IPv6 Security

M Babik[1], J Chudoba[2], A Dewhurst[3], T Finnern[4], T Froy[5], C Grigoras[1], K Hafeez[3], B Hoeft[6], D P Kelsey[3], F López Muñoz[7], E Martelli[1], R Nandakumar[3], K Ohrenberg[4], F Prelz[8], D Rand[9], A Sciabà[1], D Traynor[5], U Tigerstedt[10] and R Wartel[1]

[1] CERN, CH-1211 Genève 23, Switzerland
[2] Institute of Physics, Academy of Sciences of the Czech Republic Na Slovance 2 182 21 Prague 8, Czech Republic
[3] STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, Oxfordshire OX11 0QX, United Kingdom
[4] Deutsches Elektronen-Synchrotron, Notkestraße 85, D-22607 Hamburg, Germany
[5] Queen Mary University of London, Mile End Road, London E1 4NS, United Kingdom
[6] Karlsruher Institut für Technologie, Hermann-von-Helmholtz-Platz 1, D-76344 Eggenstein-Leopoldshafen, Germany
[7] Port d'Informació Científica, Campus UAB, Edifici D, E-08193 Bellaterra, Spain
[8] INFN, Sezione di Milano, via G. Celoria 16, I-20133 Milano, Italy
[9] Imperial College London, South Kensington Campus, London SW7 2AZ, United Kingdom
[10] CSC Tieteen Tietotekniikan Keskus Oy, P.O. Box 405, FI-02101 Espoo

E-mail: `david.kelsey@stfc.ac.uk, ipv6@hepix.org`

Abstract. IPv4 network addresses are running out and the deployment of IPv6 networking in many places is now well underway. Following the work of the HEPiX IPv6 Working Group, a growing number of sites in the Worldwide Large Hadron Collider Computing Grid (WLCG) have deployed dual-stack IPv6/IPv4 services. The aim of this is to support the use of IPv6-only clients, i.e. worker nodes, virtual machines or containers.

The IPv6 networking protocols while they do contain features aimed at improving security also bring new challenges for operational IT security. We have spent many decades understanding and fixing security problems and concerns in the IPv4 world. Many WLCG IT support teams have only just started to consider IPv6 security and they are far from ready to follow best practice, the guidance for which is not easy to find. The lack of maturity of IPv6 implementations together with the increased complexity of the protocol standards and the fact that the new protocol stack allows for pretty much the same attack vectors as IPv4, raise many new issues for operational security teams.

The HEPiX IPv6 Working Group is producing guidance on best practices in this area. This paper will consider some of the security concerns for WLCG in an IPv6 world and present the HEPiX IPv6 working group guidance both for the system administrators who manage IT services on the WLCG distributed infrastructure and also for their related security and networking teams.

1. Introduction

The much-heralded exhaustion of the IPv4 networking address is with us, etc. etc.

2. IPv6 security issues

IPv6 Security issues

New features

Many more ICMP message types! Cannot filter all of them (MTU discovery has to work). Must filter some of them. RFC4890 gives advice

New methods for autoconfiguring addresses, routes, DNS. Good for the end-user. Must do something against rogue Router Advertisements (see RFC6104)

Longer IP addresses. Hey, everyone knows that. They may slow down brute force scans. But no bad guy is that crude...

Cannot fragment packets en-route. Minimum MTU: 1280. But you can still hurt yourself and send small fragments if you wish. Some good news, at least

Not really a feature of IPv6 proper, but much of the network stack and application code is enticingly fresh!

Transitional technologies (e.g. tunnels) have intrinsic vulnerabilities but don't need to be there forever...

Business as usual

As long as all network monitoring and administration tools are up-to-date and (therefore) aware of IPv6.

Broadcasts and Multicasts are still there, with a vengeance. Can still use IP headers for out-of-band communications. Can still pollute Ethernet address discovery (ND instead of ARP). Can still run a rogue DHCP server. Can still try forging and injecting packets into the local network. Upper-layer protocols did not change!

3. Checklist for system administrators

(i) Ensure all security/network monitoring/logging are IPv6-capable

(ii) Filter IPv6 packets that enter and leave your network/system

(iii) Filter/disable IPv6-on-IPv4 tunnels

(iv) Deploy RA-Guard or otherwise deal with Rogue RAs

(v) Filter ICMPv6 messages wisely

(vi) Allow special-purpose headers only if needed

(vii) Make an addressing plan

(viii) Decide whether to use DHCPv6 or SLAAC+DynDNS

(ix) Use synchronised IPv4/v6 access rules

(x) Do not be tempted by transition technologies

4. Checklist for developers

When applications developed in the golden era of IPv4-only Internet face the transition to IPv6, the brunt of the work often falls on the shoulders of developers, who often belong to a different generation as the original authors. Figure 1 tries to visualise the extent of the changes that the core code of any IP-capable application undergoes in the transition. In addition to the extensively different syntax, there is a fundamental $1 \rightarrow N$ change here: no IP endpoint can be satisfied with handling just *one* IP address (as any public IPv6 endpoint communicates via the public and on the link-local address at least), but loops and address ordering start appearing everywhere. We identify the following implications of this fundamental fact on developers' practice, in rough descending order of importance:

(i) The *syntactic* change of the core IP networking code in the IPv4→IPv6 transition is large enough to oftentime justify the refactoring of larger portions of code. The *semantic* $1 \rightarrow N$

```
struct hostent *resolved_name=NULL;
struct servent *resolved_serv=NULL;
struct protoent *resolved_proto=NULL;
static char *dest_host="some.ip.host", *dest_serv="ipservice";
struct sockaddr_in destination;
resolved_host = gethostbyname(dest_host);
resolved_serv = getservbyname(dest_serv, NULL);
if (resolved_host != NULL && resolved_serv != NULL) {
     destination.sin_family = resolved_name->h_addrtype;
     destination.sin_port = htons(resolved_serv->s_port);
     memcpy(&destination.sin_addr, resolved_host->h_addr_list[0],
          resolved_host->h_length);
     resolved_proto = getprotobyname(resolved_serv->s_proto)
     if (resolved_proto != NULL) {
          int fd = socket(AF_INET, SOCK_STREAM, resolved_proto->p_proto);
          connect(fd, &destination, sizeof(destination));
          /* Check for errors, connect, etc...  */
     }
}
```

```
struct addrinfo ai_req, *ai_ans, *cur_ans;
static char *dest_host="some.ip.host", *dest_serv="ipservice";
ai_req.ai_flags = 0;
ai_req.ai_family = PF_UNSPEC;
ai_req.ai_socktype = SOCK_STREAM;
ai_req.ai_protocol = 0; /* Any protocol is OK */
if (getaddrinfo(dest_host, dest_serv, &ai_req, &ai_ans) != 0) {
     for (cur_ans = ai_ans; cur_ans != NULL; cur_ans = cur_ans->ai_next) {
          int fd = socket(cur_ans->ai_family, cur_ans->ai_socktype,
               cur_ans->ai_protocol);
          connect(fd_socket,cur_ans->ai_addr,cur_ans->ai_addrlen);
          /* Check for errors - This loop has the ability to change the */
          /* order of the getaddrinfo results!  */
     }
}
```

Figure 1. C code snippets showing how the basic IP service resolution and connection changes from legacy IPv4-only to a dual-stack or IPv6-only environment. This represents the zeroth-order porting effort for much IPv4-only code. The newer structure is more terse, but the changes are extensive enough, both syntactically and semantically, to probably trigger the refactoring of much larger sections of code.

change may be *forcing* some rethiking at the design level. A possible temptation here is to provide parallel sections of code that handle the IPv6 case only: a few other reasons why this may not be a good idea are listed below. In any case, there is an implicit expectation that a change that should be affecting the *transport* layer of the network only should cause no ripple in the upper layers, i.e. that the perceived responsiveness, performance and reliability of the code remain unchanged. *Extensive* stress-testing should therefore be planned on IPv6-ported code.

(ii) Code that binds and connects IP sockets is suddenly faced with making choices that used to be delegated to the operating system or networking-capable libraries. Lists of addresses may be received in a given order, but it's now the responsibility of the socket-handling code to iterate and re-iterate on the list, handle exceptions and possibly operate in parallel on various entries to implement some form of 'happy-eyeballs'[1] alghorithm. As the ordering of both source and destination addresses established at the system level by the system administrator[2] may have security implications, developers should go the extra mile to keep that ordering even if they have to reshuffle the list for any reason. Applications should allow users to prefer/enable either IPv4 or IPv6 via configuration, but should always honor the system-level administrator's choice by default.

(iii) Fresh new code that hasn't been tested broadly and in the wild is *per se* attractive to anyone looking for malicious exploits. Especially in the case where IPv6-specific code or processes are developed for *parallel* deployment with well-proven IPv4 code, one should make sure that any security measure, filter or wisdom that was included in the code for the IPv4 case isn't simply forgotten for IPv6. While it may not be immediately apparent, *all* constructs that are meaningful for IPv4 have their translation or counterpart for IPv6.

References

[1] `http://hepix-ipv6.web.cern.ch`

[2] All Internet Engineering Task Force Requests For Comments (RFC) documents are available from URLs such as http://www.ietf.org/rfc/rfcNNNN.txt where NNNN is the RFC number, for example `http://www.ietf.org/rfc/rfc2460.txt`

[3] See for instance `http://www.google.com/ipv6/statistics.html`. The 10% global connectivity threshold was crossed in January 2016.

[4] Hogg S, Vyncke E - IPv6 Security, Cisco Press 2009, ISBN-13: 978-1-58705-594-2

[1] See RFC6555 [2].

[2] Via `/etc/gai.conf`, `ip addrlabel` or their equivalent.