

Socket Security: Using SO_REUSEADDR and SO_EXCLUSIVEADDRUSE

Developing secure applications is a priority for most developers today; however, socket security is often overlooked but is especially critical. Socket security deals with other processes binding to the same port that an existing application already has a socket bound to. In the past, any process could steal the port that another socket was using which results in a denial of service type attack as the hijacking port would start receiving traffic on the stolen interface(s) and port.

In general, socket security applies mostly to server side processes. That is, applications that accept connections or receive datagram traffic. These applications typically bind to a well known port and are easy targets for malicious code. Client side applications are less prone to this type of attack as most clients bind to random (“ephemeral”) local ports. Client applications should bind to ephemeral ports (by specifying port 0 in the SOCKADDR structure passed to bind) unless there is a very compelling reason not to.

This article describes the default level of security for sockets on the various Windows platforms as well as how the socket options SO_REUSEADDR and SO_EXCLUSIVEADDRUSE affect security. The different platforms support varying degrees of security as show in Table 1. The Windows 9x code base (including Windows Millennium) has the least amount of security while Windows Server 2003 is the most secure as the underlying security model for sockets was tightened. The following sections describe in detail the socket security behavior using the SO_REUSEADDR and SO_EXCLUSIVEADDRUSE options as well as how the new enhanced socket security on Windows Server 2003 affects these options. The last section details how new applications should use these options to prevent denial of service attacks.

Platform	SO_REUSEADDR Available?	SO_EXCLUSIVEADDRUSE Available?	Enhanced Socket Security
<i>Windows 95</i>	Yes	No	No
<i>Windows 98</i>	Yes	No	No
<i>Windows ME</i>	Yes	No	No
<i>Windows NT 4.0</i>	Yes	Yes (SP4 and later)	No
<i>Windows 2000</i>	Yes	Yes	No
<i>Windows XP</i>	Yes	Yes	No
<i>Windows Server 2003</i>	Yes	Yes	Yes
<i>Windows Vista</i>	Yes	Yes	Yes

Table 1: Socket Security Support

Using SO_REUSEADDR

The SO_REUSEADDR option allows a socket to forcefully bind to a port that is already in use by another socket. The second socket simply has to call setsockopt with SO_REUSEADDR and a boolean TRUE value before calling bind. Once the second socket is bound then the behavior for all the sockets bound to the same port is undetermined. That is, in the case all sockets are TCP, it is non-deterministic which socket will receive incoming connection requests. A malicious program could forcefully bind sockets to ports already in use with the SO_REUSEADDR option to deny service. No special privileges are required to use this option. The exception to the non-deterministic behavior is multicast sockets. If two sockets are bound to the same interface and port and are members of the same multicast group, data will be delivered to both sockets.

Using SO_EXCLUSIVEADDRUSE

Before the SO_EXCLUSIVEADDRUSE option was introduced there wasn't much an application could do to prevent someone from stealing the port its sockets were bound to. The SO_EXCLUSIVEADDRUSE option was introduced to prevent this. The option is set with the setsockopt function before a socket is bound. After the option is set, the behavior of subsequent bind calls to the same port is different depending on the addresses used. Table 2 describes the various combinations. Note that the error code returned by the second bind can change depending on whether the second caller set any socket security options prior to bind.

First Caller		Second Caller					
		Default		SO_REUSEADDR		SO_EXCLUSIVEADDRUSE	
		Wildcard	Specific	Wildcard	Specific	Wildcard	Specific
Default	Wildcard	INUSE	INUSE	Success	Success	INUSE	INUSE
	Specific	INUSE	INUSE	Success	Success	INUSE	INUSE
SO_REUSEADDR	Wildcard	INUSE	INUSE	Success	Success	INUSE	INUSE
	Specific	INUSE	INUSE	Success	Success	INUSE	INUSE
SO_EXCLUSIVEADDRUSE	Wildcard	INUSE	INUSE	ACCESS	ACCESS	INUSE	INUSE
	Specific	INUSE	INUSE	ACCESS	ACCESS	INUSE	INUSE

Table 2: Bind Behavior with Various Options Set

The rows down describe the options and interface specified for the first caller while the columns across are for the second caller. The “wildcard” label indicates binding to the wildcard address for the given protocol (i.e. 0.0.0.0 for IPv4 and :: for IPv6). “Specific” refers binding to a specific address assigned to an interface. The table entries indicate whether the second caller succeeds or fails. “INUSE” indicates bind failing with WSAEADDRINUSE (10048) while ACCESS indicates failure with WSAEACCES (10013). “Success” indicates bind succeeds.

Note that when two sockets are bound to the same port number but each on a different explicit interface, there is no conflict. For example, take the case where a machine has two IP interfaces: 10.0.0.1 and 10.99.99.99. If the first bind is to 10.0.0.1 and port 5150 with the `SO_EXCLUSIVEADDRUSE` option set, then the second bind to 10.99.99.99 and port 5150 with any or no options set will succeed. However if on the other hand the first socket is bound to the wildcard address (0.0.0.0) and port 5150 (with `SO_EXCLUSIVEADDRUSE` set) then any subsequent bind to the same port regardless of IP address will fail either with `WSAEADDRINUSE` (10048) or `WSAEACCESS` (10013) depending on what options were set on the second bind socket.

In the case where the first bind sets no options or `SO_REUSEADDR` and the second bind performs a `SO_REUSEADDR` then the second socket has “hijacked” the port and the behavior is undetermined as to which socket will receive packets. This is why `SO_EXCLUSIVEADDRUSE` was introduced as anyone can call `SO_REUSEADDR` which means any existing socket bound with no options (or `SO_REUSEADDR`) can be hijacked by binding with the `SO_REUSEADDR` option.

While the `SO_EXCLUSIVEADDRUSE` option is extremely useful, there is an important caveat. If at least one connection which originated from or was accepted on a port bound with exclusive access is active then all binds to this port will fail. In this case, a “connection” is defined as a socket which is explicitly connected to a peer via *connect*, *WSAConnect*, or *ConnectEx* with the exclusive flag set or a connection returned from a listening socket (such as from *accept*, *WSAAccept*, or *AcceptEx*) which has the exclusive option set (on the listening socket). An active port for TCP is defined as in the ESTABLISHED, FIN_WAIT, FIN_WAIT_2, or LAST_ACK states (for a comprehensive explanation of the TCP states consult *TCP/IP Illustrated* by W. Richard Stevens). For ports without the exclusive flag set, the port may be reused as soon as the socket on which bind was called (i.e. the socket the connection was originated on or the listening socket) is closed.

On the other hand, a socket with the exclusive flag set cannot always be reused immediately after socket closure. For example, if a listening socket with the exclusive flag set accepts a connection after which the listening socket is closed, another socket cannot bind to the same port as the first listening socket with the exclusive flag until the accepted connection is no longer active.

This issue can become more complicated because even though the socket has been closed the underlying transport may not terminate its connection. Even after the socket is closed, the system must send all the buffered data, transmit a graceful disconnect to the peer, and wait for a graceful disconnect from the peer. It is possible for that the underlying transport may never release the connection such as when the peer advertises a 0 size window or other attacks. In the previous example, the listening socket was closed after a client connection was accepted. Now even if the client connection is closed, the port still may not be reused if the client connection remains in an active state because of unacknowledged data, etc.

To avoid this situation, applications should ensure a graceful shutdown themselves by calling *shutdown* with the `SD_SEND` flag and waiting in a *recv* loop until zero bytes are returned. Not only does this avoid the problem with port reuse, but it also guarantees all data has been

received by the peer as well as confirming to the peer that all of its data was successfully received.

The `SO_LINGER` option may be set on a socket to prevent the port going into one of the “active” wait states; however, this is discouraged as it can lead to undesired effects as it can cause the connection to be reset. For example, if data has been received but not yet acknowledged by the peer and the local machine closes the socket with `SO_LINGER` set then the connection will be reset, and the peer will discard the unacknowledged data. Also, picking a suitable time to linger is difficult as a value too small will result in many aborted connections while a large timeout can leave the system vulnerable to denial of service attacks (by establishing many connections and thereby stalling numerous application threads). Closing a socket with a non-zero linger value may also cause the `closesocket` call to block (see the documentation on `closesocket` for more information).

One final note about the `SO_EXCLUSIVEADDRUSE` option is that in the past only applications running with Administrator privileges could set this option. This restriction has been removed on Windows XP SP2 (and later) as well as Windows Server 2003 and later.

Enhanced Socket Security

New socket security was applied in Windows Server 2003. This was added because the default socket security easily allowed other processes to hijack ports from unsuspecting applications. The major security change is that by default sockets are not in a shareable state. That is, if an application wants other processes to be able to reuse the same port, it must explicitly enable it. To do so, the first caller of `bind` should set `SO_REUSEADDR` on the socket. The only exception is when the second bind is performed by the same user account as the first in which case the behavior is like it was before. This is for backward compatibility reasons. Table 3 shows the permutations of binding when both callers are the same user.

Also note that with the enhanced security some of the error conditions have changed. Specifically, some of the cases where `WSAEADDRINUSE` was returned previously will now fail with `WSAEACCES`.

First Caller		Second Caller					
		Default		SO_REUSEADDR		SO_EXCLUSIVEADDRUSE	
		<i>Wildcard</i>	<i>Specific</i>	<i>Wildcard</i>	<i>Specific</i>	<i>Wildcard</i>	<i>Specific</i>
Default	<i>Wildcard</i>	INUSE	Success	ACCESS	Success	INUSE	Success
	<i>Specific</i>	Success	INUSE	Success	ACCESS	INUSE	INUSE
SO_REUSEADDR	<i>Wildcard</i>	INUSE	Success	Success	Success	INUSE	Success
	<i>Specific</i>	Success	INUSE	Success	Success	INUSE	INUSE
SO_EXCLUSIVEADDRUSE	<i>Wildcard</i>	INUSE	ACCESS	ACCESS	ACCESS	INUSE	ACCESS
	<i>Specific</i>	Success	INUSE	Success	ACCESS	INUSE	INUSE

Table 3: Both callers are the same user

The rows down describe the options and interface specified for the first caller while the columns across are for the second caller. The “wildcard” label indicates binding to the wildcard address for the given protocol (i.e. 0.0.0.0 for IPv4 and :: for IPv6). “Specific” refers binding to a specific address assigned to an interface. The table entries indicate whether the second caller succeeds or fails. “INUSE” indicates bind failing with WSAEADDRINUSE (10048) while ACCESS indicates failure with WSAEACCES (10013). “Success” indicates bind succeeds.

There are a few somewhat confusing entries in the table worth explaining. For example, if the first caller sets SO_EXCLUSIVEADDRUSE on a specific address and a second caller binds to the wildcard address on the same port it will succeed. In this case the second caller is in reality bound to all interface *except* the specific interface the first caller is bound to. For example, if the first caller binds to 10.0.0.1 with the exclusive option set while the second caller binds to the wildcard address, the second caller will receive traffic destined to any local interface except 10.0.0.1. Note that the reverse does not hold true. That is, if the first caller sets the exclusive flag and binds to the wildcard address, then there is nothing the second caller can do to allow it to bind to the same port.

The other interesting change is this same scenario except the first and second callers are different users. Note that what user account the two callers are is irrelevant. That is, an administrator does not have any extra privileges over a normal user. Table 4 contains the bind permutations when the two sockets are created by different users.

First Caller		Second Caller					
		Default		SO_REUSEADDR		SO_EXCLUSIVEADDRUSE	
		Wildcard	Specific	Wildcard	Specific	Wildcard	Specific
Default	Wildcard	INUSE	ACCESS	ACCESS	ACCESS	INUSE	ACCESS
	Specific	Success	INUSE	Success	ACCESS	INUSE	INUSE
SO_REUSEADDR	Wildcard	INUSE	ACCESS	Success	Success	INUSE	ACCESS
	Specific	Success	INUSE	Success	Success	INUSE	INUSE
SO_EXCLUSIVEADDRUSE	Wildcard	INUSE	ACCESS	ACCESS	ACCESS	INUSE	ACCESS
	Specific	Success	INUSE	Success	ACCESS	INUSE	INUSE

Table 4: Callers are different users

The rows down describe the options and interface specified for the first caller while the columns across are for the second caller. The “wildcard” label indicates binding to the wildcard address for the given protocol (i.e. 0.0.0.0 for IPv4 and :: for IPv6). “Specific” refers binding to a specific address assigned to an interface. The table entries indicate whether the second caller succeeds or fails. “INUSE” indicates bind failing with WSAEADDRINUSE (10048) while ACCESS indicates failure with WSAEACCES (10013). “Success” indicates bind succeeds.

In the case where the callers are different users, note the default behavior. If the first caller does not set any options on the socket and binds to the wildcard address, then the second caller cannot set the reuse flag and steal the port and the default behavior with no options set is failure as well.

Application Strategies

When developing new applications it is important to consider the type of socket security necessary. Client applications (i.e. programs which connect or send data to a service) rarely require any extra steps since they typically bind to a random local port. If the client does require binding to a specific port in order to function correctly then adding socket security is a must.

The `SO_REUSEADDR` option has very few uses in normal applications aside from multicast sockets where data will be delivered to all sockets bound to the same port. Otherwise, applications that rely on setting this option are better off being redesigned to remove the dependency because security is in effect turned off once this option is enabled. Any malicious attacker is free to steal the port.

For the most part, all server applications should set `SO_EXCLUSIVEADDRUSE`. Not only does it prevent attackers from hijacking the port, but it will also indicate if someone is bound to an explicit interface on the same port. For example, the bind call for a server listening on the wildcard address and port X with the exclusive flag set will fail if there is another socket bound to an explicit interface and port X. If the server didn't set the exclusive flag, the bind would succeed but some other process would receive the traffic from the explicit interface. As mentioned earlier, setting the exclusive flag does have a side effect which should be taken into account in the initial design phase.

Lastly, even though Windows Server 2003 has increased socket security, an application should set the exclusive flag to ensure it is bound to all the interfaces requested (as explained in the previous paragraph). The security change in Windows . Server 2003 adds increased security for legacy applications that cannot be easily changed, but application developers should still design their products with all aspects of security in mind.

-- Anthony Jones