shim6 implementation

Installation Guide and User's Manual

Authors:
Jeff Ahrenholz
Tom Henderson

This software was developed under funding supplied by ONR contract N00014-06-C-0319 and The Boeing Company.

Table of Contents

1 Introduction

shim6 (Site Multihoming by IPv6 Intermediation) is a protocol being developed in the Internet Engineering Task Force (IETF) to provide a host-based solution to allow IPv6 sites to multi-home (connect to more than one Internet provider) for load sharing, redundancy, and fault tolerance. The shim6 protocol allows such hosts to hold multiple *locators* (addresses) and to migrate transport protocol sessions from one locator pair to another without disruption. The protocol includes components for securely (cryptographically) binding locator sets together to prevent address hijacking, and a separate reachability protocol (REAP) that can monitor the reachability of peer locators in active use.

shim6 is architecturally related to the Host Identity Protocol (HIP), in that they both implement an identifier/locator split at the boundary between the network and transport layer. Their control protocols also share many of the same packet formats.

1.1 Goals

The goals of this shim6 effort have been to develop an open-source Linux implementation of shim6 that borrows from (and can later integrate with) our HIP implementation. We are interested in the long-term potential of the ID/locator split architecture and in eventual evolution towards some kind of a combined shim6/HIP implementation:

- we envision that components of shim6 may be useful to HIP (particularly REAP);
- shim6 has some nice features such as deferred context establishment and support for referrals without lookups that make it attractive for light-weight and near-term deployment;

1.1.1 Non-goals

The following goals are deferred for a later phase of development:

- context forking and balancing across multiple paths
- policy granularity of when to invoke/bypass shim6 processing
- HBA and CGA generation and verification
- IPV6_DONTSHIM option (and other shim6-api extensions for applications)
- support for UDP tunneling and other tunneling support
- RFC3484-bis source address selection modifications for shim6
- Windows and OS X support, or user-mode support

1.2 Software architecture

Our shim6 implementation is a multithreaded C++ user-space processes that interfaces with a standard Linux kernel (extended via a loadable kernel module) through standard system interfaces and with a GUI-based status application via sockets IPC. Although we do not require kernel patches, we do require specific kernel versions since we are working with kernel modules that do not have stable APIs across kernel releases.

Although some components of our architecture are defined for multi-platform support, we currently focus on Linux. Figure 1.1 illustrates the overall software architecture. Cross-layer interaction with TCP (e.g., resetting congestion window upon locator change) is not shown and needs further consideration whether it can be done without a kernel modification.

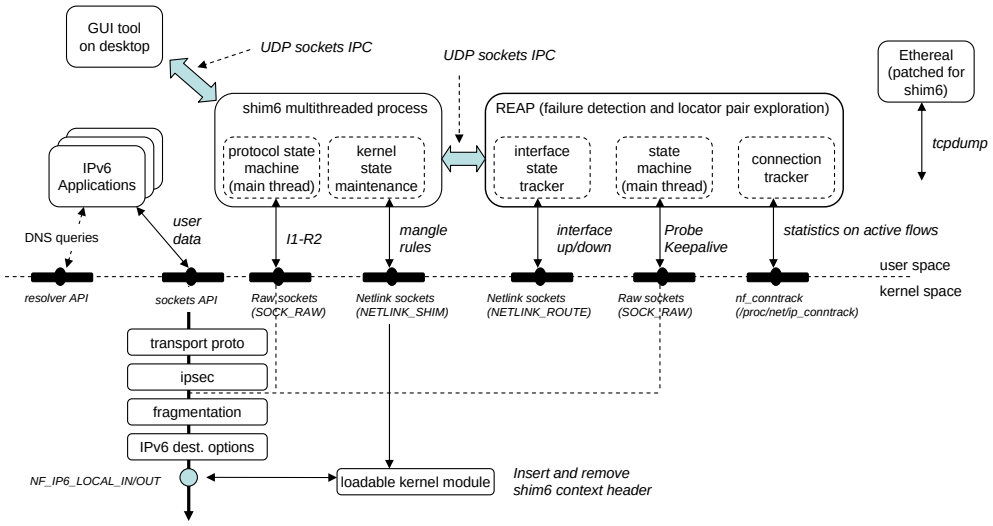

Figure 1.1: Overall shim6 implementation architecture.

The overall strategy is to use loadable Netfilter mangle modules in the IPv6 firewall tables to implement the shimming and deshimming of packets, to use Netfilter queue to receive shim6 control packets, and the Linux Netfilter conntrack (connection tracking) feature to track active flows. The shim6 daemon itself is a process that shares a HIP-abstraction library with our HIP daemon, and the failure detection daemon (reapd) is designed to be reused across HIP and shim6.

Below is an excerpt from Section 4 of the shim6 protocol draft. The above Figure 1.1 can probably best be understood by walking through what happens in the software to cause the following shim6 behavior to occur.

```
The shim6 protocol operates in several phases over time.  The
following sequence illustrates the concepts:

o  An application on host A decides to contact B using some upper-
   layer protocol.  This results in the ULP on A sending packets to
   B. We call this the initial contact.  Assuming the IP addresses
```

```
selected by Default Address Selection [12] and its extensions [13]
work, then there is no action by the shim at this point in time.
Any shim context establishment can be deferred until later.
```

No action.

```
o  Some heuristic on A or B (or both) determine that it is
   appropriate to pay the shim6 overhead to make this host-to-host
   communication robust against locator failures.  For instance, this
   heuristic might be that more than 50 packets have been sent or
   received, or a timer expiration while active packet exchange is in
   place.  This makes the shim initiate the 4-way context
   establishment exchange.

   As a result of this exchange, both A and B will know a list of
   locators for each other.

   If the context establishment exchange fails, the initiator will
   then know that the other end does not support shim6, and will
   continue with standard unicast behavior for the session.
```

Netfilter connection tracking is used to implement the supported heuristic. A thread in the REAP user-space software can monitor this according to policy and invoke a shim6 context establishment and locator exchange (by telling the shim6 daemon) when necessary.

The above step just builds state (ULID/locator context) in the shim6 daemon but does not affect any data plane packets.

However, the reachability protocol (REAP) starts to monitor the ULIDs for reachability, and should be provided with a list of alternate locators learned during the shim6 handshake.

```
o  Communication continues without any change for the ULP packets.
   In particular, there are no shim extension headers added to the
   ULP packets, since the ULID pair is the same as the locator pair.
   In addition, there might be some messages exchanged between the
   shim sub-layers for (un)reachability detection.

o  At some point in time something fails.  Depending on the approach
   to reachability detection, there might be some advice from the
   ULP, or the shim (un)reachability detection might discover that
   there is a problem.

   At this point in time one or both ends of the communication need
   to probe the different alternate locator pairs until a working
   pair is found, and switch to using that locator pair.
```

REAP process or thread separately does this probing, and notifies shim6 software via sockets IPC when a ULID should be replaced with alternate locator(s).

When notified, the host needs to assume that packets may start arriving on that shim context or may need to be sent on new shim context.

If the local host has new dest locators, a two step process is needed for each active locator that is != dst ULID:

1. add a rule to divert packets to a SHIM6 module
   `ip6tables -A INPUT <matching rule> -j SHIM6`
2. add context state to the SHIM6 module

If the local host has new src locators, a two step process will be needed for each active locator that is != src ULID:

1. add a rule to divert packets to a SHIM6 module
   `ip6tables -A OUTPUT <matching rule> -j SHIM6`
2. add context state to the SHIM6 module

> o Once a working alternative locator pair has been found, the shim will rewrite the packets on transmit, and tag the packets with shim6 Payload extension header, which contains the receiver's context tag. The receiver will use the context tag to find the context state which will indicate which addresses to place in the IPv6 header before passing the packet up to the ULP. The result is that from the perspective of the ULP the packet passes unmodified end-to-end, even though the IP routing infrastructure sends the packet to a different locator.

Nothing needed, until connection being shimmed goes away or the locator pair changes again. There needs to be some garbage collection of shim state not being used for a long time. Again, Netfilter connection tracking could be used for this.

> o The shim (un)reachability detection will monitor the new locator pair as it monitored the original locator pair, so that subsequent failures can be detected.

The above is a function of the REAP protocol.

> o In addition to failures detected based on end-to-end observations, one endpoint might know for certain that one or more of its locators is not working. For instance, the network interface might have failed or gone down (at layer 2), or an IPv6 address might have become deprecated or invalid. In such cases the host can signal its peer that this address is no longer recommended to try. Thus this triggers something similar to a failure handling in that a new, working locator pair must be found.

The above is a function of the REAP protocol.

> The protocol also has the ability to express other forms of locator preferences. A change in any preferences can be signaled to the peer, which will made the peer record the new preferences. A change in the preferences might optionally make the peer want to use a different locator pair. If it makes this decision, it follows the same locator switching procedure as after a failure

(by verifying that its peer is indeed present at the alternate locator, etc).

This exchange will take part in the shim6 daemon, and updates are pushed to the REAP protocol. Change in local locator status may be monitored by the Netlink ROUTING socket.

o When the shim thinks that the context state is no longer used, it can garbage collect the state; there is no coordination necessary with the peer host before the state is removed. There is a recovery message defined to be able to signal when there is no context state, which can be used to detect and recover from both premature garbage collection, as well as complete state loss (crash and reboot) of a peer.

SHIM6 kernel module needs to be able to send this NO_CONTEXT message to shim6 daemon.

The exact mechanism to determine when the context state is no longer used is implementation dependent. An implementation might use the existence of ULP state (where known to the implementation) as an indication that the state is still used, combined with a timer (to handle ULP state that might not be known to the shim sub-layer) to determine when the state is likely to no longer be used.

NOTE: The ULP packets in shim6 can be carried completely unmodified as long as the ULID pair is used as the locator pair. After a switch to a different locator pair the packets are "tagged" with a shim6 extension header, so that the receiver can always determine the context to which they belong. This is accomplished by including an 8-octet shim6 Payload Extension header before the (extension) headers that are processed by the IP endpoint sublayer and ULPs. If subsequently the original ULIDs are selected as the active locator pair then the tagging of packets with the shim6 extension header can also be stopped.

1.3 Implementation status

Currently, this implementation supports the following:

- Four-packet I1, R1, I2, R2 shim6 context establishment
- Sends periodic keep-alive packets
- Probes locator pairs using the Probe packet, using the Probes Sent and Probes Received fields
- Shims and deshims data packets based on context establishment and reachability state
- Communicates local address list in the I2 and R2 messages
- Handling of the ULID Pair option
- Context tag and nonce management

1.4 Future work

These items are left for future revisions:

- Garbage collection and cleanup of context
- Packet retransmission with binary exponential backoff
- Context recovery, context confusion, and the R1bis and I2bis packets
- R1 validator and Locator List options
- Handling fragmentation of packets that exceed the MTU size after the IPv6 extension header has been added (note: we have a workaround in place in our shimming module)
- shim6 Error, Update Request, and Update Acknowledgment messages
- Keepalive Timeout option
- Handling ICMP error messages and positive or negative feedback from upper layer protocols
- Communication from the kernel module to the shim6 daemon about packets received with a context tag that does not match
- Context forking and balancing across multiple paths; Forked Instance Identifier option
- Generate and validate cryptographically bound addresses such as CGA or HBA; CGA Parameter Data Structure and CGA Signature options
- Native shim6 API for applications, such as the IPV6_DONTSHIM option
- Support for UDP tunneling and other tunneling support
- RFC5014 source address selection rules.

2 Installation

2.1 Prerequisites

Background information on shim6 is available at these links:

- http://tools.ietf.org/wg/shim6/
- http://www.shim6.org/

2.1.1 Operating Systems

This software has only been tested on Linux i386 systems. There are several Linux kernel dependencies that are known to be unstable across recent kernel versions. We have tested using 2.6.20 kernels on Ubuntu and Fedora Linux.

2.2 Installing supporting libraries

2.2.1 Overview

The following open source packages and programs are needed for shim6:

- **socklibpp:** provides (more) portable libraries for socket programming
- **conntrack:** user-space utility for tracking the number of packets in a flow
- **libnetfilter_conntrack:** netfilter library to access conntrack
- **libnetfilter_queue:** netfilter library for receiving SHIM6 control packets in a user-space queue from the match rule
- **libnfnetlink:** low-level netfilter library required by the others

These dependencies are part of this software distribution:

- **hal (HIP Abstraction Library):** platform independent abstraction library for HIP and shim6, mainly providing sockets IPC
- **ip6t_shim6:** ip6tables match rule to catch SHIM6 packets, and send them to the control netfilter queue or to ip6t_shim6_data
- **ip6t_shim6_data:** ip6tables target to shim and deshim data packets

2.2.2 Supporting libraries

The following libraries are necessary. We recommend that you install these from a source package because some of the RPM and DEB packages for Linux distributions are incomplete (even development packages).

- patch socklibpp with our IPv6 patch
  - check out the library from soureforge's anonymous CVS
  - patch, compile and install it
- install conntrack-tools. This has been tested with version 0.9.5. However, if your Linux distribution already supports conntrack, installation of conntrack-tools is not necessary.
- install libnfnetlink (version 0.0.30 or later)
  - `./configure && make && sudo make install`

  - need a symlink from `/usr/lib/libnfnetlink.so.0 -> /usr/local/lib/libnfnetlink.so.0` if you leave the default prefix.
- install libnetfilter_queue (version 0.0.15 or later)
  - this library requires libnfnetlink (above)
  - `./configure && make && sudo make install`
  - need a symlink from `/usr/lib/libnetfilter_queue.so.1 -> /usr/local/lib/libnetfilter_queue.so.1` if you leave the default prefix
  - On Fedora Linux, you need to build with: `PKG_CONFIG_PATH=../libnfnetlink-0.0.25 ./configure ...`
- install libnetfilter_conntrack (version 0.0.81 or later)
  - `./configure && make && sudo make install`
  - need a symlink from `/usr/lib/libnetfilter_conntrack.so.1 -> /usr/local/lib/libnetfilter_conntrack.so.1` if you leave the default prefix
- install hal (HIP Abstraction Layer) library provided in this distribution
  - `./bootstrap.sh && ./configure && make && sudo make install`
  - need a symlink from `/usr/lib/libhal-0.1.so -> /usr/local/lib/libhal-0.1.so`

2.2.3 ip6tables extensions

The kernel-level firewall tool for IPv6, ip6tables, has been extended in two ways to support SHIM6. There is a **shim6 match** module for receiving SHIM6 control and data packets, and a **shim6_data target** for inserting and removing the shim layer from data packets.

1. build and install both shim6 ip6tables kernel modules
   - untar the hip-modules tarball:

```
cd hip-modules/ip6t_shim6
make
sudo ip6tables -L
sudo insmod -f ./ip6t_shim6.ko
sudo insmod -f ./ip6t_shim6_data.ko
```

2. build and install the iptables libraries – for extending the ip6tables command
   - this adds a dynamically-loaded extension to your ip6tables command to support the shim6 match and shim6_data target
   - this is really only needed if you wish to properly display the SHIM6 rules or manually modify them
   - first edit the Makefile so that `IP6T_VER` matches `ip6tables -V`; also on Fedora, change `LD=ld` (for Ubuntu you need to use `LD=cc`)
   -

```
cd hip-modules/iptables
make
sudo make install
```

   - "notes:" the shared libraries are installed in `/lib/iptables/`

The following is performed automatically by the shim6 daemon, but included here for reference.

- setting up a match rule to send SHIM6 control messages to the user-space shim6d daemon: `sudo ip6tables -A INPUT -m shim6 --control -j NFQUEUE --queue-num 1`
- setting up an inbound match + target rule to deshim SHIM6 data packets:

```
sudo ip6tables -t mangle -A INPUT -m shim6 --data \\
--mcid 0x010203040506 -j shim6_data --cid 0x010203040506 \\
--src_loc src_ulid --dst_loc dst_ulid
```

- setting up an outbound target rule to shim certain flows:

```
sudo ip6tables -t mangle -A INPUT --src src_ulid --dst dst_ulid \\
-j shim6_data --cid 0xa1a2a3a4a5a6 --src_loc src_locator \\
--dst_loc dst_locator
```

2.3 Installing shim6

2.3.1 REAP daemon

This implements the REAchability Process that monitors local address information, remote peer status, and takes care of verifying reachability.

- compile the reap daemon code

```
./bootstrap.sh && ./configure && make
```

- before starting reapd, it may be useful to initialize the kernel's iptables and connection tracking:

```
sudo ip6tables -F
sudo modprobe ng_conntrack_ipv6
sudo conntrack -F
```

- now you can run reapd with:

```
sudo src/reapd/reapd
```

2.3.2 SHIM6 daemon

The SHIM6 daemon implements the SHIM6 protocol, exchanging SHIM6 control messages and inserting context state into the kernel, using both the REAP daemon and the HAL library. The REAP daemon should be running first.

- compile the shim6 daemon code

```
./bootstrap.sh && ./configure && make
sudo src/shim6d/shim6d
```

- the SHIM6 daemon manages the ip6tables rules, so you do not need to manually manipulate them; upon normal exit, shim6 ip6tables rules are removed

3 Running shim6

3.1 Overview

shim6 requires two processes and a number of modules to be running:

- `ip6t_shim6.ko` (kernel module),
- `ip6t_shim6_data.ko` (kernel module),
- `nf_conntrack_ipv6`, (kernel module)
- `reapd` (user-space program), and
- `shim6d`

If you performed a `make install` in the previous installation step, these executables are likely in your `/usr/local/sbin` path:

Here is a sample list of commands that could be executed:

```
cd /usr/local/sbin
sudo ip6tables -F
sudo modprobe nf_conntrack_ipv6
sudo insmod -f ./ip6t_shim6.ko
sudo insmod -f ./ip6t_shim6_data.ko
sudo reapd
sudo shim6d
```

Of course, all of the above could be somewhat automated in an init script.

3.2 Testing shim6

So now what? shim6 and reap are running, but are they doing anything?

Below are some initial steps that can be taken to exercise your implementation.

- Normal triggering using conntrack
  - if you've installed the conntrack libraries, shim6d will automatically receive notifications of new IPv6 connections from the kernel; after 45 packets it will attempt a SHIM6 context exchange
  - you may want to add multiple IPv6 locators to your network interface before starting
  - conntrack tips: `sudo conntrack -F` (flush the conntrack table), `sudo conntrack -f ipv6 -L` (view IPv6 entries)
  - you could establish a SHIM6 context using ssh, for example

    ```
    ssh <dst-ulid>
    ```

  - after 45 packets, a SHIM6 context exchange should occur, and the inbound deshimming rules are installed; the packets flow as usual without any shim since there is no failure
  - if you use unidirectional UDP traffic (for example a VLC UDP video stream), you should see SHIM6 keepalives transmitted periodically in the opposite direction of the flow of traffic

- now you can try blocking one direction of communication to produce a failure

```
ip6tables -I OUTPUT --dst <dst-ulid> -j DROP
```

- after detecting a path failure, reapd will begin its probing process, and choose another working pair if available
- the previous connection using the ULIDs should still be functional, and now contain the shim6 IPv6 extension header and use a new working locator pair

- Debugging port on UDP 19384
  - Once shim6d is running on both hosts you can manually trigger a SHIM6 context establishment handshake using a local debugging port (uses the `netcat` utility):

```
nc -u localhost 19384
```

  - the debugging port supports the following commands:

```
# dump shim6 contexts
dump
# trigger context exchange
context <ipv6-src-ulid> <ipv6-dst-ulid>
# manually delete context for the provided ulids
delete <ipv6-src-ulid> <ipv6-dst-ulid>
# update context (send update and possibly cause shimming)
update <context_id>
# manually insert mangle rule for adding an outbound shim
shim <context_id> <ipv6-src-loc> <ipv6-dst-loc>
# manually insert mangle rule for removing inbound shims
deshim <context_id>
```