Network Working Group
Internet-Draft
Expires: November 30, 2004

M. Larsen
Ericsson
June 2004

Port Randomisation
draft-larsen-tsvwg-port-randomisation-00

Status of this Memo

Copyright Notice

Abstract

The Internet protocols TCP and UDP are both vulnerable to data injection attacks. The consequences of injected data range from nuisance through broken connections and corrupted local data.

This document describe a simple, efficient and client local method for random selection of the client port number, such that the possibility of an attacker guessing the exact value is reduced. This is not a replacement for cryptographic methods such as IPsec or the

TCP MD5 signature option. However, the proposed method provides improved security/obfuscation with very little effort and without any key management overhead.

The proposed algorithm has similarities with the algorithm proposed in [RFC1948].

Table of Contents

1. Introduction

The Internet protocols TCP [RFC793] and UDP [RFC768] are both vulnerable to data injection attacks. The consequences of injected data (which may be both control data and payload data) range from nuisance through broken connections and corrupted local data [TCPsecure][Watson].

To make such attacks possible, the attacker must usually know both local and peer IP addresses and ports (the connection four-tuple) and any sequence numbers involved in the communication. Alternatively the attacker must make a good prediction of the these parameters to reduce the search space. The connection must also exist long enough for the attack to be executed. Such attacks are feasible as illustrated by [Watson].

Besides IP addresses, Internet protocols like TCP and UDP use a set of ports (local and peer) to identify communication endpoints. Services are usually located at fixed, 'well-known' ports [IANA] at the host supplying the service (the server). Client applications connecting to any such service will contact the server by specifying the server IP address and service port number. The IP address and port number of the client are normally left unspecified by the client application and thus chosen automatically by the client networking stack. Ports chosen automatically by the networking stack are known as ephemeral ports [Stevens].

While the well-known service port and both server and client IP address may be available to an attacker, the ephemeral port of the client are usually unknown and must be guessed.

This document describe a method for random selection of the ephemeral port, thereby reducing the possibility of an off-path attacker guessing the exact value. This is not a replacement for cryptographic methods such as IPsec or the TCP MD5 signature option [RFC2385]. However, the proposed method provides improved obfuscation with very little effort and without any key management overhead.

The mechanism is a local modification and may be incrementally deployed. The mechanism is fully compliant with both [RFC793] and [RFC768].

Since the mechanism is an obfuscation technique, focus has been on a reasonable compromise between level of obfuscation and ease of implementation. Thus the algorithm must be computationally efficient, and not require substantial data structures.

2. Randomising Ports

2.1 Ephemeral Port Range

The Internet Assigned Numbers Authority (IANA) assigns the unique parameters and values used in protocols developed by the Internet Engineering Task Force (IETF), including well-known ports [IANA]. IANA has traditionally reserved the following use of the 16-bit port range of TCP and UDP:

- The Well Known Ports, 0 through 1023.
- The Registered Ports, 1024 through 49151
- The Dynamic and/or Private Ports, 49152 through 65535

The range for assigned ports managed by the IANA is 0-1023, the remainder is registered by IANA but not assigned.

The ephemeral port range traditionally includes the 49152-65535 range, and should also include the 1024-49151 range. However, since this range include user specific server ports this may not always be possible. A host should use the largest possible range, since this improves the obfuscation provided by randomising the ephemeral ports.

Note that this method may also be used when dynamically reassigning ports as proposed by [Shepard].

2.2 Choosing the Port

Choosing a random port can, if a suitable random source is available, be implemented as a simple random selection, i.e.:

```
port = min_ephemeral + random() % (max_ephemeral - min_ephemeral)
```

Figure 1

Several well-know operating systems use this approach.

However, since the resulting connection four-tuple must be unique, the chosen port may already be in use with identical IP addresses and server port, thus the four-tuple is not unique. Consequently multiple ports may have to be tried and verified against all existing connections before a port can be chosen.

Although carefully chosen random sources and four-tuple lookup mechanisms optimised through e.g. hashing, will mitigate the cost of this verification, some systems may still not like to incur this unknown search time.

Systems that are specially vulnerable to this kind of repeated four-tuple collisions are systems that create many connections from a single local IP address to a single service (i.e. both IP addresses and peer port are fixed). Gateways such as proxy servers are an example of such a system.

Finding ports that result in a unique four-tuple are handled by some operating systems by having a global 'next ephemeral port' variable that is equal to the previously chosen ephemeral port + 1, i.e. the selection process is:

```
next_ephemeral_port = 1024;  /*initialisation, could be random*/

do {
    port = next_ephemeral_port;
    if (next_ephemeral_port == max_ephemeral_port) {
        next_ephemeral_port = min_ephemeral_port;
    } else {
        next_ephemeral_port++;
    }
} until (four-tuple is unique);
```

Figure 2

We will refer to this as 'Algorithm 1'. Note that the loop prevention mechanism has been left out for clarity.

This works well, since the number of connections (globally, across all four-tuples) that has a life-time longer than it takes to exhaust the total ephemeral port range is small, thus four-tuple collisions are rare.

However, this method has the drawback, that the 'next_ephemeral_port' variable and thus the ephemeral port range is shared between all connections and it is easy to predict the next ports chosen by the client. If an attacker operates an innocent server to which the client connects, it is easy to obtain a reference point for the current value of 'next_ephemeral_port.

Ideally, we would like a 'next_ephemeral_port' value for each set of (local/peer IP addresses, peer port). These should be initialised with random values within the ephemeral port range and would thus separate the ephemeral port ranges of the connections entirely. Since we do not want to store all these 'next_ephemeral_port' values, we propose an offset function F(), that can be computed from the local/peer IP addresses, peer port and a secret key. F() will yield (practically) different values for each set of arguments, i.e.:

```
next_ephemeral_port = 1024;  /*initialisation, could be random*/

offset = F(local_IP, remote_IP, remote_port, secret_key);
do {
    port = min_ephemeral +
           (next_ephemeral_port + offset)
               % (max_ephemeral - min_ephemeral);
    next_ephemeral_port++;
} until (four-tuple is unique);
```

Figure 3

We will refer to this as 'Algorithm 2'. Note that the loop prevention mechanism has been left out for clarity.

In other words, the function F() provides a connection-local fixed offset of the global ephemeral port range controlled by 'next_ephemeral_port'. Both the 'offset' and 'next_ephemeral_port' variables may take any value within the storage type range since we are restricting the resulting port similar to that shown in Figure 1. This allows us to simply increment the 'next_ephemeral_port' variable and rely on the unsigned integer to simply wrap-around.

The function F() should be a cryptographic hash function like MD5 [RFC1321]. The function should use both IP addresses, the peer port and a secret key value to compute the offset. The peer IP address is the primary separator and must be included in the offset calculation. The local IP address and peer port may in some cases be constant and not improve the connection separation, however, they should also be included in the offset calculation.

Cryptographic algorithms stronger than e.g. MD5 should not be necessary, given that port randomisation is a pure obfuscation technique. The secret should be chosen as random as possible, see [RFC1750] for recommendations on choosing secrets.

Note that on multiuser systems, the function F() could include user specific information, thereby providing protection not only on a host to host basis, but on user to service basis.

2.3 Secret Key

Every complex manipulation (like MD5) is no more secure than the input values, and in the case of ephemeral ports, the secret key. If an attacker is aware of which cryptographic hash function is being used by the victim (which we should expect), and the attacker can obtain enough material (e.g. ephemeral ports chosen by the victim), the attacker may simply search the entire secret key space to find

matches.

To protect against this, the secret key should be of a reasonable length. Key-lengths of 32-bits or 64-bits should be adequate, since a 32-bit secret would result in approximately 65k possible secrets if the attacker is able to obtain a single ephemeral port (assuming a good hash function). If the attacker is able to obtain more ephemeral ports 64-bits or more should be used.

Another possible mechanism of protecting the secret key is to change it after some time. If the host platform is capable of producing reasonable good random data, the secret key can be changed.

Changing the secret will cause abrupt shifts in the chosen ephemeral ports, and consequently collisions may occur. Thus the change in secret key should be done with consideration and could be performed whenever one of the following events occur:

- Some predefined/random time has expired.
- The secret has been used N times (i.e. we consider it insecure).
- There are few active connections (possibility of collision is low).
- There is little traffic (the performance overhead of collisions is tolerated).
- There is enough random data available to change the secret key (pseudo-random changes should not be done).

2.4 Choosing Algorithm

Algorithm 1 has the advantage, that it provides complete randomisation, but may not scale well with many simultaneous connections. Algorithm 2 provides complete separation in local/peer IP address and peer port space, and only limited separation in other dimensions (See Section Section 2.3), however, this algorithm scales well.

Thus Algorithm 1 should be used when the cost of choosing an ephemeral port is not important, or when the ratio of used ports and available ports are low (for given local/peer IP addresses and peer port). A switch to algorithm 2 should happen if the cost of choosing an ephemeral port is important and when the ratio between used ports and available ports increase.

Note that when the ratio between used ports and available ports increase, the obfuscation resulting from port randomisation decrease and has no effect when the entire port space is in use.

The ratio where to switch between algorithms depend on the cost of

the four-tuple uniqueness test. Systems capable of handling many simultaneous connections normally has an efficient PCB-lookup. However, verifying a four-tuple for uniqueness requires a lookup against all existing connections, even unconnected (but bound). Additionally, options exist, that will allow reuse of ports, making the detection even more complex than a PCB-lookup. The the cost of a four-tuple verification may easily be many times that of a single PCB lookup.

While the ratio is very implementation dependent and calculating the exact ratio may be difficult without using additional resources, an appropriate ratio can be estimated and used for an algorithm switch. E.g. if the ephemeral port range contain N possible ports, the switch to algorithm 2 may happen when the total number of connections reach N/2.

3. Security Considerations

Randomising ports is no replacement for cryptographic mechanisms, such as IPsec.

An eavesdropper, which can monitor the ephemeral ports of other hosts (and thus also sequence numbers etc.) can easily hijack or corrupt the connection. Randomising ports does not provide any additional protection against this kind of attacks. In such situations stronger authentication techniques should be used.

If the local offset function F() results in identical offsets for different inputs, the port-offset mechanism proposed in this document has no or reduced effect.

If random numbers are used as the only source of the secret key, they must be chosen in accordance with the recommendations given in [RFC1750].

If all ports available in the ephemeral port range are in use, randomisation provides no obfuscation.

If an attacker use dynamically assigned IP addresses, the current ephemeral port offset (Algorithm 2) for a given four-tuple can be sampled and subsequently be used to attack an innocent peer reusing this address. However, this is only possible until a re-keying happens as described above. Also, since ephemeral ports are only used on the client side (e.g. the one initiating the connection), both the attacker and the new peer needs to be servers in the above scenario. Although servers using dynamic IP addresses exist, they are not very common and with an appropriate re-keying mechanism the effect of this attack is limited.

4. Acknowledgements

The offset function was inspired by the mechanism proposed for defending against TCP sequence number attacks [RFC1948].

5. References

5.1 Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC1750] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[RFC1948] Bellovin, S., "Defending Against Sequence Number Attacks", RFC 1948, May 1996.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

5.2 Informative References

[TCPsecure] Dalal, M., "Transmission Control Protocol security considerations", draft-ietf-tcpm-tcpsecure-01.txt (work in progress), June 2004.

[Watson] Watson, P., "Slipping in the Window: TCP Reset attacks", december 2003.

[IANA] "IANA Port Numbers", <http://www.iana.org/assignments/port-numbers>.

[Stevens] Stevens, W., "Unix Network Programming, Volume 1: Networking APIs: Socket and XTI, Prentice Hall", 1998.

[Shepard] Shepard, T., "Reassign Port Number option for TCP", draft-shepard-tcp-reassign-port-number-00 (work in progress), July 2004.

Author's Address

Michael Vittrup Larsen
Ericsson
Skanderborgvej 232
Aarhus DK-8260
Denmark

Phone: +45 8938 5100
EMail: michael.vittrup.larsen@ericsson.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.