeLists.txt
cd LLVM
mkdir build
cd build
cmake -DLLVM_ENABLE_PROJECTS="clang" -DLLVM_USE_LINKER=gold -DCMAKE_BUILD_TYPE=Release ../
```

```
make -j$(nproc)
cd ../../
```

11. Merge bitcode (with custom python script or manually):

```
#custom python script llvm_bitcode_merge.py
import os
import subprocess
import sys
project_folder=sys.argv[1]
src_dir=os.path.join(os.getcwd(),project_folder,"build")
result = []
for i in os.listdir(src_dir):
    if ".bc" in i:
        result.append(src_dir+"/"+i)
print (result)
subprocess.Popen(["./llvm-link"]+result+["-o","./merged.bc"])

python llvm_bitcode_merge.py libtorrent
mv merged.bc merged-libtorrent.bc
cd libtorrent
python ../llvm_bitcode_merge.py examples
mv merged.bc ../merged-examples.bc
cd ..
python llvm_bitcode_merge.py boost_1_74_0
mv merged.bc merged-boost.bc
```

12. Merge LLVM bitcode:

```
llvm-dis merged-boost.bc -o merged.ll
llvm-dis merged-libtorrent.bc -o merged2.ll
llvm-dis merged-examples.bc -o merged3.ll
```

13. Run fuzzgen-preprocessor (for libtorrent and boost) (https://github.com/HexHive/FuzzGen):):

```
$PWD/LLVM/build/bin/fuzzgen-preprocessor -outfile=libtorrent.meta -library-root=$PWD/libtorrent
$PWD/libtorrent/src/
$PWDLLVM/build/bin/fuzzgen-preprocessor -outfile=libtorrent.meta -library-root=$PWD/boost_1_74_0/ -p
$PWD/boost_1_74_0/ $PWD/boost_1_74_0/
```

14. Run FuzzGen:

```
mkdir fuzzer-libtorrent
./fuzzgen -mode=debian -analysis=basic -arch=x64 -no-progressive -lib-name=libtorrent -meta=libtorrent.meta
-lib-root=$PWD/libtorrent -consumer-dir=$PWD/libtorrent/example -path=$PWD/boost_1_74_0/ merged.ll -
outdir=./fuzzer-libtorrent -static-libs='libtorrent.a'
```

Recommended Remediation:

It is not clear as to the exact reason why FuzzGen did not run properly. **Ubuntu 18.04** was leveraged in testing but **Ubuntu 20.04** might be needed to provide a smoother setup.

Note that there is an alternative tool called FUDGE that performs similar tasks to FuzzGen but utilizes the Clang AST instead of utilizing the IR. This tool however is closed source and utilized internally at Google. Perhaps if the FUDGE tool were open sourced in the future or a collaboration effort with

Google were possible, then the FUDGE tool could be leveraged to generate libfuzzer fuzzing stubs for libtorrent.

References:

FuzzGen
FuzzGen Automatic Fuzzer Generation
FuzzGen Usenix
FUDGE: Fuzz Driver Generation at Scale

I3: Type Confusion and Integer Overflow Improvements

Description:

The libtorrent library processes untrusted network and file data. While doing so it commonly leverages signed integer datatypes and datatype conversions/casting. Converting signed integers to an unsigned integer or unsigned integer to a signed integer can often leads to security concerns when these numbers are used in memory allocation related functionality (see references section for more information).

Potential Areas of Improvement

- **src/utp_stream.cpp**:578-598
- **src/udp_socket.cpp**:189,195
- **src/utp_stream.cpp**:2558,2559
- **src/stack_allocator.cpp**:920,929,930

The following section of code is found in **src/utp_stream.cpp** and leverages integer conversions and values. If the integer values were negative and the **TORRENT_ASSERT** was not issued, then a potential memory corruption issue could occur. It does appear that the following code leverages the **numeric_cast** call which partially mitigates the type conversion/confusion issue since **numeric_cast** will throw an exception if there is a positive or negative conversion overflow. However, it would be best practice to assure that **to_copy** was a **positive** number (unsigned value) and the **min()** comparison compared two positive integers (unsigned values).

```
571       for (auto i = m_receive_buffer.begin()
572                 , end(m_receive_buffer.end()); i != end;)
573         {
574                 if (target == m_read_buffer.end())
575                 {
576                         UTP_LOGV("  No more target buffers: %d bytes left in buffer\n"
577                                 , m_receive_buffer_size);
578                         TORRENT_ASSERT(m_read_buffer.empty());
579                         break;
580                 }
581
```

```
582 #if TORRENT_USE_INVARIANT_CHECKS
583                 check_receive_buffers();
584 #endif
585
586                 packet* const p = i->get();
587                 int const to_copy = std::min(p->size - p->header_size, aux::numeric_cast<int>(target-
>len));
588                 TORRENT_ASSERT(to_copy >= 0);
589                 std::memcpy(target->buf, p->buf + p->header_size, std::size_t(to_copy));
590                 ret += std::size_t(to_copy);
591                 target->buf = static_cast<char*>(target->buf) + to_copy;
592                 TORRENT_ASSERT(target->len >= std::size_t(to_copy));
593                 target->len -= std::size_t(to_copy);
594                 m_receive_buffer_size -= to_copy;
595                 TORRENT_ASSERT(m_read_buffer_size >= to_copy);
596                 m_read_buffer_size -= to_copy;
597                 p->header_size += std::uint16_t(to_copy);
598                 if (target->len == 0) target = m_read_buffer.erase(target);
```

Note that a result of an incorrect length of **to_copy** or **header_size** could result in a buffer overflow on line 589.

Recommended Remediation:

The assessment team recommends utilizing unsigned fixed width integer variables when processing integer input. This can help prevent against type confusion issues or integer overflow or underflows.

References:

Modern Memory Safety in C and CPP
Boost numeric_cast

APPENDICES

A1: Statement of Coverage

The security assessment team focused on assessing the following areas of the libtorrent library:

- BitTorrent Peer connections
- DHT messages
- uTP packets
- UDP tracker messages
- HTTP tracker messages
- .torrent files
- malicious file paths
- HTTP peer connections
- HTTP parser

The assessment team primarily utilized Libtorrent RC_2_0 for security testing, fuzzing and source code review. In a couple situations libtorrent RC_1_2 and libtorrent rasterbar **1.2.10** release were utilized for testing the security of code segments using different build configuration options.

The techniques that were leveraged were the development and utilization of fuzzing harnesses (libfuzzer), manual code analysis, automated static code analysis, automated fuzzing harness/stub generation, protocol analysis and dynamic analysis. While there was progress that was made utilizing the technique of automated fuzzing harness generation, the assessment team was not able to complete the automated fuzzing harness generation in the allotted time but has provided details on the progress for other researchers or the libtorrent team to continue (see Automated Fuzzing Generation Appendix). As such, the assessment team invites Mozilla and other COTS vendors to sponsor additional work in this regard with the IncludeSec team to finish the FuzzGen porting and framework implementation for libtorrent and other open source projects.

A2: Appendix – Known BitTorrent Protocol Vulnerabilities and Improvements

There are several known security concerns with the BitTorrent protocol (or set of protocols) and most (if not all) listed here are known by the **libtorrent** development team. The assessment team is listing these for documentation purposes and to add discussion to the potential remediation as it impacts the **libtorrent** library.

BitTorrent Protocol Security Concerns:

1. BitTorrent traffic can be transmitted unencrypted
2. RC4 encryption is utilized
3. SHA1 hashing is leveraged for data integrity

4. PE encryption is susceptible to active man-in-the-middle attacks
5. BitTorrent protocol is susceptible to traffic analysis and can be detected
6. Authentication is not required to join the network under normal protocol operation
7. BitTorrent protocol can be utilized to create distributed denial of service attacks

See the references section for more information on BitTorrent protocol security weaknesses.

A3: Appendix – Libtorrent Ubuntu Build Automation Script

The following is a script that can be leveraged to install dependencies for building libtorrent on Ubuntu. The script has been tested on Ubutnu 18.04. It's possible that these commands could be added to the build documentation to help libtorrent developers speed up the process of building libtorrent.

```
#!/bin/bash
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install git clang libssl-dev cmake
git clone --recurse-submodules https://github.com/arvidn/libtorrent.git
wget https://dl.bintray.com/boostorg/release/1.74.0/source/boost_1_74_0.tar.gz
tar xzf boost_1_74_0.tar.gz
cd boost_1_74_0/
./bootstrap.sh
sudo ln -s $PWD/b2 /usr/local/bin/b2
echo 'using clang : 6 : clang++-6.0 ;' >> ~/user-config.jam
echo "export BOOST_ROOT=$PWD/" >> ~/.bashrc
echo "export BOOST_BUILD_PATH=$PWD/tools/build/" >> ~/.bashrc
export BOOST_ROOT=$PWD/
export BOOST_BUILD_PATH=$PWD/tools/build/
cd ../libtorrent/
b2 cxxstd=14 -j$(nproc)
```

Perhaps in the future a Docker file could be created that could further help developers speed the build setup processes up.

A4: Appendix – Libtorrent Ubuntu Fuzzer Automation Script

The following is a script that can be leveraged to install dependencies, build and fuzz the libtorrent library on Ubuntu. The script has been tested on Ubutnu 18.04.

```
#!/bin/bash
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install git clang libssl-dev cmake
git clone --recurse-submodules https://github.com/arvidn/libtorrent.git
wget https://dl.bintray.com/boostorg/release/1.74.0/source/boost_1_74_0.tar.gz
tar xzf boost_1_74_0.tar.gz
cd boost_1_74_0/
./bootstrap.sh
```

```
sudo ln -s $PWD/b2 /usr/local/bin/b2
echo 'using clang : 6 : clang++-6.0 ;' >> ~/user-config.jam
echo "export BOOST_ROOT=$PWD/" >> ~/.bashrc
echo "export BOOST_BUILD_PATH=$PWD/tools/build/" >> ~/.bashrc
export BOOST_ROOT=$PWD/
export BOOST_BUILD_PATH=$PWD/tools/build/
cd ../libtorrent/fuzzers/
wget https://github.com/arvidn/libtorrent/releases/download/2.0/corpus.zip
unzip corpus.zip
b2 cxxstd=14 -j$(nproc)
./run.sh
```

Perhaps in the future a Docker file could be created that could further help developers speed up the build and fuzzing processes.

A5: Appendix – Integer Overflow bdecode Test Script

The following is a code snippet demonstrating potential integer overflow conditions that potentially could occur in the bdecode.cpp **parse_int()** function and displaying other numeric_limits:

```
#include <limits>
#include <iostream>
#include <inttypes.h>

using namespace std;

// clang++-10 -fsanitize=undefined test_int_overflow.cpp

int main(int argc, char * argv[])
{

   std::cout << "int32_t: " << numeric_limits<int32_t>::max() << std::endl;
   std::cout << "uint32_t: " << numeric_limits<uint32_t>::max() << std::endl;
   std::cout << "int64_t: " <<numeric_limits<int64_t>::max() << std::endl;
   std::cout << "uint64_t: " <<numeric_limits<uint64_t>::max() << std::endl;
   std::cout << "long long: " <<numeric_limits<long long>::max() << std::endl;
   std::cout << "unsigned long long: " <<numeric_limits<unsigned long long>::max() << std::endl;

   std::cout << "uint64_t max divided by 10: " <<numeric_limits<uint64_t>::max()/10 << std::endl;
   std::cout << "int64 max divided by 10: " <<numeric_limits<int64_t>::max()/10 << std::endl;

   //test values for testing integer overflow conditions
   //int64_t val = -922337203685477581;
   int64_t val = -9223372036854775806;
   //int64_t val = -5764607523034234880;

   std::cout << "val is: " << val << std::endl;

   //this check simulates the integer overflow detection check in bdecode.cpp of the libtorrent library
   if (val > std::numeric_limits<std::int64_t>::max() / 10)
   {
           std::cout << "Overflow Detected" << std::endl;
   }
   else {
           std::cout << "No Overflow" << std::endl;
```

```
   }
   val = val*10;

   std::cout << "val multiplied by 10: " << val << std::endl;
   return 0;

}
```

The following shows the steps to compile and utilize the code:

```
clang++-10 -fsanitize=undefined test.cpp
./a.out
int32_t: 2147483647
uint32_t: 4294967295
int64_t: 9223372036854775807
uint64_t: 18446744073709551615
long long: 9223372036854775807
unsigned long long: 18446744073709551615
uint64_t max divided by 10: 1844674407370955161
int64 max divided by 10: 922337203685477580
val is: -9223372036854775806
No Overflow
test.cpp:37:13: runtime error: signed integer overflow: -9223372036854775806 * 10 cannot be represented in
type 'long'
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior test.cpp:37:13 in
val multiplied by 10: 20
```

A6: Appendix – Libtorrent MT19937 PRNG Utilization

The following code can be used to emulate behavior similar to the random.cpp code in the libtorrent library:

```
#include <random>
#include <iostream>

//g++ poc_generate_mt19937.cpp
//Generate 1000 pseudo random numbers utilizing standard mt19937 library

int main()
{
    std::random_device rd;  //Will be used to obtain a seed for the random number engine
    std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
    std::uniform_int_distribution<std::uint32_t> distrib(0, 4294967295);

    for (int n=0; n<1000; ++n)
        //Use distrib to transform to create uniform distribution and enforce min/max
        std::cout << distrib(gen) << std::endl;
}
```