CHAPTER-6
RELIABILITY BASED SECURITY

6.1 INTRODUCTION

Migrating crawling agents (migrants) based methods minimize network utilization and also keep update of documents. The process of selection and filtration of web documents can be done at web servers rather than search engine side which reduces network load caused by the web crawlers. When very large volumes of data are stored at remote hosts on web, it is good to process the data in its locality rather than transferring it over the network. The remote platform provides the computational environment in which an agent operates. The platform from which an agent originates is referred to as the home platform, and normally is the most trusted environment for an agent. The agent approach uses the bandwidth of the network to migrate agent to a remote platform, and allows it to continue to run after leaving a node even if it loses connection with the node where it was created thereby providing the better utilisation on communication and allows parallel distributed applications.

Since the migrant’s code from the search engine side is transferred and executed on web servers, an environment controlled by another party, gives rise to several security issues in mobile agent computing. Such issues include authentication, authorization (or access control), intrusion detection etc. Security issues are becoming more significant in the case of large heterogeneous distributed web. This chapter presents a reliability based approach to maintain safety and security of the migrants as well as environment in which they execute.

6.2 RELIABILITY BASED APPROACH

A migrant can interact with the documents much faster than from across the network by migrating to the web server i.e. location of web documents to be processed. This also reduces network traffic. The distributed crawling with migrant approach uses a crawler manager at the search engine side that deputes migrant to the web servers with a list of URLs of respective web server(s). The migrant, on reaching a web server; crawls the pages, selects the required pages for its collection and comes back
to the search engine with the collection. It extracts all links embedded in the pages, adds them to the list of URLs to be crawled, and keeps on crawling all until all URLs in the list are not crawled. It reduces unnecessary overhead of bringing all the pages to the search engine side. The size of the collection can further be reduced by filtering the required specialized web pages and even compressing them. This approach performs the job of selecting the information at the place where information lies.

Since migrants roam across the web world and need to execute on a remote platform, it requires the agent designers to work on security related issues as discussed earlier. It is essential to maintain security and integrity of migrants and the remote platforms both. Migrants need to be protected from the attacks of other migrants/agents and remote platforms as well as the migrants should not behave maliciously and attack on remote platforms. The proposed approach adjusts the security of migrants by examining the security requirements by computing its reliability factor.

In the proposed approach it is assumed that when a migrant reaches a remote platform first time, it is considered least reliable \((R_{\text{new}}=.5)\) and given a restricted secured environment to execute with minimum possible resources. In case a migrant reaches a remote platform \(n^{th}\) time, its reliability \((R_{\text{old}})\) is checked. If it is found that its reliability is above a threshold value (say .5), it is given environment with requested resources in incremental order to run. If it remains reliable during its stay at remote platform, its reliability \(R_{\text{old}}\) be increased by \(\Delta R\) else if it remains unreliable, \(R_{\text{old}}\) be decreased by \(\Delta R\).

The reliability factor of a migrant can be computed by the following formula,

\[
R_{\text{new}} = R_{\text{old}} + \Delta R \quad \text{.....(1)}
\]

where,

\[R_{\text{old}}\] : old reliability factor for a migrant

\[R_{\text{new}}\] : new reliability factor for a migrant

\[\Delta R\] : change to be made in reliability factor
The proposed architecture of migrants based crawling system is shown in Figure 6.1.

It is proposed to manage reliability with restriction. When migrant is not reliable the restriction is maximum (reliability=0, restriction=max), and when the agent is reliable the restriction is minimum (reliability=max, restriction=0). So as the reliability increases, the restriction of the environment decreases, as shown in Figure 6.2.
Various parameters which are considered to check the reliability of migrants are given in Table 6.1.

Table 6.1: Parameters to check reliability

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcrag_owner</td>
<td>owner of the migrating crawling agent</td>
</tr>
<tr>
<td>mcrag_issuer</td>
<td>issuer of the migrating crawling agent</td>
</tr>
<tr>
<td>mcrag_size</td>
<td>size of migrating crawling agent</td>
</tr>
<tr>
<td>mcrag_ver</td>
<td>version of migrating crawling agent</td>
</tr>
<tr>
<td>mcrag_data</td>
<td>data information of migrating crawling agent</td>
</tr>
<tr>
<td>mcrag_attrib</td>
<td>attributes of migrating crawling agent</td>
</tr>
<tr>
<td>mcrag_state</td>
<td>state information of migrating crawling agent</td>
</tr>
<tr>
<td>mcrag_req_resources</td>
<td>amount of resources required by migrating agent</td>
</tr>
</tbody>
</table>
When a migrant reaches at remote platform first time, the host platform acquires the above information from the migrant and needs to store for future reference. When a migrant reaches a platform next time, its parameters values be matched with values extracted last time and ΔR is computed based on weights as specified above. Now \( R_{new} \) can be computed by adding/subtracting \( \Delta R \) into \( R_{old} \).

On reaching a web server, various permissions will be allotted to a migrant in incremental manner:­

\[
\begin{align*}
\text{Level-I} & \quad \{ \\
\text{Arrival of crawling agent} & \\
\text{Agent information extraction} & \\
\text{Agent identification and authorization} & \\
\text{Level-II} & \quad \text{Memory access} \\
\text{Level-III} & \quad \text{CPU access} \\
\text{Level-IV} & \quad \text{Storage access} \\
\text{Level-V} & \quad \{ \\
\text{Crawling and processing of pages} & \\
\text{Agent with data collection is allowed to leave} & \\
\end{align*}
\]

Depending upon the value of \( R_{old} \), permissions of a specific level (Level-I to Level-V) are given to a migrant to start/continue execution. The migrant may also be transferred from higher level to lower level to revoke some of the permissions, if \( \Delta R \) is found negative. The migrant may also be transferred from lower level to higher level to grant few more permissions, if \( \Delta R \) is found positive. If \( R_{new} \) of a migrant is below .5, then all the data contained in the migrant is erased and the migrant is terminated.

The complete flow of execution of the proposed architecture is shown in Figure 6.3.
Figure 6.3: Flow of execution of a migrant on remote platform
Algorithm for the proposed architecture is given in Figure 6.4.

```
Begin

select url_q from all_urls (subset of URLs related to a specific web server)

migrant is dispatched to web server

begin

mcrag_agent reaches a specific web server

check for reliability

for all url ∈ url_q

begin

check permissions from robot.txt

access local webpage

analyse webpage

extract keywords from webpage

if relevant

compress webpage

store compressed webpage in the pages_coll

extract new_links from webpage and add in url_q

end

for all new_link ∈ webpage

begin

if new_link is local

add new_link to url_q

else

add new_link to external_url_list

end

end

all_urls = all_urls U url_q U external_url_list

End
```

**Figure 6.4:** Algorithm for the Proposed Architecture
Migrant with \textit{url\_q, external\_url\_list} and \textit{pages\_coll} returns back to the search engine side.

Algorithm for checking reliability and allowing execution with permissions of different levels is given in Figure 6.5.

\begin{center}
\begin{minipage}{0.8\textwidth}
\begin{algorithmic}
\State Begin
\State select \textit{url\_q} (subset of Urls related to a specific web server)
\State migrant is dispatched to web server
\State \textit{mcrag\_agent} arrives at specific web server
\If {\textit{mcrag\_agent} arrives first time}
\State Begin
\State set $R_{new} = .5$
\State $R_{old} = R_{new}$
\State migrant is provided a secured restricted environment with Lavel-I permissions
\If {remains reliable}
\State reliability of \textit{mcrag\_agent} $R_{old}$ is increased by $\Delta R$
\EndIf
\If {does not remains unreliable}
\State reliability of \textit{mcrag\_agent} $R_{old}$ is decreased by $\Delta R$
\EndIf
\EndIf
\If {\textit{mcrag\_agent} arrived n$^{th}$ time}
\State Compute $\Delta R$ (i.e. change in reliability)
\EndIf
\end{algorithmic}
\end{minipage}
\end{center}
$R_{\text{new}} = R_{\text{old}} \pm \Delta R$

while migrant is willing to continue or $R_{\text{new}} \geq .5$

if $R_{\text{new}} < .5$

discard any requests from \textit{mcrag\_agent}

if $R_{\text{new}} \geq .5 && < .6$

migrant is allowed to execute with Level-I permissions

if $R_{\text{new}} \geq .6 && < .7$

migrant is allowed to execute with Level-II permissions

if $R_{\text{new}} \geq .7 && < .8$

migrant is allowed to execute with Level-III permissions

if $R_{\text{new}} \geq .8 && < .9$

migrant is allowed to execute with Level-IV permissions

if $R_{\text{new}} \geq .9$

migrant is allowed to execute with Level-V permissions

while \textit{crag\_agent} remains reliable

allowed to continue

Compute $\Delta R$ (i.e. change in reliability)

if \textit{crag\_agent} remains reliable

reliability of \textit{mcrag\_agent} $R_{\text{new}}$ is increased by $\Delta R$

Contd…
if crag_agent remains unreliable

reliability of mcrag_agent $R_{new}$ is decreased by $\Delta R$

i.e. $R_{new} = R_{new} \pm \Delta R$

End of while

End

**Figure 6.5:** Algorithm for Checking Reliability and allowing permission of Execution

The following examples show dynamic computation of Reliability of a migrant and allotment of permissions by taking different datasets:

**Case 1:** when the agent arrives at web server 1\textsuperscript{st} time.

Consider the data given below:

$$R_{new} = R_{old} = .5$$

**Agent is allowed to execute with Level-I permissions.**

**Case 2:** when the agent arrives at web server n\textsuperscript{th} time.

$$R_{old} = .5$$

On checking various parameters of migrant, mcrag_owner, mcrag_issuer, mrcag_size and mrcag_path_history are found same,

so, $\Delta R = .01 + .01 + .01 + .02 = .05$

$$R_{new} = R_{old} + .05 = .55$$

**Agent is allowed to execute with Level-I permissions.**
Case 3: when the agent arrives at web server n\textsuperscript{th} time.

\[ R_{\text{old}} = .55 \]

On checking various parameters of migrant, mcrag\_owner, mcrag\_issuer, mrcag\_size and mrcag\_path\_history are found same,

so, \(\Delta R = .01 + .01 + .01 + .02 = .05\)

\[ R_{\text{new}} = R_{\text{old}} + .05 = .60 \]

Agent is allowed to execute with Level-II permissions.

On computing the reliability dynamically, during its execution, it is found that excessive usage of CPU is observed,

So, \(\Delta R = -.2\)

\[ R_{\text{new}} = R_{\text{old}} - .2 = .60 - .2 = .40 \]

Reliability decreased below threshold value, Agent is terminated.

Case 4: when the agent arrives at web server n\textsuperscript{th} time.

\[ R_{\text{old}} = .40 \]

\[ R_{\text{new}} = R_{\text{old}} = .40 \]

Agent is terminated.

The proposed reliability based approach resolve the issue of security based by dynamic computation of reliability factor. It proposes to provide a restricted secured environment to the migrating agent and its restriction be decreased (or increased) in incremental manner as reliability of the agent increases (or decreases).

The next chapter concludes the outcome of the work proposed in the thesis. Future possibilities of the research work based on the proposed design are also suggested.