opsec
Internet-Draft
Intended status: Standards Track
Expires: September 3, 2012

A. Yourtchenko
S. Asadullah
Cisco
M. Pisica
BT

March 2, 2012

Human-safe IPv6: Cryptographic transformation of hostnames as a base for secure and manageable addressing

draft-yourtchenko-opsec-humansafe-ipv6-00

Abstract

Although the IPv6 address space within a single /64 subnet is very large, the typical distribution of the addresses in this space is very non-uniform. This non-uniformity, together with the dictionary-based DNS brute-force enumeration, allows practical remote mapping of the IPv6 addresses in these subnets. This document proposes a technique which can be used to decrease the exposure of the server subnets to trivial scanning. As a side effect, the proposed technique allows to drastically simplify the address management.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 3, 2012.

Copyright Notice

Table of Contents

1. Introduction

The conventional wisdom says that a typical IPv6 subnet has the address space of 2^64 addresses, which makes it impossible to scan. This results in commonly held assertion that it is impossible to scan the IPv6 subnets, and the protocol is inherently more secure against scanning than IPv4. However, the currently deployed addressing techniques do not provide for a uniform distribution of the hosts within the entirety of the space - certain addresses are much more frequently used than the others. As a result, for the mostly-server subnets, more often than not one can realistically map the hosts that are present on that segment.

2. Caveats of version -00

(section to be removed in the -01)

This version of the document does assume the 64-bit Interface ID can have any values, whereas there are various restrictions that need to be taken into account (e.g., the U/L bit value). This is done deliberately as -00 is aimed at illustrating the principle and collecting the feedback from the community. Addressing these will be done as part of the future work on the document and the accompanying code.

3. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

4. Problem Statement

The problem is twofold: first, from the security point of view, one should try to avoid the easy to guess patterns that the traditional address assignment entails. At the same time, the naive approach of assigning purely random addresses to servers is not very scalable in real world for maintenance reasons

5. Proposed Solution

The idea is to exploit the randomness property of the encryption function output. The interface identifier, used within the IPv6 address of the host, would be derived from the 64-bit data

corresponding to hostname, encrypted with a site-wide "secret".

This satisfies the requirement of having the interface identifiers evenly distributed within the 2^64 space within the subnet; At the same time, such a formal mechanism of generating the host ID allows to reduce the maintenance overhead for the assignment and operation of the IPv6 addresses. Also it would allow, if needed, a DNS-less operation - after the network-wide secret is disseminated, the generation of the interface IDs can be distributed.

For flexibility, we define the forward and reverse transformation between the hostname and interface identifier as a two step process - the first step is to derive from the hostname the 64-bit "cleartext blob", which is being encrypted in the second step. Of course for the decryption the steps are reversed.

This document does not propose to replace/eliminate any of the existing address definition schemes, nor does it require the implementation in the devices - the addresses can be generated and assigned manually, and the enclosed algorithm can be used within the address management application.

6. Deriving the Cleartext Blob from Hostname

The method that is used to perform a 1:1 mapping of the hostname into the cleartext blob will determine the maximum length of the hostname. The most simple and obvious method used to illustrate the principle is an identity transform - therefore the hostname is itself the cleartext blob, and therefore the maximum length is 8 characters. Assuming the host name is using the characters from the range [0-9a-z-_], this would mean using 6 bits per character - therefore allowing to increase the maximum stored hostname length up to 10 characters. Potentially one can use other compression mechanisms - e.g. Huffman encoding or arithmetic encoding - however, one must leave a sufficient number of invalid values to detect the possible typos in the address.

7. Encrypting the Cleartext Blob

Any good enough encryption mechanism with the block size of 64 bit will suffice. For the demonstration purposes we choose DES - but possibly other encryption mechanisms can be used.

8. Security Considerations

Since the hostnames are not a secret data after one makes a connection to the server, one may argue that if an encryption algorithm is vulnerable to a known plaintext attack, this approach may make the mapping job easier.

Also, the fact that the encryption key distribution is rather wide, one may have concerns about the exposure of the hostnames from the addresses. However, we note that the scope of this proposal is merely to raise the barrier for the anonymous remote mapping, as well as to make the address management easier.

9. Acknowledgements

The authors are thankful to the following people for their review and valuable comments: Gunter Van de Velde, Warren Kumari, Ron Broersma, Jan Zorz, Ragnar Anfinsen, ...

10. IANA Considerations

This document has no IANA actions.

11. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Appendix A. A Sample Implementation

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <openssl/des.h>
#include <arpa/inet.h>
#include <ctype.h>


/*
 * A sample implementation of the human-safe
 * IPv6 addressing algorithm.
 * Requires the OpenSSL library, please compile
```

```
    * with "gcc human-safe.c -lssl"
    */


   /*
    * Encrypt and Decrypt routines are using DES as an example of
    * a symmetric encryption that has a 64-bit block size.
    */

   int encrypt(char *dst, char *src, char *key) {
     int n=0;
     DES_cblock k;
     DES_key_schedule sch;

     memset(k, 0, sizeof(k));
     memcpy(k, key, 8);
     DES_set_odd_parity(&k);
     if (DES_set_key_checked(&k, &sch) < 0) {
       printf("Error checking key\n");
     }
     DES_ecb_encrypt( (unsigned char (*)[8])src,
         (unsigned char (*)[8])dst, &sch, DES_ENCRYPT);
     return n;
   }


   int decrypt(char *dst, char *src, char *key) {
     int n=0;
     DES_cblock k;
     DES_key_schedule sch;

     memset(k, 0, sizeof(k));
     memcpy(k, key, 8);
     DES_set_odd_parity(&k);
     if (DES_set_key_checked(&k, &sch) < 0) {
       printf("Error checking key\n");
     }
     DES_ecb_encrypt( (unsigned char (*)[8])src,
         (unsigned char (*)[8])dst, &sch, DES_DECRYPT);

     return n;
   }

   /*
    * For the reference implementation, the mapping of the hostname
    * to cleartext blob is an identity transform
    */
```

```
   int hostname_enc(char *dst, char *src) {
     memcpy(dst, src, 8);
     return 1;
   }

   int hostname_dec(char *dst, char *src) {
     int i;
     for(i=0;i<8;i++) {
       if(!isalnum(src[i])) {
         return 0;
       }
     }
     memcpy(dst, src, 8);
     /* If it was the full-length string, null-terminate it */
     dst[8] = 0;
     return 1;
   }

   /* Main functions */

   host_to_addr(char *addr, char *host, char *prefix, char *secret) {
     int i;
     char blob[8];
     char xor_block[8];

     inet_pton(AF_INET6, prefix, addr);
     /* zero out the interface id part of /64 */
     for (i=8; i<16; i++) {
       addr[i] = 0;
     }
     hostname_enc(blob, host);
     encrypt(&addr[8], blob, secret);
     encrypt(xor_block, addr, secret);
     for(i=0; i<8; i++) {
       addr[i+8] ^= xor_block[i];
     }
     return i;
   }

   int addr_to_host(char *host, char *addr, char *secret) {
     char aptr[16];
     char blob[8];
     char xor_block[8];
     int i;
     memcpy(aptr, addr, 16);

     encrypt(xor_block, addr, secret);
     for(i=0;i<8;i++) {
```

```
      aptr[8+i] ^= xor_block[i];
    }
    decrypt(blob, &aptr[8], secret);
    if (!hostname_dec(host, blob)) {
      printf("Hostname decode failed, address error ?\n");
      return 0;
    }
    return 1;
  }



  void usage(char *name) {
    printf("Usage: \n");
    printf("         %s <secret> encode <prefix> <hostname>\n", name);
    printf("         %s <secret> decode <address>\n", name);
  }


  int main(int argc, char* argv[]) {
    char *secret;
    char *operation;
    char *prefix;
    char *hostname;
    char hostname_decoded[42];
    char addr_encoded[16];
    char buf[42];

    if (argc < 4) {
      usage(argv[0]);
      exit(1);
    }

    secret = argv[1];
    operation = argv[2];
    if (0 == strcmp(operation, "encode")) {
      prefix = argv[3];
      hostname = argv[4];
      host_to_addr(addr_encoded, hostname, prefix, secret);
      inet_ntop(AF_INET6, addr_encoded, buf, sizeof(buf));
      printf("%s\n", buf);
    } else if (0 == strcmp(operation, "decode")) {
      prefix = argv[3];
      inet_pton(AF_INET6, prefix, addr_encoded);
      addr_to_host(hostname_decoded, addr_encoded, secret);
      printf("%s\n", hostname_decoded);
    } else {
      usage(argv[0]);
```

```
      exit(1);
    }
    exit(0);
  }
```

Appendix B. Changes

Authors' Addresses

Andrew Yourtchenko
Cisco Systems, Inc.
De Kleetlaan, 7
Brussels, Diegem B-1831
Belgium

Email: ayourtch@cisco.com

Salman Asadullah
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134
USA

Email: sasad@cisco.com

Mircea Pisica
BT

Email: mircea.pisica@bt.com