## Description

Hello,

Coverity Scan warns ext/socket/init.c and raddrinfo.c.

rsock_s_recvfrom in ext/socket/init.c does:

```c
arg.alen = (socklen_t)sizeof(arg.buf);
```

then calls rsock_io_socket_addrinfo:

```c
return rb_assoc_new(str, rsock_io_socket_addrinfo(sock, &arg.buf.addr, arg.alen));
```

rsock_io_socket_addrinfo indirectly calls init_addrinfo in ext/socket/raddrinfo.c.

```c
(rsock_io_socket_addrinfo -> rsock_fd_socket_addrinfo -> rsock_addrinfo_new -> init_addrinfo)
```

init_addrinfo does:

```c
memcpy((void *)&rai->addr, (void *)sa, len);
```

Note that sa is &arg.buf.addr, and len is arg.alen. &arg.buf.addr is a pointer to sockaddr, and arg.len is sizeof(union_sockaddr), not sizeof(sockaddr), which is indeed inconsistent.

I don't think this inconsistency will cause actual harm, but it would be better to fix.

--

Yusuke Endoh mame@ruby-lang.org

## History

### #1 - 06/18/2015 07:49 AM - akr (Akira Tanaka)

- Status changed from Open to Feedback

I'm not sure the problem.

arg.alen is initialized as sizeof(union_sockaddr) and modified by recvfrom() which is less than or equal to sizeof(union_sockaddr).

rai is rb_addrinfo_t and rai->addr is union_sockaddr.

So the memcpy() doesn't overflow.

I don't think this inconsistency will cause actual harm, but it would be better to fix.

Do you have an idea to fix it?

I guess the inconsistency is caused by "struct sockaddr" is used as a type for generic socket addresses but actually a fixed length buffer which may be not enough for some addresses. It is a Unix tradition and too difficult to fix.

### #2 - 06/18/2015 03:59 PM - mame (Yusuke Endoh)

- Status changed from Feedback to Open
Akira Tanaka wrote:

arg.alen is initialized as sizeof(union_sockaddr) and modified by recvfrom() which is less than or equal to sizeof(union_sockaddr).

rai is rb_addrinfo_t and rai->addr is union_sockaddr.

So the memcpy() doesn't overflow.

I think that Coverity Scan is warning against sa, not rai. sa is &arg.buf.addr, not &arg.buf. If it were &arg.buf, there is certainly no problem.

Honestly I'm not sure the C language specification: is it guaranteed that a pointer to a field of a union and a pointer to the union itself? In short, (void*)&arg.buf.addr == (void*)&arg.buf? If it is guaranteed, there is no problem. But I couldn't find the guarantee from the specification.

I don't think this inconsistency will cause actual harm, but it would be better to fix.

Do you have an idea to fix it?

I guess the inconsistency is caused by "struct sockaddr" is used as a type for generic socket addresses but actually a fixed length buffer which may be not enough for some addresses.

It is a Unix tradition and too difficult to fix.

I'm not familiar with socket apis. Do you mean that the apis are ill-designed so that we cannot use them in the strict C language? If so, I agree that it is difficult to fix.

--
Yusuke Endoh mame@ruby-lang.org

#3 - 06/19/2015 01:36 AM - akr (Akira Tanaka)

Yusuke Endoh wrote:

Honestly I'm not sure the C language specification: is it guaranteed that a pointer to a field of a union and a pointer to the union itself? In short, (void*)&arg.buf.addr == (void*)&arg.buf? If it is guaranteed, there is no problem. But I couldn't find the guarantee from the specification.

Yes.

There is a description in the committee draft of C99.

[13] The size of a union is sufficient to contain the largest of its members. The value of at most one of the members can be stored in a union object at any time. A pointer to a union object, suitably converted, points to each of its members (or if a member is a bit-field, then to the unit in which it resides), and vice versa.

This section is quoted to Wikipedia (from C90).
https://en.wikipedia.org/wiki/Union_type

JIS X 3010:2003 section 6.7.2.1 has same description in Japanese.

I'm not familiar with socket apis. Do you mean that the apis are ill-designed so that we cannot use them in the strict C language? If so, I agree that it is difficult to fix.

I don't say "cannot" here.

#4 - 06/19/2015 02:11 PM - mame (Yusuke Endoh)

- Status changed from Open to Rejected

I talked with akr on twitter, and was convinced that (void*)&arg.buf.addr == (void*)&arg.buf was guaranteed. So closing.

6.3.2.3 (7) says that a cast to char * yields a pointer to the lowest addressed byte of the object. This indirectly guarantees the equality, I think.

A pointer to an object or incomplete type may be converted to a pointer to a different object or incomplete type. If the resulting pointer is not correctly aligned for the pointed-to type, the behavior is undefined. Otherwise, when converted back again, the result shall compare equal to the original pointer. When a pointer to an object is converted to a pointer to a character type, the result points to the lowest addressed byte of the object. Successive increments of the result, up to the size of the object, yield pointers
to the remaining bytes of the object.

Thank you very much!

--
Yusuke Endoh mame@ruby-lang.org