Network Working Group   M. Bagnulo
Internet-Draft   UC3M
Expires: April 14, 2005   October 14, 2004

Hash Based Addresses (HBA)
draft-bagnulo-multi6dt-hba-00

Status of this Memo

Copyright Notice

Abstract

This memo describes a mechanism to provide a secure binding between the multiple addresses with different prefixes available to a host within a multihomed site. The main idea is that information about the multiple prefixes is included within the addresses themselves. This is achieved by generating the interface identifiers of the addresses of a host as hashes of the available prefixes and a random number. Then, the multiple addresses are generated by appending the different prefixes to the generated interface identifiers. The

result is a set of addresses, called Hash Based Addresses (HBAs), that are inherently bound.

Table of Contents

1. Introduction 3
2. CGA compatibility considerations 5
3. Multi-Prefix Extension for CGA 7
4. HBA-Set Generation 8
5. HBA verification 11
6. Security considerations 13
7. Contributors 15
8. Acknowledgments 16
9. References 17
9.1 Normative References 17
9.2 Informative References 17
Author's Address 17
Intellectual Property and Copyright Statements 18

1. Introduction

In order to preserve inter-domain routing system scalability, IPv6 sites obtain addresses from their Internet Service Providers. Such addressing strategy significantly reduces the amount of routes in the global routing tables, since each ISP only announces routes to its own address blocks, rather than announcing one route per customer site. However, this addressing scheme implies that multihomed sites will obtain multiples prefixes, one per ISP. Moreover, since each ISP only announces its own address block, a multihomed site will be reachable through a given ISP if the ISP prefix is contained in the destination address of the packets. This means that, if an established communication needs to be routed through different ISPs during its lifetime, addresses with different prefixes will have to be used. Changing the address used to carry packets of an established communication exposes the communication to numerous attacks, as described in [5], so security mechanisms are required to provide the required protection to the involved parties. This memo describes a tool that can be used to provide protection against some of the potential attacks, in particular against future/ premeditated attacks (a.k.a. time shifting attacks in [6]).

It should be noted that, as opposed to the mobility case where the addresses that will be used by the mobile node are not known a priori, the multiple addresses available to a host within the multihomed site are pre-defined and known in advance in most of the cases. The mechanism proposed in this memo takes advantage of this address set stability, and provides a secure binding between all the addresses of a node in a multihomed site. The mechanism does so without requiring the usage of public key cryptography, providing a cost efficient alternative to public key cryptography based schemes.

This memo describes a mechanism to provide a secure binding between the multiple addresses with different prefixes available to a host within a multihomed site. The main idea is that information about the multiple prefixes is included within the addresses themselves. This is achieved by generating the interface identifiers of the addresses of a host as hashes of the available prefixes and a random number. Then, the multiple addresses are generated by appending the different prefixes to the generated interface identifiers. The result is a set of addresses, called Hash Based Addresses (HBAs), that are inherently bound. A cost efficient mechanism is available to determine if two addresses belong to the same set, since given the prefix set and the additional parameters used to generate the HBA, a single hash operation is enough to verify if an HBA belongs to a given HBA set. No public key operations are involved in the verification process. In addition, it should also be noted that it is not required that all interface identifiers of the addresses of an

HBA set are equal, preserving some degree of privacy through changes in the addresses used during the communications.

2. CGA compatibility considerations

As described in previous section, the HBA technique uses the interface identifier part of the IPv6 address to encode information about the multiple prefixes available to a multihomed host. However, the interface identifier is also used to carry cryptographic information when Cryptographic Generated Addresses [1] are used. Therefore, conflicting usages of the interface identifier bits may result if this is not taken into account during the HBA design. There are at least two valid reasons to provide CGA-HBA compatibility:

First, the current Secure Neighbor Discovery specification [2] uses the CGAs defined in [1] to prove address ownership. If HBAs are not compatible with CGAs, then nodes using HBAs for multihoming wouldn't be able to do Secure Neighbor Discovery using the same addresses (at least the parts of SeND that require CGAs). This would imply that nodes would have to choose between security (from SeND) and reliability (from multi6). In addition to SeND, there are other protocols that are considering to benefit from the advantages offered by the CGA scheme, such as mobility support protocols [7]. Those protocols would also become incompatible with HBAs if HBAs are not compatible with CGAs.

Second, CGAs provide additional features that cannot be achieved using only HBAs. In particular, because of its own nature, the HBA technique only supports a predetermined prefix set that is known at the time of the generation of the HBA set. No additions of new prefixes to this original set are supported after the HBA set generation. In most of the cases relevant for site multihoming, this is not a problem because the prefix set available to a multihomed set is not very dynamic. New prefixes may be added in a multihomed site when a new ISP is available, but the timing of those events are rarely in the same time scale than the lifetime of established communications. It is then enough for many situations that the new prefix is not available for established communications and that only new communications benefit from it. However, in the case that such functionality is required, it is possible to use CGAs to provide it. This approach clearly requires that HBA and CGA approaches are compatible. If this is the case, it then would be possible to create HBA/CGA addresses that support CGA and HBA functionality simultaneously. The inputs to the HBA/CGA generation process will be both a prefix set and a public key. In this way, a node that has established a communication using one address of the CGA/HBA set can tell its peer to use the HBA verification when one of the addresses of its HBA/CGA set is used as locator in the communication or to use CGA (public/private key based) verification when a new address that does not belong to the HBA/CGA set is used as locator in the

communication.

So, because of the aforementioned reasons, it is a goal of the HBA design to define HBAs in a way that they are compatible with CGAs as defined in [1] and their usages described in [2]. This means that it must be possible to generate addresses that are both an HBA and a CGA i.e. that the interface identifier contains cryptographic information of CGA and the prefix-set information of an HBA. The CGA specification already considers the possibility of including additional information into the CGA generation process through the usage of Extension Fields in the CGA Parameter Data Structure. It is then possible to define a Multi-Prefix Extension for CGA so that the prefix set information is included in the interface identifier generation process.

Even though a CGA compatible approach is adopted, it should be noted that HBAs and CGAs are different concepts. In particular, the CGA is inherently bound to a public key, while a HBA is inherently bound to a prefix set. This means that a public key is not strictly required to generate an HBA. Requiring a public key to be included in the HBA generation process when the node does not have an need for one seems an overkill. It seems sensible then to allow the existence of three different types of addresses:

- CGA-only addresses: These are addresses generated as specified in [1] without including the Multi-Prefix Extension. They are bound to a public key and to a single prefix (contained in the basic CGA Parameter Data Structure). These addresses can be used for SeND [2] and if used for multihoming, their application will have to be based on the public key usage.

- CGA/HBA addresses: These addresses are CGAs that include the Multi-Prefix Extension in the CGA Parameters Data Structure used for their generation. These addresses are bound to a public key and a prefix set and they provide both CGA and HBA functionalities. They can be used for SeND as defined in [2] and for any usage defined for HBA (such as a multi6 protocol)

- HBA-only addresses: These addresses are bound to a prefix set but they are not bound to a public key. Because CGA compatibility, the CGA Parameter Data Structure will be used for their generation, but a random nonce will be included in the Public Key field instead of a public key. These addresses can be used for HBA based multihoming protocols, but they cannot be used for SeND,

3. Multi-Prefix Extension for CGA

The format of the Multi-Prefix Extension is the following:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Ext Type    |    Ext Len    |P|        Reserved             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                           Prefix[1]                           +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                           Prefix[2]                           +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   .                               .                               .
   .                               .                               .
   .                               .                               .
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                           Prefix[n]                           +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Ext Type: 8-bit identifier of the Multi-Prefix Extension (TBD IANA)

Ext Len: 8-bit unsigned integer. Length of the Extension in 8-octet units, not including the first 4 octets.

P flag: Set if a public key is included in the Public Key field of the CGA Parameter Data Structure. Reset if a additional Modifier bits are included in the CGA Parameter Data Structure.

Reserved: 15-bit reserved field. Initialized to zero.

Prefix[1...n]: Vector of 64-bit prefixes, numbered 1 to n.

4. HBA-Set Generation

The HBA generation process is based on the CGA generation process defined in section 4 of [1]. The goal is to require the minimum amount of changes to the CGA generation process.

The CGA generation process has three inputs: a 64-bit subnet prefix, a public key (encoded in DER as an ASN.1 structure of the type SubjectPublicKeyInfo), and the security parameter Sec.

The main difference between the CGA generation and the HBA generation is that while a CGA can be generated independently, all the HBAs of a given HBA set have to be generated using the same parameters, which implies that the generation of the addresses of an HBA set will occur in a coordinated fashion. In this memo, we will describe a mechanism to generate all the addresses of a given HBA set. The generation process of each one of the HBA address of an HBA set will be heavily based in the CGA generation process defined in [1]. More precisely, the HBA set generation process will be defined as a sequence of lightly modified CGA generations.

The changes required in the CGA generation process when generating a single HBA are the following: First, the Multi-Prefix Extension has to be included in the CGA Parameters Data Structure. Second, in the case that the address being generated is an HBA-only address, a random nonce (encoded in DER as an ASN.1 structure of the type SubjectPublicKeyInfo) will have to be used as input instead of a valid public key.

The resulting HBA-set generation process is the following:

The inputs to the HBA generation process are:
- o A vector of n 64-bit prefixes
- o A Sec parameter, and
- o In the case of the generation of a set of HBA/CGA addresses a public key is also provided as input (not required when generating HBA-only addresses)

The output of the HBA generation process are:
- o An HBA-set
- o their respective CGA Parameters Data Structures

The steps of the HBA-set generation process are:

1. Multi-Prefix Extension generation. Generate the Multi-Prefix Extension with the format defined in section 3. Include the vector of n 64-bit prefixes in the Prefix[1...n] fields. The Ext Len field value is (n*8). If a public key is provided, then the P

flag is set. Otherwise, the P flag is reset.

2. Modifier generation. Generate a Modifier as a random or pseudorandom 128-bit value. If a public key has not been provided as an input, generate the Extended Modifier as a 384-bit random or pseudorandom value. Format the Extended Modifier as a DER-encoded ASN.1 structure of the type SubjectPublicKeyInfo defined in the Internet X.509 certificate profile [3].

3. Concatenate from left to right the Modifier, 9 zero octets, the encoded public key or the encoded Extended Modifier (if no public key was provided) and the Multi-Prefix Extension. Execute the SHA-1 algorithm on the concatenation. Take the 112 leftmost bits of the SHA-1 hash value. The result is Hash2.

4. Compare the 16*Sec leftmost bits of Hash2 with zero. If they are all zero (or if Sec=0), continue with step (5). Otherwise, increment the modifier by one and go back to step (3).

5. Set the 8-bit collision count to zero.

6. For i=1 to n do

    6.1. Concatenate from left to right the final modifier value, Prefix[i], the collision count, the encoded public key or the encoded Extended Modifier (if no public key was provided). Execute the SHA-1 algorithm on the concatenation. Take the 64 leftmost bits of the SHA-1 hash value. The result is Hash1[i].

    6.2. Form an interface identifier from Hash1[i] by writing the value of Sec into the three leftmost bits and by setting bits 6 and 7 (i.e., the "u" and "g" bits) both to zero.

    6.3. Generate address HBA[i] by concatenating Prefix[i] and the 64-bit interface identifier to form a 128-bit IPv6 address with the subnet prefix to the left and interface identifier to the right as in a standard IPv6 address [4].

    6.4. Perform duplicate address detection if required. If an address collision is detected, increment the collision count by one and go back to step (6). However, after three collisions, stop and report the error.

    6.5. Form the CGA Parameters Data Structure that corresponds to HBA[i] by concatenating from left to right the final modifier value, Prefix[i], the final collision count value, the encoded public key or the encoded Extended Modifier and the Multi-Prefix Extension.

[Note: most of the steps of the process are taken from [1]]

5. HBA verification

HBAs are constructed as a CGA Extension, so a properly formated HBA and its correspondent CGA Parameter Data Structure will successfully finish the verification process described in section 5 of [1]. Such verification is useful when the goal is the verification of the binding between the public key and the HBA. However, for multihoming applications, it is also relevant to verify if a given HBA address belongs to a certain HBA set. An HBA set is identified by a CGA Parameter Data structure that contains a Multi-Prefix Extension. So, it is then needed to verify if an HBA belongs to the HBA set defined by a CGA Parameter Data Structure. It should be noted that it may needed to verify if an HBA belongs to the HBA set defined by the CGA Parameter Data Structure of another HBA of the set. If this is the case, the CGA verification process as defined in [1] will fail, because the prefix included in the Subnet Prefix field of the CGA Parameter data Structure will not match with the one of the HBA that is being verified. However, this not means that this HBA does not belong to the HBA set. In order to address this issue, it is only required to verify that the HBA prefix is included in prefix set defined in the Multi-Prefix Extension, and if this is the case, then substitute the prefix included in the Subnet Prefix field by the prefix of the HBA, and then preform the CGA verification process defined in [1].

So, the process to verify that an HBA belongs to an HBA set determined by a CGA Parameter Data Structure is called HBA verification and it is the following:

The inputs to the HBA verification process are:
- o An HBA
- o An CGA Parameter Data Structure

The steps of the HBA verification process are the following:

1. Verify that the 64-bit HBA prefix is included in the prefix set of the Multi-Prefix Extension. If it is not included, the verification fails. If it is included, replace the prefix contained in the Subnet Prefix field of the CGA Parameter Data Structure by the 64-bit HBA prefix.

2. Run the verification process described in section 5 of [1] with the HBA and the new CGA Parameters Data Structure as inputs. The steps of the process are included below, extracted from [1]

2.1. Check that the collision count in the CGA Parameters data structure is 0, 1 or 2. The CGA verification fails if the collision count is out of the valid range.

2.2. Check that the subnet prefix in the CGA Parameters data structure is equal to the subnet prefix (i.e., the leftmost 64 bits) of the address. The CGA verification fails if the prefix values differ. [Note: This step is trivially successful because step 1]

2.3. Execute the SHA-1 algorithm on the CGA Parameters data structure. Take the 64 leftmost bits of the SHA-1 hash value. The result is Hash1.

2.4. Compare Hash1 with the interface identifier (i.e., the rightmost 64 bits) of the address. Differences in the three leftmost bits and in bits 6 and 7 (i.e., the "u" and "g" bits) are ignored. If the 64-bit values differ (other than in the five ignored bits), the CGA verification fails.

2.5. Read the security parameter Sec from the three leftmost bits of the 64-bit interface identifier of the address. (Sec is an unsigned 3-bit integer.)

2.6. Concatenate from left to right the modifier, 9 zero octets, and the public key, and any extension fields that follow the public key in the CGA Parameters data structure. Execute the SHA-1 algorithm on the concatenation. Take the 112 leftmost bits of the SHA-1 hash value. The result is Hash2.

2.7. Compare the 16*Sec leftmost bits of Hash2 with zero. If any one of them is non-zero, the CGA verification fails. Otherwise, the verification succeeds. (If Sec=0, the CGA verification never fails at this step.)

6. Security considerations

The goal of HBAs is to create a group of addresses that are securely bound, so that they can be used interchangeably when communicating with a node. If there is no secure binding between the different addresses of a node, a number of attacks are enabled, as described in [5]. It particular, it would possible for an attacker to redirect the communications of a victim to an address selected by the attacker, hijacking the communication. When using HBAs, only the addresses belonging to an HBA set can be used interchangeably, limiting the addresses that can be used to redirect the communication to a well, pre-determined set, that belongs to the original node involved in the communication. So, when using HBAs, a node that is communicating using address A can redirect the communication to a new address B if and only if B belongs to the same HBA set than A.

This means that if an attacker wants to redirect communications addressed to address HBA1 to an alternative address IPX, the attacker will need to create a CGA Parameters data structure that generates an HBA set that contains both HBA1 and IPX.

In order to generate the required HBA set, the attacker needs to find a CGA Parameter data structure that fulfills the following conditions:

- o the prefix of HBA1 and the prefix of IPX are included in the Multi-Prefix Extension
- o HBA1 is included in the HBA set generated.

(this assumes that it is acceptable for the attacker to redirect HBA1 to any address of the prefix of IPX).

The remaining fields that can be changed at will by the attacker in order to meet the above conditions are: the Modifier, other prefixes in the Multi-Prefix Extension and other extensions. In any case, in order to obtain the desired HBA set, the attacker will have to use a brute force attack, which implies the generation of multiple HBA sets with different parameters (for instance with a different Modifier) until the desired conditions are meet. The expected number of times that the generation process will have to be repeated until the desired HBA set is found is exponentially related with the number of bits containing hash information included in the interface identifier of the HBA. Since 59 of the 64 bits of the interface identifier contain hash bits, then the expected number of generations that will have to be performed by the attacker are O(2^59).

The protection against brute force attacks can be improved increasing the Sec parameter. A non zero Sec parameter implies that steps 3-4 of the generation process will be repeated O(2^(16*Sec)) times

(expected number of times). If we assimilate the cost of repeating the steps 3-4 to the cost of generating the HBA address, we can estimate the number of times that the generation is to be repeated in O(2^(59+16*Sec)).

Interaction with IPSec. In the case that both IPSec and CGA/HBA address are used simultaneously, it is possible that two public keys are available in a node, one for IPSec and another one for the CGA/HBA operation. In this case, an improved security can be achieved by verifying that the keys are related somehow, (in particular if the same key is used for both purposes).

7. Contributors

This document was originally produced of a MULTI6 design team consisting of (in alphabetical order): Jari Arkko, Marcelo Bagnulo Braun, Iljitsch van Beijnum, Geoff Huston, Erik Nordmark, Margaret Wasserman, and Jukka Ylitalo.

8. Acknowledgments

The initial discussion about HBA benefited from contributions from Alberto Garcia-Martinez, Tuomas Aura and Arturo Azcorra.

The HBA-set generation and HBA verification processes described in this document contain several steps extracted from [1].

9. References

9.1 Normative References

[1] Aura, T., "Cryptographically Generated Addresses (CGA)", draft-ietf-send-cga-06 (work in progress), April 2004.

[2] Arkko, J., Kempf, J., Sommerfeld, B., Zill, B. and P. Nikander, "SEcure Neighbor Discovery (SEND)", draft-ietf-send-ndopt-06 (work in progress), July 2004.

[3] Housley, R., Polk, W., Ford, W. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[4] Hinden, R. and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

9.2 Informative References

[5] Nordmark, E., "Threats relating to IPv6 multihoming solutions", draft-ietf-multi6-multihoming-threats-01 (work in progress), September 2004.

[6] Nikander, P., Arkko, J., Aura, T., Montenegro, G. and E. Nordmark, "Mobile IP version 6 Route Optimization Security Design Background", draft-ietf-mip6-ro-sec-01 (work in progress), July 2004.

[7] Haddad, W., Madour, L., Arkko, J. and F. Dupont, "Applying Cryptographically Generated Addresses to Optimize MIPv6 (CGA-OMIPv6)", draft-haddad-mip6-cga-omipv6-02 (work in progress), June 2004.

Author's Address

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
SPAIN

Phone: 34 91 6249500
EMail: marcelo@it.uc3m.es
URI: http://www.it.uc3m.es

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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