



Snort Back Orifice Preprocessor Vulnerability Details

Chris Ries
Security Research Analyst

VigilantMinds Inc.
4736 Penn Avenue
Pittsburgh, PA 15224

info@vigilantminds.com

Background

On October 18th, 2005, Sourcefire announced a stack-based buffer overflow in Snort's Back Orifice Preprocessor [1]. The vulnerability can be exploited by a remote attacker using a single UDP packet and can lead to remote code execution. The Back Orifice preprocessor must be enabled for a remote attacker to exploit this vulnerability.

The vulnerable preprocessor is responsible for identifying traffic associated with Back Orifice (BO), a Windows-based remote administration program that was released in 1998. BO can be used by attackers as a backdoor to control compromised hosts.

The vulnerability affects Snort versions 2.4.0-2.4.2 and has been fixed in the latest version, 2.4.3. The latest BO preprocessor version also has the ability to detect exploitation attempts against this vulnerability.

The overflow was originally discovered by Internet Security Systems (ISS) X-Force [2].

The Back Orifice Protocol

The Back Orifice client and server communicate via encrypted UDP packets. The unencrypted BO packet header has the following structure.

Field	Size in bytes
MAGIC	8
LEN	4
ID	4
TYPE	1
DATA	Variable
CRC	1

- The **MAGIC** field always contains the value "!*QWTY?".
- The **Length** field contains the length (in bytes) of the entire BO packet, including the header.
- The **ID** field contains an identification number that is used to correlate transmitted and received traffic between clients and hosts.
- The **Type** field contains the command of the packet. The vulnerability occurs in code that processes a PING (0x01) packet. Other type values include commands to reboot or shutdown the machine running the BO server, or to manipulate the machine's processes and file system. Please see [reference 3](#) at the conclusion of this paper for a complete listing of commands.
- The **Data** field contains a variable-length sequence of bytes and is interpreted differently depending on the type of packet.
- The **CRC** field appears to be silently ignored by the BO server.

The BO packet is encrypted using an extremely basic algorithm prior to transmission. The algorithm essentially works by applying an XOR (exclusive or) operation between the packet's bytes and numbers returned by a simple random number generator. In order to identify BO traffic, Snort's BO preprocessor breaks this encryption algorithm and looks for the MAGIC value in the first eight bytes of the packet.

It is possible to break the encryption algorithm for two reasons:

1. The plaintext of the first eight bytes of the packet is known (MAGIC value).
2. The range of possible values for the seed of the random number generator is small.

Please see [reference 3](#) at the conclusion of this paper for more information on the BO encryption algorithm.

Once a packet is identified as BO traffic, the preprocessor attempts to identify whether the traffic is sourcing from a BO client or server, and then it generates an event.

Vulnerability Details

The vulnerability occurs when processing a BO packet with an overly long value in the length field. The packet cannot have a source or destination UDP port of 31337 (the default server port number) in order for the vulnerability to be triggered.

The stack overflow exists in the BoGetDirection() function of the preprocessor, which is used to determine whether the UDP BO packet is being sent from a BO client or server.

The function first attempts to determine the direction by seeing if either the source or destination port is equal to the default server port, 31337. If the default server port is not the source or destination port, the function decrypts and parses the BO header. In the case of a PING packet, the data from the packet is also copied to statically sized stack buffers.

The following is the relevant code from the BoGetDirection() function. The while loop marked in **bold** performs the copying.

```
static int BoGetDirection(Packet *p, char *pkt_data)
{
    u_int32_t len = 0;
    u_int32_t id = 0;
    u_int32_t l, i;
    char type;
    char buf1[1024];
    char buf2[1024];
    char *buf_ptr;
    char plaintext;

    ... irrelevant code removed ...

    /* Get length from BO header - 32 bit int */
    for ( i = 0; i < 4; i++ )
    {
        plaintext = (char) (*pkt_data ^ (BoRand()%256));
        l = (u_int32_t) plaintext;
        len += l << (8*i);
        pkt_data++;
    }

    ... irrelevant code removed ...
}
```

```

/* Adjust for BO packet header length */
len -= 18;

if ( type == BO_TYPE_PING )
{
  i = 0;
  buf_ptr = buf1;
  *buf1 = 0;
  *buf2 = 0;
  /* Decrypt data */
  while ( i < len )
  {
    plaintext = (char) (*pkt_data ^ (BoRand()%256));
    *buf_ptr = plaintext;
    i++;
    pkt_data++;
    buf_ptr++;
    if ( plaintext == 0 )
      buf_ptr = buf2;
  }
  /* null-terminate string */
  *buf_ptr = 0;

  ... irrelevant code removed ...
}

```

The while loop that copies the data from the packet uses the length field (stored in the 'len' variable) of the header to determine the number of bytes to copy. No verification is made to ensure that the data can fit into the stack buffers buf1[1024] and buf2[1024]. With an overly long value for the length field, stack memory beyond these buffers can be overwritten with bytes from the packet's data. This leads to the stack overflow.

Exploitation of the Vulnerability

This vulnerability is not quite as simple as a classic buffer overflow, which is caused by a call to an insecure function such as strcpy or sprintf. However, it is still relatively simple and can be exploited with basic stack smashing techniques. VigilantMinds developed reliable remote code execution exploits for the vulnerability for a number of different target platforms.

The following BO header will trigger the vulnerability if it is encrypted using the BO encryption algorithm and sent using source and destination ports other than 31337.

Field	Size in bytes	Value
MAGIC	8	"!*QWTY?"
LEN	4	Large value (ex. 5000)
ID	4	Random ID value
TYPE	1	0x01 (BO PING)
DATA	Variable	Long string (ex. 5000 bytes)
CRC	1	Any value

The return address of the BoGetDirection() stack frame can be overwritten in order to execute attacker-supplied code if a specially crafted string is supplied for the data value. The large size of the overflowed buffers facilitates exploitation, because there is ample room for any shellcode and a long NOP sled. The vulnerability can be triggered by any traffic monitored by the vulnerable Snort process, so the exploit packet does not need to be targeted at the host running Snort. Furthermore, reverse connect shellcode usage allows for blind exploitation of the vulnerability. An attacker can send out exploit packets to any address, and wait for compromised hosts to connect back.

Countermeasures

Sourcefire listed two preventative measures in their announcement of the vulnerability. The first was a workaround that involved disabling the preprocessor. They provided the following instructions to do so [1].

1. Locate the line "preprocessor bo"
2. Comment out this line by preceding it with a hash (#). The new line will look like "#preprocessor bo"
3. Save the file
4. Restart snort

This workaround mitigates the risk of the vulnerability, but also prevents Snort from identifying Back Orifice UDP traffic.

The second solution was to upgrade to Snort v2.4.3. The updated preprocessor in this version of Snort not only fixes the vulnerability, but also includes the ability to detect attempts to exploit the vulnerability. Reliable detection cannot be accomplished by a Snort signature since BO packets are encrypted. However, since the preprocessor decrypts the BO packet it has the ability to inspect the decrypted header's length field for overly long values.

An additional measure that minimizes the risk of this vulnerability is to run Snort on a hardened operating system. For example, if Snort is being run on Linux, a solution such as the PaX project can make exploitation of vulnerabilities such as these much more difficult [4]. PaX and similar projects utilize a non-executable stack, address-space randomization, and other techniques as defensive measures against security bugs.

References

- [1] http://www.snort.org/rules/advisories/snort_update_20051018.html
- [2] <http://xforce.iss.net/xforce/alerts/id/207>
- [3] <http://www.magnux.org/~flaviovs/boproto.html>
- [4] <http://pax.grsecurity.net/>