#include
#include "ROVppAS.h"
#include "NegativeAnnouncement.h"
#include "Prefix.h"

ROVppAS::~ROVppAS() {
delete incoming_announcements;
delete anns_sent_to_peers_providers;
delete all_anns;
delete peers;
delete providers;
delete customers;
delete member_ases;
delete as_graph;
}

bool ROVppAS::pass_rov(Announcement &ann) {
if (ann.origin == attacker_asn) {
return false;
} else {
return true;
}
}

/** Push the received announcements to the incoming_announcements vector.
*

- Note that this differs from the Python version in that it does not store

- a dict of (prefix -> list of announcements for that prefix).
  *

- @param announcements to be pushed onto the incoming_announcements vector.
  / void ROVppAS::receive_announcements(std::vector &announcements) { for (Announcement &ann : announcements) { // Check if it's a NegativeAnnouncement if (NegativeAnnouncement d = dynamic_cast>(&ann)) { // Check if you have an alternative route (i.e. escape route) // WARNING: This is the first thing that must be done in this scope std::pair> check = received_valid_announcement(ann);
  if (check.first) {
  // Check if the attaker's path and this path intersect
  // MARK: Can this be done in practice?
  // TODO: ann should be the hijacked announcement (modify NegativeAnnouncement to save

the HijackedAnn)
if (paths_intersect(*(check.second), ann)) {
make_negative_announcement_and_blackhole(ann, *(check.second));
}
} else { // You don't have an escapce route
make_negative_announcement_and_blackhole(ann, *(check.second));
}
} else { // It's not a negative announcement (i.e. it's just a regular annoucement)
// Check if the Announcement is valid
if (pass_rov(ann)) {
// Do not check for duplicates here
// push_back makes a copy of the announcement
incoming_announcements->push_back(ann);

```
// Check if should make negative announcement
// Check if you have received the Invalid Announcement in the past
std::pair<bool, Announcement*> check = received_hijack_announcement(ann);
if (check.first) {
  // Check if the attaker's path and this path intersect
  // MARK: Can this be done in practice?
  if (paths_intersect(ann, *(check.second))) {
    make_negative_announcement_and_blackhole(ann, *(check.second));
  } // TODO: You should remove the blocked announcement if you made a negative announces
}
```

} else { // Not valid, so you drop the announcement (i.e. don't add it to
incoming_announcements)
// Add announcement to dropped list
dropped_ann_map[ann.prefix] = ann;
// Check if should make negative announcement
// Check if you have received the Valid announcement in the past
std::pair<bool, Announcement> *check = received_valid_announcement(ann); if (check.first) {*
// Also check if the attaker's path and this path intersect if (paths_intersect((check.second),
ann)) {
make_negative_announcement_and_blackhole(*(check.second), ann);
}
}
}
}
}
}

```
/** Make negative announcement AND Blackhole's that announcement
*/
void ROVppAS::make_negative_announcement_and_blackhole(Announcement &legit_ann,
Announcement &hijacked_ann) {
// Create Negative Announcement
// TODO: Delete following block of code after testing
// Prefix<> s("137.99.0.0", "255.255.255.0");
// Prefix<> p("137.99.0.0", "255.255.0.0");
// NegativeAnnouncement neg_ann = NegativeAnnouncement(13796, p.addr, p.netmask, 22742,
std::set>());
// NegativeAnnouncement neg_ann = NegativeAnnouncement(uint32_t aorigin,
// uint32_t aprefix,
// uint32_t anetmask,
// uint32_t from_asn,
// std::set> n_routed);
NegativeAnnouncement neg_ann = NegativeAnnouncement(legit_ann.origin,
legit_ann.prefix.addr,
legit_ann.prefix.netmask,
legit_ann.priority,
asn,
false,
std::set>(),
hijacked_ann);
neg_ann.null_route_subprefix(hijacked_ann.prefix);
// Add to set of negative_announcements
negative_announcements.insert(neg_ann);
// Add to Announcements to propagate
incoming_announcements->push_back(neg_ann);
// Add Announcement to blocked list (i.e. blackhole list)
blackhole_map[hijacked_ann.prefix] = hijacked_ann;
}

std::pair ROVppAS::received_valid_announcement(Announcement &announcement) {
// Check if it's in all_anns (i.e. RIB in)
for (std::pair, Announcement> ann : *all_anns) {
if (ann.first < announcement.prefix && ann.second.origin == victim_asn) {
return std::make_pair(true, &(ann.second));
}
}

// Check if it was just received and hasn't been processed yet
for (Announcement ann : *incoming_announcements) {
```

```
if (ann.prefix < announcement.prefix && ann.origin == victim_asn) {
return std::make_pair(true, &ann);
}
}

// Otherwise
return std::make_pair(false, nullptr);
}

std::pair ROVppAS::received_hijack_announcement(Announcement &announcement) {
// Check if announcement is in dropped list
for (std::pair, Announcement> prefix_ann_pair : dropped_ann_map) {
if (announcement.prefix < prefix_ann_pair.first && prefix_ann_pair.second.origin == attacker_asn)
{
return std::make_pair(true, &prefix_ann_pair.second);
}
}

// Check if the announcement is in the blackhole list
for (std::pair, Announcement> prefix_ann_pair : blackhole_map) {
if (announcement.prefix < prefix_ann_pair.first && prefix_ann_pair.second.origin == attacker_asn)
{
return std::make_pair(true, &prefix_ann_pair.second);
}
}

// otherwise
return std::make_pair(false, nullptr);
}

bool ROVppAS::paths_intersect(Announcement &legit_ann, Announcement &hijacked_ann) {
// Get the paths for the announcements
std::vector legit_path = get_as_path(legit_ann);
std::vector hijacked_path = get_as_path(hijacked_ann);
// Double for loop to check over entire (except the beginning and end of each path)
for (std::size_t i=1; i<legit_path.size(); ++i) {
for (std::size_t j=1; j<hijacked_path.size(); ++j) {
if (legit_path[i] == hijacked_path[j]) {
return true;
}
}
}
// No hops of the paths match
```

```
return false;
}

std::vector ROVppAS::get_as_path(Announcement &announcement) {
// will hold the entire path from origin to this node,
// including itself
std::vector as_path;
uint32_t current_asn = asn; // it's own ASN
AS * curr_as = this;
uint32_t origin_asn = announcement.origin; // the origin AS of the announcement
Announcement curr_announcement = announcement;
// append to end of ASN path
as_path.push_back(current_asn);
// Add to the path vector until we reach the origin
while (current_asn != origin_asn) {
// Check if the received from asn is 0
// This means it's coming from an AS which was seeded with this announcement
// Just take the origin as the ASN in this case
if (curr_announcement.received_from_asn == 0) {
as_path.push_back(curr_announcement.origin);
break;
}
// Get the previous AS in path
current_asn = curr_announcement.received_from_asn;
curr_as = as_graph->get_as_with_asn(current_asn);
// Get same announcement from previous AS
curr_announcement = *(curr_as->get_ann_for_prefix(curr_announcement.prefix));
// append to end of ASN path
as_path.push_back(current_asn);
}
return as_path;
}

std::ostream& ROVppAS::stream_blacklist(std:: ostream &os) {
for (std::pair, Announcement> prefix_ann_pair : blackhole_map){
// rovpp_asn,prefix,hijacked_ann_received_from_asn
os << asn << ",";
prefix_ann_pair.second.to_csv(os);
}
return os;
}
```