Automated Peer-to-Peer Security-Update Propagation Network

ZAKIYA M. TAMIMI

Faculty of Information Technology
Arab American University-Jenin
Jenin, West Bank, P.O. 240
PALESTINE
ztamimi@kent.edu http://www.cs.kent.edu/~ztamimi

Abstract: - In this project we design and implement a security-update propagation network that employs peer-to-peer model, where computers act as equals and all hosts participate in providing security update services. When a computer node joins the network to download new security updates it is automatically enlisted to serve other computers with older updates. Hence, as network size becomes larger its capability to provide service will increase and performance balances very well. To protect the network against malicious computers that may join in to spread infected files, we employ digital signature to verify the validity of any download and identify infected nodes. In addition, we incorporate other features in our network such as: the support for multiple platforms, automation, new update notification, and software update.

Key-words: - Peer-to-peer, System patch, Security flaw, Platform, Internet worm, Digital signature, Multithreading.

1 Introduction

In recent years Internet users and services have faced unprecedented volume of security outbreaks including: very fast worms, Trojans, and Distributed Denial of Service (DDoS) attacks. Cyber criminals spread their malicious codes that will transform infected machines into zombies that can be used to launch further attacks. Internet worms that penetrate into a machine through some security flaw can spread at large scale in super spreads harming not only infected machines but also network traffic. Such security threats are facilitated by the massive size and high-connectivity of the Internet. One logical way to counter massive scale attacks is by securing end hosts. Some software suppliers provide system patches that can fix some software security flaw or can detect and remove malicious codes. The dissemination of these system patches (or security updates) is implemented using fixed number download centers, which cannot measure up in terms scale or performance with current security threats.

Some research work [3, 5, 6, 8] has proposed making worm-like mobile codes that will combat malicious worms at their rate. Such solutions however pose legal issues as well as possible side effects [1, 2, 4, 8]. Another solution would be to exploit the collective power of the many end-user machines in detecting worms and spreading patches [7, 9, 10]. By considering the fact that Internet worms make use of end-users computers as engines to spread their infections, it makes sense that employing the same end-users machines to detect and clean worm infection will create an equal “force” to counter worm spread.

There are quite few examples of distributed computing systems that employ end-users machines to achieve high computing power, massive storage, and connectivity. The most popular model for such collaboration between nodes is peer-to-peer systems, which have been employed in file sharing and media streaming, e.g. [9] and [10]. Contrary to client-server model, in peer-to-peer (or simply p2p) model there is no specialized client or server node. Instead all nodes are equally peers. Since any node that joins the network of peers can provide service as well as receive a service, p2p systems are naturally balanced and scale in performance.
IT was proven mathematically in [7] that a p2p system of computers can disseminate security patches effectively which a fixed number of patch server cannot attain [7]. The goal of this research is to design and implement security-update propagation software

that will disseminate security updates among end-host in peer-to-peer fashion. An end-user computer that joins the network to download new security updates will be thereafter enlisted to serve other computers with older updates. Thus, performance of the system scales and balances naturally.

Next sub-section briefly describes related research work in the same line as this. In section 2 key design objectives of the system are listed. A definitions of terms used throughout this paper are in section 3. Discussion of the system protocols, data structure, and functions described in section 4. In section 5, implementation techniques are briefly discussed. In section 6 present issues concern relate to the deployment of the system. Finally we conclude in section 7.

1.1 Related Work

Shakkottai et al have studied in [7] two methods for patch distribution to fight autonomous worms. They prove using fluid model that using P2P system is far more effective than traditional patch centers. Earlier Vojnovi´c et al proposed a hierarchical system of network computers arranged in subnets, each with a "superhost". Patches are spread to "superhosts" through an overlay network and then distributed to end hosts within each subnet [12].

2 Design Objectives

In this section we discuss key design objectives that guide our design.

- The system should provide file sharing of security update files, such that many replicas of the file can be searched for and downloaded.
- In addition, the system should be automated, that is to say, the decision of which updates are needed, where to locate them, and how to install them should require no human interaction.
- As some peer-to-peer file sharing systems have enabled the spread of worms though shared files, we make it our objective to prevent malicious nodes or nodes with virus infections from spreading their infection.
- The system should also be platform independent. In other words, the resulting network should be able to provide together system updates for various platforms.
- The nodes of the system should be notified promptly of new updates as they are made available by originator.
- Since security needs tend to change in short time, we want our software to be updatable "on the go". That is to say, without installing a new version of the whole thing and of course without human interaction.

Some of the system's objectives include scalability and performance-balance is inherited characteristics of peer-to-peer system. As more nodes join the system the demand for service will increase and so will the service capacity of the network; since every new node can provide it as well as receive it. Peer-to-peer system self organized and nodes will change their roles as provider or recipient of service automatically and thus require low administration. Most important of all it is possible to establish a huge service network with super capacity as almost zero-cost by aggregating end-hosts resources such storage, bandwidth, and processing power.

3 Definitions and assumptions

In our system there are two types of entities: index servers and peer nodes. A peer node can be in a server mode or client mode. Both modes are actually running simultaneously. We refer to a peer node running in client mode as client and a peer node running in server mode as server.

Update files are executables that when downloaded and installed can do one or more of the following functions: diagnostic, fixing, malicious code detection and removal. The diagnostic function is to test for existing security flaw, or collect information about current security status of a machine. Once a flaw is detected it can be fixed (or patched), e.g. close an open port. Update file can also detect for the existence of some known worm or Trojan and can remove them from the system. A single security update may require multiple steps of download, install, and restart. More important one peer node may download different security downloads from different other peer nodes.

Software update is another type downloads that are used to upgrade the peer node software. Such upgrades can be to fix a flaw in the code or to enhance performance or extend functionality.

Both Update files and software updates have version numbers that are incremental. The version number is the release date and time for the file. Example, if the release date and time is: 16/4/2007 at 17:55 then the version number would be v200704161755. Any peer node is said to have version number that is of the highest version installed. For any platform the most recent download is the one with highest version number, and any peer node that has the maximum version number installed is said to be up-to-date. If two nodes have different version numbers, the goal of any peer node is to become up-to-date by downloading and installing all downloads up to the most recent download.

A peer node is called malicious peer if it attempts to upload a modified copy of some original download.

Every list of servers have an Expiration date which is the maximum version in the database table, its used to flush the list only if the version number of the client equals the maximum version number.

4 Design

The behavior of a peer node and index server can be described as shown in figure 1 and figure 2, respectively. When the peer node joins the network it will inquire about potential server nodes from the index server. If the node however is up-to-date (has a version number equal to the maximum version number at the index server) it will enlist itself as server at the index server and can provide its services to other nodes.

If however the peer node is not up-to-date, the index server will provide the client node with a list of servers from which it can download necessary updates. The client will connect to one of the servers and inquire about the list of needed downloads. Once the list is received from a server the client will process the list by downloading and installing each file unit the list is empty. To insure that originality of the file and thus prevent the spread of infected files each file is digitally signed by its originator and the client will verify the signature before installing the file. If the signature cannot be verified the client will report detection of malicious node to the index server. Once the client is up-to-date it will be enlisted as a server by index server. It will remain in that state until it is notified of a new update by the index.

4.1 Server Lookup

Index server keeps an index table that contains records of all peer nodes in the system. Index table format is shown below:
{(IP, Platform, Region, Version#, Node-type, Expiration-date, Last connection time)$_k$: k=0..n}.

The region field is the country of the node, which can be determined given the IP address by using some web service. Node type can be either "C", stands for client, or "S", which stands for server. Expiration date is important in order to remove records that are of no use. The expiration date is updated each time a node connects to index server to be the time of last connection plus some constant. When expiration date is reached the expired record will be deleted from index table. A client node will connect to index server in order to request a list of servers. This list contains potential servers that can server a specific client. The format of the list of servers is as follows: {(server IP, server ver#, server region)$_k$: k=0..n}

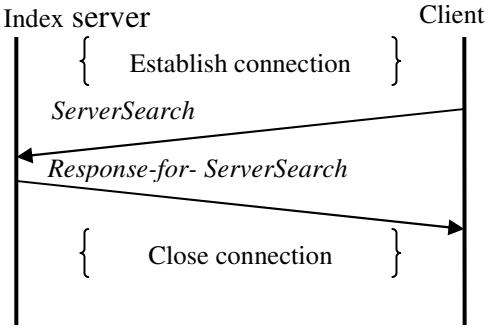

Fig 3: looking up potential servers at the index server

Figure 3 shows the message exchange between a peer node and an index server. The client will send *ServerSearch* message to the index server, which has the following format:
"ServerSearch"| IP address| Platform| Version number

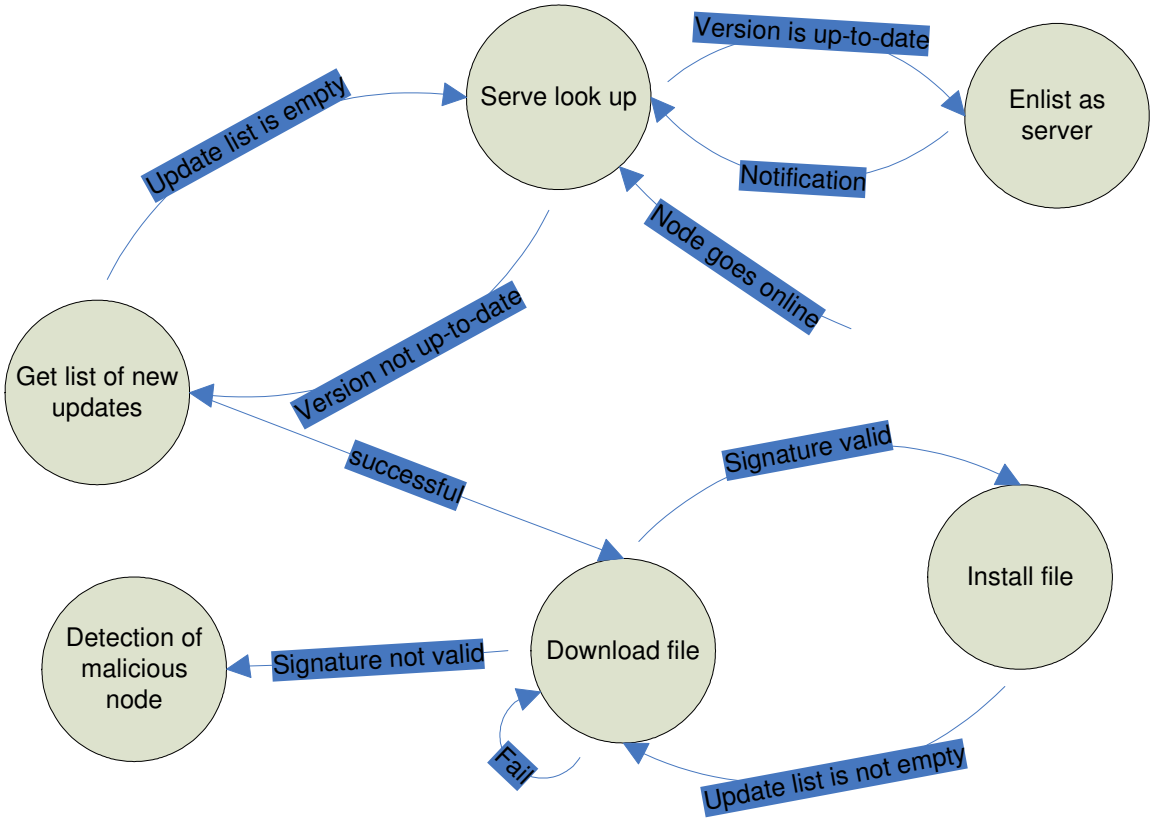

Fig 1. Peer node state diagram

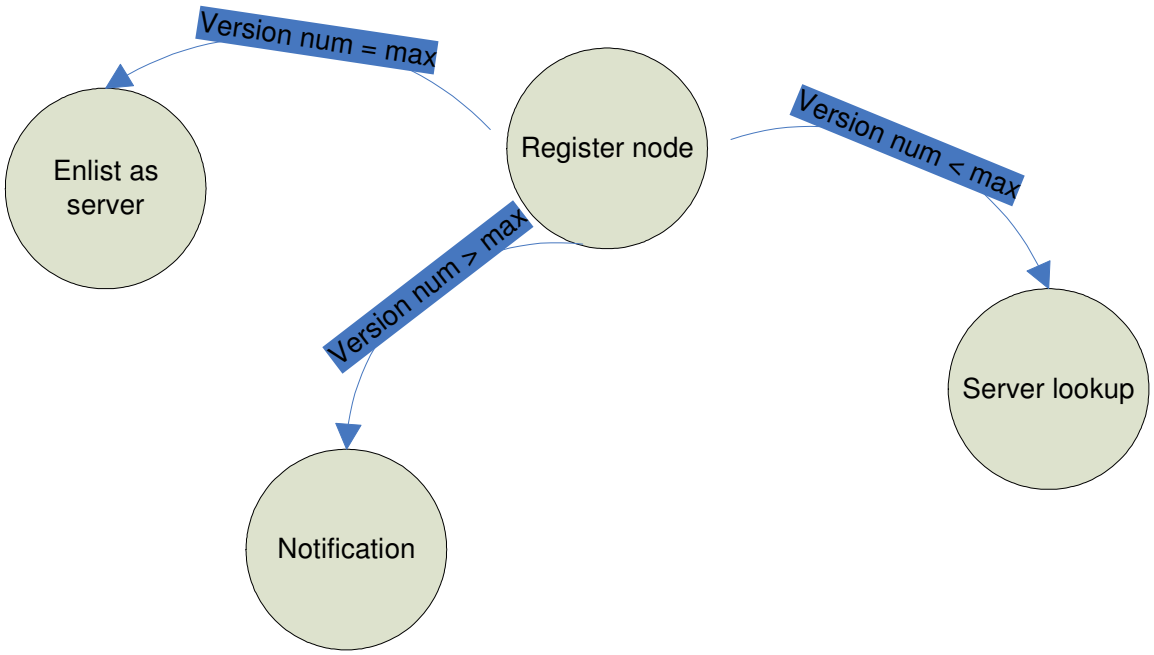

Fig 2. Index server state diagram

The index server will check the client's version number and if it equals to the maximum version number in the index table, it return a response message that contains response code "up-to-date" and will enlist the peer node as server. If the index server finds no potential servers of the same platform then it will return a response message with "error" response code. The index server search for the server IPs that is the preference same region for the client and the version number is higher than the client number. If the result of the search is less than some constant number (n) then the index server will search for the non-regional server that have the same platform and higher version number, then add the result to the list to fill up the list with (n) IP addresses. Then it will return *Response-for-Server-Search* message*, which* has the following format:
"Success"| List of servers

Client will store the list of server sorted according version number in descending order and then will try to establish successful connection with a server starting with the most up-to-date. The client caches the IP list to reuse it as long as its version number is less than the highest version number in the list and will flush this list when the local version number is equal for the highest version number in the server list. The client will try when possible to connect to a server of the same region as the client's. This will make faster requests and responses with that server faster since that server is closer.

4.2 Get List of New Updates

List of updates is a list of all file names that are necessary to be downloaded and installed by a node to become up-to-date. The format of the list is as follows: {(filename, ver#, flag)k, k=0..n}. File name may also its version number, e.g. "Blaster2.0-v20051204020.exe". Flags will direct the client on how to install the downloaded file, which is discussed in subsection 4.7. Figure 4 shows the messages exchanged between a peer node and a server peer node in order to get the list of updates. Initially the client sends *request-list-of-updates* message with the following format:
"RequestUpdates"| Client version number

In response the server node will search its update list for all file that have higher version number than client's. It will then generate a working list of update that contains only the files that are needed by the client and send *response-for-list-of-updates-request* message with the following format:
"Success"| Working list of updates

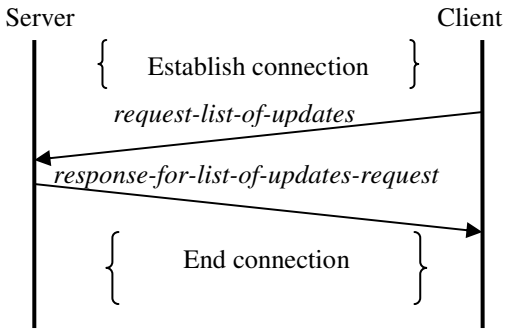

Fig 4: get list of new update

If the server cannot generate such working list it will return a message with "error" response code. The client will store the working list and process it starting with lowest version number element until the list is empty. For each element it will download the corresponding file, remove element from working list and add it to its local list of updates.

4.3 Download File

To download a file the client will connect to some server, which has version number higher than the required file version number. It will then send *request-file-download* message with the file name. If the server doesn't have the file it will respond with an "error" response message. If the file is available, the server will send *response-for-file-download-request* message with the following format:
"Success"| Binary of the file

This interaction is described in figure 5. Each download file weather diagnostic, fix, or software update should contain a digital signature in order to verify it originality and hence prevent the spread of infected file. A digital signature is a hash on the file that is encrypted using the private key of the file originator. To validate the originality of the file, the client will should first decrypted the digital signature using the public key of the originator. If the signature is valid the client will cache the original copy (with digital signature) and then will install the file. On the other hand, if the signature cannot be verified, the client will move into detection of malicious node state, which is discussed in section 4.5.

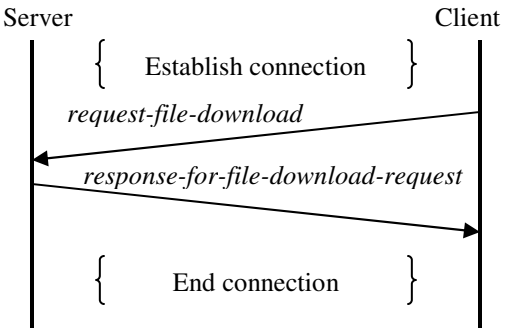

Fig 5: request and response messages to download a file.

4.4 Install File

After the client has downloaded and verified the digital signature of the received file, then it will install the file according to the flags that are given in the corresponding element in the working update list. For example if the flag is "fix" then the client will run this file which will fix will do action like close some ports, delete or modify files, stop some service, etc. if the flag is "diagnostic" the client should run this file which will diagnose for some specific condition and if the condition is found as result of this diagnostic the client will install the next file in the working update list, which is the corresponding fix of this condition. Otherwise the client can safely skip the next item in the working list. The file can also be a software update file, which has corresponding flag "SoftwareUpdate". In this case the file will be copied to replace an older version and the application will have to restart. The advantage of have separate diagnostic and fix file is to have the executables of smaller size and thus save time of download and install when a fix isn't needed.

4.5 Detection of Malicious Node

When the digital signature of some downloaded file cannot be verified, the client will mark the corresponding server from which it downloaded the file as malicious. Then it will send *Warning-malicious-node-detected* message to the index server with the following format:

"Warning"| IP address of malicious server| Infected file name

Figure 6 show warning message sent from client peer to index server. In response the index server will update the type of the corresponding node to "M" which stands for malicious.

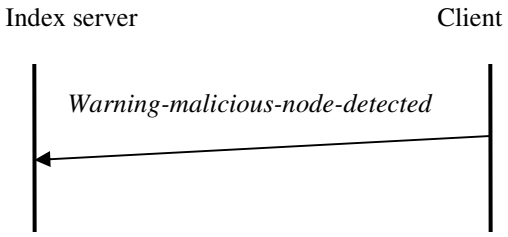

Fig 6: warning message.

4.6 Notification

As mentioned before an index server keeps track of all nodes in the system and their types whether client or server. When a server node X with version number higher than the maximum version number at the index server table connects, the index server will update its maximum version number and will notify all online server nodes that have the same platform as node X of the new updates, as shown in the figure 7. The notification message has the following format:

"NewUpdateAvailable"| New version number

We reason that notifying server nodes is enough as follows. Any node in the system can be either online or offline. If a node is offline then it will go into Server lookup state when it starts up. If it is online in client status, then it will automatically go into Server lookup state after finishing the updating process. On the other hand, the server will react to a notification message by switching to client mode and do Server lookup.

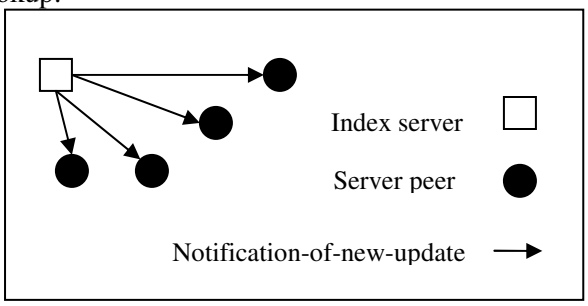

Fig 7: Index server will notify all online server nodes

5 Implementation

For the purpose of this project we have chosen Java as implementation language for multiple reasons. Java supports object serialization that is the ability of a program written in Java to read or write a whole object to and from a raw byte stream. We use serialization to exchange data structures like List of Servers and List of Updates between nodes. This has

simplified coding process and saved time of parsing and processing messages that otherwise could be exchanged using sockets technology. We still make use of sockets to exchange short messages as well as in the implementation of file upload/download object that is discussed in section 4.4.

We use multithreading to enable a server node to serve multiple clients at one time. Also, a server node can initiate a client process meanwhile maintaining connections as server with other client nodes. A client process can initiate multiple threads to download multiple files in parallel. However, the installation of files is kept in sequence since updates are incremental in nature.

One of the features of our software is its ability to update itself. Instead of asking end-users to download a new version of the software; we simply provide the bytecode (.class file) of the updated object. The file is downloaded as explained in sec 4.3 and then installed by replacing the old (.class) object with the new one, as explained in section 4.4.

6 Deployment

In this section we discuss concerns of how to make such network a real-world service.

- Initially, end users who choose to join the network will have to download our software. This can be done through a website that provides the executable as well as installation instructions.
- Our current version supports a single index server. However in real world, multiple index servers should be designed and deployed such that they can collaborate with each other. For example one index server should redirect a client request if it cannot find any matches for clients query in the local index table. Also, an index server should notify other index servers when it detects the release of a new update.
- In order for the client peer to find the IP address of an index server we suggest employing a CDN architecture. A client node will need to know the a single virtual IP address that is translated by DNS to IP address of one of the index servers based on location and load balancing.
- When a new security updated is released, it should be installed manually on some machines (e.g. computers of the update originator) and their version number should be updated. As those machines join the network, the index server will register them as servers and will notify other servers in the network, as explained in subsection 4.6.

7 Conclusion

In this paper we presented a practical design and implementation of a peer-to-peer system that can be used to disseminate security updates. To our best knowledge, this is the first implementation of such service and has many features including: new updated notification, support of multiple platforms, and self-immune to malicious nodes. Our current implementation does not handle virtual IP's. In the future versions we would like to solve this problem. Another enhancement we would like to add in the future is to allow the index server to remove offline nodes from its table and thus improve the chances of connection establishment with a server in the list of servers

References

[1] Z. Tamimi, J. I. Khan, Model-Based Analysis of Two Fighting Worms. *Proc. of ICCCE'06*, Kuala Lumpur, Malaysia, May 2006, 157-163

[2] A. Gupta and D. C. DuVarney, Using predators to combat worms and viruses: a simulation-based study. *Proc. of Computer Security Applications Conference*, Tucson, Arizona, USA, December 2004, 116- 125

[3] F. Castaneda, E. C. Sezer, J. Xu, WORM vs. WORM: Preliminary Study of an Active Counter-Attack Mechanism. *Proc. of WORM'04*, Washington DC, USA, October 2004, 83–93.

[4] H. Kim, I. Kang, On the functional validity of the worm-killing worm. *IEEE Communications*, Paris, France, June 2004, 1902-1906

[5] D. M. Nicol, M. Liljenstam, Models of Active Worm Defenses. *IPSI Conference*, Studenica, Serbia, June 2004

[6] H. Toyoizumi, A. Kara, Predators: Good Will Mobile Codes Combat against Computer Viruses. *Proc. of New Security Paradigms Workshop*. Virginia Beach, USA. September 2002, 13-21.

[7] S. Shakkottai, R. Srikant, Peer to Peer Networks for Defense Against Internet Worms. *Proc. of Inter-Per.* Pisa, Italy, Oct 2006.

[8] Z. Tamimi, J. Khan, Modeling and Analysis of Worm Attacks with Predator and Patching Interplay. *Proc. Of IASTED CIIT'06*, Virgin Islands, USA, Nov 2006.

[9] KaZaa, www.kazza.com

[10] Skype, www.skype.com

[11] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, P. Barham, Vigilante: End-to-End Containment of Internet Worms. *Proc. of SOSP'05)*, Brighton, United Kingdom, October 2005.

[12] M. Vojnovi´c, A. J. Ganesh, On the effectiveness of automatic patching. *Proc. of WORM'*05, Fairfax, VA, USA, November 2005.