Network Working Group
Internet-Draft
Expires: March 4, 2008

Z. Czirkos
G. Hosszu
BME EET

Sept 2007

The Komondor Peer to Peer Security System
draft-czirkos-komondor-p2p-security-01

Status of this Memo

Copyright Notice

Abstract

This memo presents Komondor, an experimental peer-to-peer security system. Komondor is a network intrusion detection system. Hosts running this software organize themselves automatically in an overlay network, where they share information about intrusion attempts they detect. This way the security of all participants is increased, as they are able to prepare for some of the attacks in advance.

The overlay created is a peer-to-peer network, which is completely decentralized. This communication model ensures stability, and by using this the system remains operational over an unstable network.

Table of Contents

1. Introduction

This memo describes Komondor, a peer-to-peer based collaborative security system [2].

Komondor is a host security software, which is able to detect network-sized intrusion attempts. Hosts running this software share information about detected intrusion attempts, for example by exchanging suspicious IP addresses. This way the security of all entities is increased, as they are able to prepare for the attacks in advance. The heterogenity of hosts in such a network also increases security, as different operating system and software versions are vulnerable to different kinds of attacks.

Komondor entities organize themselves into a peer-to-peer network, where every entity is responsible for collecting and analyzing data. This network model has no central server, and therefore it can remain operational on the unstable network. A centralized model would be completely unfeasible for such an application, as an intruder attacking the central server would shut down the intrusion detection network completely.

The Komondor system implements a Distributed Hash Table (DHT). In the overlay network, every Komondor entity (node) is assigned a unique identifier, a nodeID, which is a 32-bit number. The source IP address of every attacker identified is hashed with a hash function similar to SHA-1, which also results in a 32-bit number. Every node stores reports, which have a hash value close to its identifier. This way reports of the same attacker are collected by a single node, so it is able to analyze them to detect network-size attacks.

If an IP address becomes suspicious, the entity collecting and analyzing its data issues a broadcast message in the overlay, alerting every other nodes. This way they can prepare their protection systems in advance, expecting the same attacker.

2. The Structure of the Overlay

2.1. Overview

The Komondor intrusion detection system is built over an overlay network, which is similar to Kademlia [1].

Kademlia is a distributed hash table (DHT), which is using the XOR metric. In the DHT, every node has a unique identifier, which is a 32-bit number for Komondor. Information to be stored must be in a key/value format. Here the key is the IP address of the attacker, and the value is a report of an intrusion attempt detected.

When storing information in the overlay, a node uses a hash function to generate a 32-bit number from the key. (Note that this number is the same size as the node identifiers.) The report is then sent to the node which has its identifier closest to the hashed value. The distance is calculated using the XOR function.

There is no routing defined in the overlay network. It is merely used to let nodes discover each other, e.g. to find out the IP address and port number of the participators. Once the destination of the message is known, the reports of intrusion attempts detected can be sent directly to it.

2.2. The Structure

This section describes the structure of the Komondor overlay network, which is very similar to Kademlia.

When joining the overlay, every node chooses a random 32-bit nodeID, which will essentially be its address in the application level network. The purpose of the overlay is to help nodes find out the Internet address of each other in logarithmically many steps.

Nodes in Kademlia are leaves of a binary tree. Each node's position is determined by the shortest unique prefix in its nodeID. For any given node, the tree is divided into successively smaller subtrees, which do not contain the node. For node 0011, these subtrees are prefixed by the numbers 1, 01, 000 and 0010. These prefixes can be acquired by leaving the first n bits of the address untouched, then inverting the following bit. The tallest common subtree for node 0011 is 0xxx, therefore the neighbouring subtree is 1xxx. The second smallest is 00xx, therefore the neighbouring subtree is 01xx, and so on.

Nodes in the overlay are required to know at least one node in each subtree they belong to. If this is achieved, every node can be

located by knowing its nodeID. Distance between two nodes (or a hashed value and an identifier) is calculated using the XOR function. Note that this effectively describes the binary tree representation of the system presented above, as a larger number will indicate a shortest common prefix of nodeID's.

Every node should maintain a list of other nodes in the overlay it is aware of. For the system to be scalable, these list is handled using the binary tree. Every subtree is assigned a fixed length list of nodes. The members of such a list are IP address, port number, nodeID triplets. Kademlia calls these lists k-buckets, and refers to their maximum size as k [1].

Nodes of a Komondor system can quit unexpectedly, because of having network errors or being under heavy attack. Therefore the system-wide configuration parameter k must be chosen big enough to ensure that not too many nodes leave the network, before all k-buckets of another one become empty (i.e. they contain only dead connection information). For the highest subtrees, which contain the closest nodes for a given node, these lists will be usually empty, as no appropriate nodes exist.

K-buckets are to be propagated by two means. First, peer messages can contain addresses of other nodes in the overlay. Second, when a node receives a message, the source address and port can also be remembered.

2.3. The Lookup Procedure

To find the IP address and port number of a participator with a specific nodeID, a node should initiate a recursive lookup procedure.

Every node manages its own lookup procedures. Such a procedure consists of successive find node messages. It is started with sending a find node message to the closest node to the destination. The distances between the desired nodeID and the known IDs are calculated using the XOR function. As every node has greater knowledge of its surroundings, the reply will contain peer messages with addresses of nodes even closer. Then the initiator can issue another find node message.

A lookup is complete, when the k closest nodes to the destination address have been queried. As the Internet address of the destination is known at this point, more reports about the same attacker can easily be sent at once.

3. Intrusion Detection

This section describes the intrusion detection used by a Komondor overlay.

The detection scenario can be summarized as follows. When a member of the overlay detects a suspicious event, it notes the IP address of the intruder. This IP address is hashed using the hash function used by all nodes. The result is a 32-bit number, which is treated as a nodeID, and looked up in the overlay. The report about the detected event is the sent to the specific node by an attack message.

The same hash function is used by all nodes, so report about the same attacker will be collected by a single node, no matter which nodes the events were detected by. The collector node is then able to summarize and analyze the reports. If the reports indicate an intrusion attempt, the collector initiates a broadcast message among nodes, which is called a protection message. This message contains the IP address of the attacker, against which nodes should take their own protective steps.

Nodes are free to use their own means of intrusion detection and protection. One example for this is to monitor system log files for detection, and to configure a firewall for protection. Mandatory tasks for nodes is to report suspicious events, and to forward broadcast messages, as described by the Komondor protocol.

3.1. Description of an Intrusion Attempt

Suspicious events detected do not necessarily indicate a real intrusion attempt. The common example for this is a mistyped password during an authentication. Every event which is reported should be therefore assigned a number, which denotes its importance.

The parameters in a report are therefore:

IP address: The IP address of the attacker.

Importance: The weight of the event, which is a number between 0 and 100.

Event type: The name of the event. For statistical purposes.

Detector: The host name of the node which detected the event. For statistical purposes.

3.2. Analysis of Reports

Every node is responsible for collecting and analyzing reports received. Each report is to be remembered for one hour. Records older than one hour are to be deleted.

An index should be calculated for every IP address which is suspected to be an attacker. This index is a summation of importance values seen in the reports, which are decreased with time. The importance of one report is:

importance = reported importance * (3600s - report age),

where the age of the report is the time that has elapsed since the report was received by the collector. Reports are expected to be received in a few seconds, so the current protocol does not require nodes to timestamp their reports.

These decreased importance values are then summed up. The index should be recalculated every minute, or when a new report is received for an IP address. If the sum of these increases above 100, the collector node must initiate a broadcast message, signalling other nodes to prepare their protection against the attacker.

The protection message contains the IP address of the attacker. Each protection message is valid for twenty minutes. After that period, if the index for an attacker is still above 100, the collector node should initiate another broadcast.

As the importance values of each reports are summed up, nodes are free to summarize more events in a short time, of the same type and same attacker, in a single report. This reduces the network traffic induced. The short time interval should not exceed ten seconds, however, as that would possibly degrade detection efficiency. Nodes are also allowed to analyze their own reports, and initiate a protection broadcast message, if required. In either case, the reports must always be sent to the collector.

3.3. Report Example

The importance indices for events can be freely chosen, when implementing and configuring a Komondor node. The specific value can be estimated by keeping the the 100-point limit in mind. For example, a failure of a user name and password type authentication can be assigned the importance of 40 points. This will allow users to mistype their password twice, and only the third failure will initiate the protection. The event type for this can be ssh_password_failed, for example.

Events which indicate an attack on their own can be assigned an importance of more than 100 points. For example, an Internet worm attack can have the type slammer_worm, and importance 120.

4. Centralized Logging

The Komondor system supports logging statistical data about detected intrusion attempts.

Reports about the same attacker are always collected by a single node in the system. When the attack seems to be over, that is to say, the last report about the specific attacker times out, the collector node is allowed to send a summary about the attack types and detectors of the event.

This procedure is not a vital part of the workings of the system. The logging works in a centralized way. The node collecting statistical data (apart from the fact that it is a completely functional Komondor node) is assigned the application level network address 0x00000001, instead of the normal randomly choosen address used by other nodes. There can be only one logging node in a Komondor overlay. Nodes in the system use the normal lookup procedure with address 0x00000001 to find out the IP address of the logging node.

The statistical report of the intrusion attempt contains the following data:

IP address: The IP address of the attacker.

Interval: The time interval of the intrusion attempt, from the first to the last event detected.

Event types: The list of all event types detected.

Detectors: The host names of all nodes which detected the specific attacker.

Protection: If the protection was successful for some node.

Most of this information is valuable only to the network administrator who configured the Komondor nodes on his network, as the name of event types, and host names are assigned by him.

The last parameter named "protection" will show if the protection against the attacker was successful for some node. The availability of this information depends on the means of th protection. For example, a firewall can be configured to log dropped packets, and that way the node will know if it is still under attack. This enables network administrators to determine the efficiency of the system.

5. Protocol Specification

The protocol described here is used between two Komondor entities.

5.1. Transport Protocol

Komondor nodes must use UDP for communication. Currently, nodes are free to choose any port number. However, they must use the same port number for incoming and outgoing messages. This allows the k-buckets to be propagated by checking the remote address and port number of messages. For example, if a node receives a message with remote address 192.0.2.92:1705, it can send its reply to this address as destination.

5.2. Message Format

The protocol is based on set of codes which are composed of eight bits (an octet), and uses the ASCII character set. Messages are essentially text lines, which consist of a node identifier, a message identifier, a command and optional parameters.

Each message is prefixed by the nodeID of the sender. This identifier is expressed as a minimum of one, a maximum of eight hexadecimal digits, representing the 32-bit identifier. This is used to assign nodeID's to socket addresses, and to detect if a node is restarted.

The second prefix of the message is also a 32-bit hexadecimal number in ASCII code. This is a random number which should be unique for each different message sent. As UDP does not guarantee message delivery, this number is used to implement a reliable communication channel.

Each message is encapsulated in its own UDP packet (datagram), so there is no delimiter used between them.

The BNF representation for the message is:

```
message  ::= <nodeID> <SPACE> <messageID> <SPACE> <command>
   [parameters]

nodeID  ::= <hex digit> { <hex digit> }

messageID  ::= <hex digit> { <hex digit> }
```

```
command  ::= <letter> { <letter> }

SPACE  ::= ' '

parameters  ::= { <SPACE> <any non-empty sequence of octects, not
   including SPACE> }
```

5.3. Reliable Communication

As UDP does not guarantee reliable message delivery, each message contains a messageID mentioned above. This message identifier is a 32-bit random number. When a node receives a message, it must reply with an acknowledge, unless the messageID is zero. The parameter of the acknowledgement is the messageID of the message which it acknowledges. The own messageID of the acknowledgement must be zero to prevent infinite loops.

Unacknowledged messages should be resent to their destination. If there is no reply from the node, the connection is to be considered dead.

Packet loss of acknowledgement messages can result in message duplicates. For this reason, every node should maintain a list of recently seen messages, to prevent processing it more than once. This list of seen messages should contain the messageID's received in the last minute.

5.4. Hashing Algorithm

Komondor uses a modified version of SHA-1 algorithm, which processes ASCII strings with a maximum length of 64 bytes. The IP addresses of attackers are hashed as strings with the algorithm presented below.

This is a C language implementation of the SHA-1 algorithm used by Komondor. It can be used for little-endian machines. uint32 is assumed to be a 32-bit unsigned integer type.

```
#define S(x,n) (((x) << (n)) | (((x) & 0xFFFFFFFF) >> (32 - (n))))
IDENTIFIER p2p_hash_string (const char *string)
{
    /* rotating left an uint32 */

    uint32 w[80];
    uint32 h0, h1, h2, h3, h4;
    uint32 a, b, c, d, e;
    int i;

    h0 = 0x67452301;
```

```
       h1 = 0xEFCDAB89;
       h2 = 0x98BADCFE;
       h3 = 0x10325476;
       h4 = 0xC3D2E1F0;

       /* copy 64bytes max to w, rest filled with 0 */
       memset (w, sizeof(w), 0);
       strncpy ((char *)w, string, 64);

       /* Extend the sixteen 32-bit words (64 bytes)
          into eighty 32-bit words */
       for (i=16; i<80; i++)
           w[i] = S(w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16],1);

       /* Initialize hash value */
       a = h0;
       b = h1;
       c = h2;
       d = h3;
       e = h4;

       /* Main loop */
       for (i=0; i<80; i++) {
           uint32 temp, f, k;
           if (i <= 19) {
               f = (b & c) | ((~b) & d);
               k = 0x5A827999;
           }
           else if (i >= 20 && i<= 39) {
               f = b ^ c ^ d;
               k = 0x6ED9EBA1;
           }
           else if (i >= 40 && i<= 59) {
               f = (b & c) | (b & d) | (c & d);
               k = 0x8F1BBCDC;
           }
           else { /* i >= 60 */
               f = b ^ c ^ d;
               k = 0xCA62C1D6;
           }

           temp = S(a,5) + f + e + k + w[i];
           e = d;
           d = c;
           c = S(b,30);
           b = a;
           a = temp;
       }
```

```
        /* Add this chunk's hash to result so far */
        h0 = h0 + a;
        h1 = h1 + b;
        h2 = h2 + c;
        h3 = h3 + d;
        h4 = h4 + e;

        /* Komondor uses 32bit identifiers,
           so we xor the five 32-bit  */
        return h0 ^ h1 ^ h2 ^ h3 ^ h4;
    }
```

5.5. Message Details

This section lists all messages which must be or should be implemented by Komondor entities. Every subsection describes a message and its parameters.

The parameters are usually encoded as ASCII strings, and they are fixed in order.

5.5.1. Acknowledgement Message

Command: ack

Parameters: <messageID>

This message should be the first immediate answer to any message received. The purpose of this is to implement reliable communication over UDP. The parameter should contain the messageID of the message received, which is acknowledged by this line.

One exception for sending acknowledgements is when the messageID received is zero (0). Also, the messageID of the acknowledgement message should be zero to prevent infinite loops.

Example: 78ae56cd 0 ack 12983efa

Node 78ae56cd acknowledges message 12983efa.

5.5.2. Ping Message

Command: ping

Parameters: none

This message tests if a connection is alive. The reply is the acknowledgement message described in the section above. Note that

every message with a messageID other than zero generater an acknowledgement, not only the ping command.

5.5.3. Quit Message

Command: quit

Parameters: none

This message informs its destination node, that the source node is intending to leave the overlay. The receiver should delete the connection from its k-buckets.

5.5.4. Findnode Message

Command: findnode

Parameters: identifier

This message requests the destination to send the list of other nodes which it is aware of. The destination should reply with a peer message, containing connection information to other nodes. Those nodes must be closest ones to the identifier in XOR space, which the destination is aware of. The list is essentially a k-bucket, or information about nodes from several k-buckets, if the bucket in question did not contain at least k nodes [1].

Example: 78ae56cd 39fa9912 findnode 98be1f7d

Node 78ae56cd queries one of its neighbors about other nodes, which have identifiers close to 98be1f7d. The neighbor will answer this request with a peer message, containing k-bucket.

5.5.5. Peer Message

Command: peer

Parameters: <ipaddress:port> { ipaddress:port }

This message contains the Internet address of one or more nodes. It is a reply to a findnode message. The IP address is to be expressed in dot notation as decimal numbers (for example 192.0.2.4). The port number must also be a decimal number.

Example: 78ae56cd 2795aef5 peer 192.0.2.4:1232 192.0.2.9:1892

Node 78ae56cd informs the receiver of the message about other nodes, reachable at 192.0.2.4:1232 and 192.0.2.9:1892.

5.5.6. Attack Message

Command: attack

Parameters: <ip> <importance> <hostname> <rulename{,rulename}> <protected>

This message is a report of an intrusion attempt detected. The parameters have their meanings as follows:

ip: first one, most important is the IP address of the attacker in dot notation, 192.0.2.107 for example.

importance: the second parameter represents the importance of the event, and is a decimal number.

hostname: The third is the name of the host which detected the attack (a string of any printable characters except space).

rulename: This is the name of the attack type (also printable, non-whitespace characters).

protected: This should be 1, if the protection is already active against this attacker, 0 otherwise.

The third, fourth and fifth parameters serve statistical purposes only.

The receiver of this message is a node selected by the DHT to collect information about the specific attacker. It should sum up the indexes of each event reported, and if the total is greater than 100, initiate a broadcast procedure to inform entities in the overlay about an undergoing attack.

The last parameter allows the collector node to estimate the effectiveness of the system.

Komondor entities are allowed to send information about more events in a single message. In this case, the importance indexes must be summed up. If there are different rule names, they must be separated by the comma (,) character. As the importance of each detection decreases with time, multiple events are only allowed to be sent in one message, if the interval between their detection is not longer than 10 seconds.

Example: 78ae56cd 9efc56e1 attack 192.0.2.107 40 jutas ssh_invalid_password 0

Node 78ae56cd with host name jutas detected an intrusion attempt from 192.0.2.107. The event detected was an invalid password in an ssh session, and the severity of it is 40 points. Protection against the attacker at 192.0.2.107 was inactive at the time of the detection. Note that detection can be still possible with an active protection, for example by checking firewall log files.

5.5.7. Protection Message

Command: protect

Parameters: <ip address>

When a suspicious IP address reaches the 100 point limit, it is considered an attacker. The Komondor entity collecting the reports belonging to the IP address must initiate a broadcast message in the overlay, informing other entities about the possibility of further intrusion attempts. The broadcast is initiated by sending the protection message to all its known neighbours.

Every node receiving a protection message must check if it has recently (in one minute) received a protection message regarding the same attacker. If not, it must forward the message to all its known neighbours.

The node collecting reports and initiating the broadcast message must also check if the points of the attacker did not decrease below the limit of 100 points in ten minutes. If this is the case, the broadcast must be resent. Therefore the nodes of the overlay are periodically informed about possible attackers, in every ten minutes.

Example: 78ae56cd 57ab9ef1 protect 192.0.2.107

Node 78ae56cd collected enough reports about events from 192.0.2.107, and it is now considered an attacker.

5.5.8. Announce Message

Command: announce

Parameters: <nodeID> <hostname> { attacker,index }

Implementing this message is optional, but recommended. The purpose of this is to enable the central logging entity to monitor the state of the overlay, and the information that is stored in the hash table. Each entity should send an announce message to the logging server periodically. The first parameter of the message is its nodeID as hexadecimal digits, and the second one is its name (a string of

printable, non-whitespace characters).

Example: 78ae56cd 1731eacb announce 78ae56cd buda 192.0.2.107,45

5.5.9. Statistics Message

Command: stat

Parameters: <first detection> <last detection> <attacker> <detectors> <rules> <max susp> <success>

This message is sent from a data collector entity to the central logging server, when the reports of the attack become outdated. It is not mandatory, but recommended behaviour, as the statistics provide useful information about attackers and the usefulness of the entire overlay.

Parameters of this message are:

First detection: the time of the first detection in Unix time format (seconds since 1970-01-01 00:00:00 UTC), expressed as a decimal number.

Last detection: the time of the last event detected, in Unix time format.

Attacker: the IP address of the attacker, in dot notation (192.0.2.107).

Detectors: a comma separated list of names of hosts which detected this attacker.

Rules: a comma separated list of types of attacks, which were detected from this attacker.

Importance: the maximum event importance index which was calculated.

Success: a 0 or 1 value indicating if the protection was successful, that is, an event was detected when the protection was already active. For example, the node detected further activity from the attacker by examining the firewall log files.

Example: 78ae56cd 1731eacb stat 1183394505 1183475323 192.0.2.107 nemere,buda sshd_invalid_password,sshd_unknown_user 290 1 Node 78ae56cd sends statistics about attacker at 192.0.2.107. The first event was detected at 1183394505, and the last one at 1183475323. Nodes with host name nemere and buda were the ones who detected this attacker, and the types of the events were invalid passwords and

unknown user names sent to the ssh service. The maximum importance points of the attack at the full interval was 290. The protection was successful against the attacker for at least one of the detector hosts: it was already aware of this attacker when it detected some activity from it.

6. Configuration Parameters of the Komondor System

6.1. Replication Factor

The replication factor is defined in Section 2.2, and is referred to as k, the size of k-buckets in the Kademlia overlay.

It has two important effects on the workings of the overlay. As the size of the k-buckets, it determines the number of nodes a specific node is aware of, in every subtree of the overlay. The number must be high enough to maintain the stability of the overlay, as nodes under attack can even disappear from the network.

It also determines, who many nodes an attack message is to be sent to, that is to say, reports of intrusion attempts detected are replicated at different nodes of the overlay. Increasing replication increases network traffic, but makes the overlay and detection more reliable. A recommended value is between 3 and 5 for smaller networks. The original Kademlia paper chooses 20, which should be enough for an overlay with tens of thousands of nodes [1].

6.2. Intrusion Detection and Protection

Currently, Komondor implementations are encouraged to use their own ways of intrusion detection and protection. This is unlikely to change in the future, as it would limit the portability of the system, and collaboration of Komondor nodes running on different platforms.

This is why the names of attack types in the attack messages serve informational and statistical purposes only (see Section 5.5.9 and Section 5.5.6.) The parameters having direct effect on the workings of an overlay are the event importance indexes in messages. As different platforms have different vulnerabilities, this is not operating system independent, either. Implementations are currently advised to set the importance indexes assigned to different events reasonably. A suspicious IP address collecting 100 points is considered an attacker by Komondor entities. Therefore, a failed login should be assigned 40 points for example, allowing for two mistyped passwords before the protection mechanisms would become active at the third failed attempt.

6.3. Timing of Intrusion Detection

Reports of intrusion attempts detected are sent to Komondor entities chosen using their nodeID's. Received reports are collected, and the importance is calculated. Importance points should decrease linearly, according to the formula described in Section 3.2. The

time the importance of each report decreases to zero is also a configuration parameter of the detection. Increasing this interval increases the sensitivity of intrusion detection, but also increases the possibility of false alarms.

6.4. Protection Methods

Possible methods for protection are platform-dependent. The current protocol specification enables Komondor implementations to exchange information about IP addresses belonging to possible attackers. One possible way of protection against them is using the firewall of the host running the Komondor entity. However, implementations are free to use any other method.

Suspicious IP addresses, which collect 100 points, are treated as attackers. If a Komondor entity collected enough intrusion attempt reports about an attacker, and the sum of the indexes reach the 100 point limit, the entity is required to broadcast a protection message using the overlay, as described in Section 5.5.7. This message should be resent every 10 minutes, until the collected points drop below the limit.

6.5. Overview of Configuration Parameters

The following table gives an overview of configuration parameters used in a Komondor overlay. It also shows their recommended values.

Replication factor (k)	3
Protection limit	100
Protection message resend	10 minutes
Attack report lifetime	20 minutes
Interval to remember seen messageID's	1 minute
Interval between detected events which can be sent in one message	10 seconds

The first four parameters directly affect the behavior of the overlay and the intrusion detection. The recommended practice is to have these parameters be adjustable through a configuration file.

7. Security Considerations

The current version of the Komondor protocol does not describe authentication among nodes.

A misbehaving Komondor node does not cause a possibility of information leak for the hosts of the overlay. It may, however, present false attack reports. This might cause denial of service to the IP addresses which are reported as attackers.

Another harm a misbehaving Komondor node can do is that is simply ignores attack reports it receives, thereby not alerting other nodes about possible dangers. This problem can be solved by using replication.

8. Implementation Details

This section contains useful information which can be used when implementing the Komondor system.

8.1. Communication over UDP

The current implementation of the Komondor system identifies nodes by their IP address:port numbers and nodeID's. Nodes are required to use the same port number with their outgoing messages, as the port number they are receiving UDP packets at. That is to say, a node which is reachable at UDP address 192.0.2.99:2001, must send its messages with source address 192.0.2.99:2001. This can easily be implemented by using the same socket and file descriptor for sending and receiving packets, on any operating system using the BSD sockets interface.

8.2. Kademlia Binary Tree

As described in Section 2.2, the Kademlia overlay organizes nodes to a binary tree. This is not a real topology, but rather a reasonable way for each node to store information about other nodes it is aware of.

Every node is required to have detailed knowledge about its surroundings in the DHT address space. More specifically, every node maintains a list of other nodes for every successively smaller subtree of the overlay, which it does not reside in. The size of these lists, the k-buckets, is fixed, and it is referred to as k. As it is described in the original paper [1], the most reasonable way to implement this is using a binary tree of lists for storing connection data. The program starts off with one single list, and as it gains knowledge about other nodes, the single list is split into multiple lists, arranged in the binary tree.

Kademlia advises the k-buckets to be sorted by the time the nodes were last seen. This is unnecessary for Komondor, when implementing an intrusion detection system over the DHT.

8.3. Usage of the Overlay

The Komondor overlay is best used on a local area network, as it is unlikely for the same intruder to attack hosts at different locations, due to the big number of Internet addresses.

9. References

[1] Maymounkov, P. and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric", <http://citeseer.ist.psu.edu/maymounkov02kademlia.html>.

[2] Czirkos, Z., "Komondor Security System - the home page of a sample implementation with dynamic monitoring of the overlay", <http://jutas.eet.bme.hu>.

[3] Czirkos, Z., Hosszu, G., and F. Kovacs, "E-Collaboration Enhanced Host Security", Encyclopedia of E-Collaboration, edited by Ned Kock, Information Science Reference, Hershey, USA, 2007, ISBN: 978-1-59904-000-4.

[4] Czirkos, Z. and G. Hosszu, "Peer-to-Peer Methods for Operating System Security", Encyclopedia of Networked and Virtual Organizations, edited by Goran D. Putnik and Maria Manuela Cunha, Information Science Reference, Hershey, USA, 2007, ISBN: 978-1-59904-885-7.

Authors' Addresses

Zoltan Czirkos
BME EET
Goldmann Gy. ter 3.
Budapest H-1111
Hungary

Phone: +36 1 463 4034
Email: czirkos@nimrud.eet.bme.hu

Gabor Hosszu
BME EET
Goldmann Gy. ter 3.
Budapest H-1111
Hungary

Phone: +36 1 463 2724
Email: hosszu@nimrud.eet.bme.hu

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).