Security analysis of Serendipity

Girish Dhoble, Manish Choudhary, Rituraj Satpute and Sahil Chadha
Georgia Institute of Technology

ABSTRACT

In the paper *Serendipity: Enabling Remote Computing among Intermittently Connected Mobile devices*[1] by Cong Shi et al., an algorithm is proposed to speed up computing process using remote computational resources available on other mobile devices. This is achieved by collaboration among mobile devices for task allocation and task progress monitoring. But, Serendipity does not take in account various security threats that emerge during the operation. It assumes that all the nodes in the ad hoc network are trusted which is not the case in real world scenario.

We analyzed the security threats on serendipity system and broadly classify the attacks in two categories, Existing attacks on Serendipity and Attacks specific to Serendipity. As serendipity is basically a mobile ad hoc network, the attacks on mobile ad hoc network are also applicable to serendipity system and fall in the category of Existing attacks on Serendipity, whereas the attacks which target specific functionality of serendipity fall under the category of Attacks specific to Serendipity.

1. INTRODUCTION

The paper *Serendipity: Enabling Remote Computing among Intermittently Connected Mobile devices*[1] by Cong Shi et al. describes an algorithm in which a mobile device uses remote computational resources available in other mobile systems in the adhoc network to speed up computing and conserve energy. Serendipity algorithm disseminates the task among mobile devices by accounting for specific properties of the available connectivity.

The author considers a scenario in which the initiator mobile device needs to run a computational task that exceeds the mobile devices' capability and where portions of task are amenable to remote execution. When a mobile device comes in contact with other devices, it assigns tasks to those devices for execution. After execution of the task, the remote device returns back the result to the initiator device. The challenge facing the initiator device is how to apportion the computational task into subtasks and how to allocate such tasks for remote processing by the devices it encounters.

The paper uses the concept of PNP blocks which is the basic component of the Serendipity system. As shown in the figure 1, the PNP block consists of pre-process program, n-parallel tasks program and a post process program. The pre-process program processes the input data (e.g., splitting the input into multiple segments) and passes them to the tasks. The workload of every task should be similar to each other to simplify the task allocation. The post-process program processes the output of all tasks; this includes collecting all the output and writing them into a single file. The PNP-block design simplifies the data flow among tasks and, thus, reduces the impact of uncertainty on the job execution.

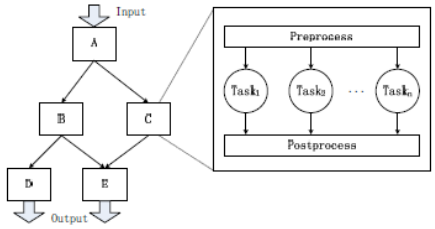


Figure 1: Figure 1 of Serendipity paper [1]

Figure 2 shows high level architecture of Serendipity system. A Serendipity node has a *job engine* process, a *master* process and several *worker* processes. Each node constructs its device profile and, then, shares and maintains the profiles of encountered nodes. A node's device profile includes its execution speed and its energy consumption model. A DAG (Directed Acyclic Graph) as shown in figure 1 is submitted to job engine which gives it to job profiler for basic checking and constructing a complete job profile. If everything is correct, the job engine will launch a new *job initiator* responsible for the new job. It is job initiator's responsibility to disseminate the task and assign task to worker nodes.

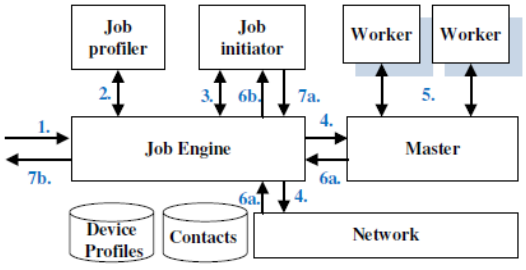


Figure 2: Figure 2 of Serendipity paper [1]

When two mobile nodes encounter, they will first exchange the metadata including their device profiles, their residual energy and a summary of their carried tasks. Using this information, the job engine will estimate

whether it is better to disseminate a task to the encountered node than to execute it locally. Such a decision is based on the goal of reducing the job completion time. For scheduling the task, job initiator uses Water Filling algorithm. For details of the working and experimental results, refer the paper [1].

We analysed the Serendipity paper [1] for the security threats on serendipity system. As Serendipity system is basically a Mobile Adhoc Network (MANET), all the existing attacks on MANET are also applicable to Serendipity system. In addition to these attacks, there are attacks which are specific to the functionality of the Serendipity system. We named these attacks as 'Attacks specific to Serendipity system'. The rest of the paper is divided in following sections. First, the paper describes the existing attacks on Serendipity system which are basically general attacks on Mobile Adhoc Network. Then the paper describes the attacks which are specific to the functionality to the Serendipity system. We have simulated a DoS (Denial of Service) attack on Serendipity system which is attack specific to the functionality of the system. Section 4 describes the experimental setup and the actual attack on the Serendipity system. Screenshots, video and source code of the simulation are provided in the Appendix.

2. EXISTING ATTACKS ON SERENDIPITY

2.1. Attack on Confidentiality

In the Serendipity system, the data is passed between nodes based on the collaboration among the mobile devices for task allocation and task progress monitoring functions. When the data is passed from sender to receiver, all the nodes in the vicinity are able to hear the communication. But as this data is not encrypted, any malicious node can intercept the communication between the sender and the receiver. If a particular node wants to perform a critical computation with some other and it does not want to reveal it to non participating nodes; this would not be possible in Serendipity as without encryption of data the adversary can intercept all communication between the initiator node and the worker node. For example, if this implementation is used in military scenario then the result would be catastrophic as in this scenario the data which is passed between sender and receiver is highly confidential.

Recommended fix

When data or result is transmitted from sender to receiver, the data should be encrypted using provably secure encryption schemes. IND-CPA, IND-CCA and INT-CTXT are some well known security definitions for a scheme which means that for a scheme to be secure, it has to be IND-CPA, IND-CCA and INT-CTXT secure for an efficient adversary. For example, an encryption scheme which is IND-CPA secure is CBC $ (Cipher Block Chaining with random initialization vector) scheme as shown in the figure 3. In this scheme, the message is divided into fixed length blocks and a random initialization vector is used for encrypting each message. At receiver node, the cipher text is decrypted to plaintext using the same random initialization vector passed with cipher text by applying decryption algorithm to the cipher text as shown in figure 4.

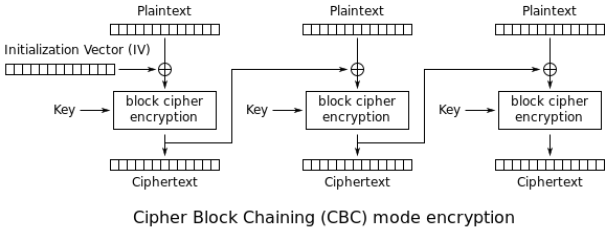


Figure 3: Wikipedia image http://en.wikipedia.org/wiki/File:CBC_encryption.svg

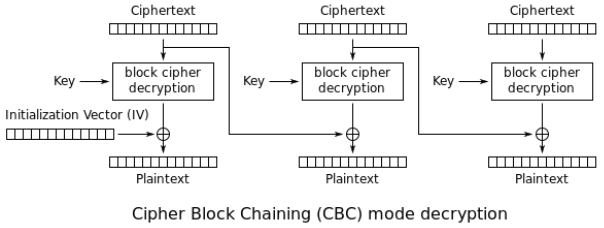


Figure 4: Wikipedia image http://en.wikipedia.org/wiki/File:CBC_decryption.svg

This scheme as mentioned earlier is a secure encryption scheme under IND-CPA security definition.

2.2. Attack on Authenticity and Integrity

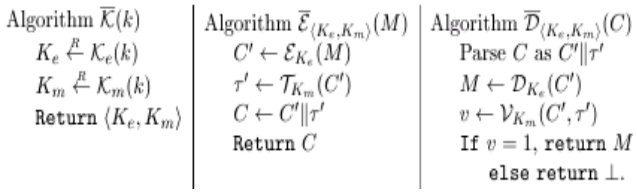
Serendipity does not implement any kind of authentication scheme. So, it is not possible for the node to verify whether the data received is from authentic source or not. Along with authenticity, integrity of the data needs to be maintained. For example, an attacker can interrupt the communication between sender and receiver; modify the data or add malicious content and send this data to the receiver which is then run on the receiver node. Attacker can also modify the result of computation which is sent from worker node to initiator node.

Recommended fix

So, it is necessary for the communicating participants to prove their identities to each other before starting any communication so as to ensure the authenticity. One way to provide authentication is by using cryptographic solutions based on public key certificates [2] to maintain

trust, in which a Trusted Third Party (TTP) or Certificate Authority (CA) [3] certifies the identity associated with a public key of each communicated entities. For maintaining the integrity of the data sent from one node to other, we can use "Encrypt then MAC" scheme in which the message is first encrypted to generate cipher text and then this cipher text is given to MAC oracle to generate MAC (Message Authentication Code). "Encrypt and MAC" scheme is secure under IND-CPA, INT-CTXT and IND-CCA security definitions.

Encrypt then MAC algorithm:



In "Encrypt then MAC" scheme, if adversary changes any part of the cipher text, the value of MAC would change and the entire cipher text will be rejected at the receiver end forcing the sender to retransmit the data. This makes the scheme secure against attacks on integrity and authenticity.

2.3. Attack on Availability

Adversary can attack availability of the system if it inserts malicious nodes in the Serendipity system and in a scenario where the data needs to be passed through the malicious node, it can block the passage of data through malicious node making data unavailable to the receiver for processing.

Recommended fix

For each pair of source and destination nodes, we can have multiple paths to reach from source to destination thus adding redundancy. So, in case there is a malicious node on one path which attacks on the availability of the data, the source can avoid this path and send data on other paths to the receiver. Our proposed solution as described in the report can also be used as a fix to attack on availability.

2.4. Attack on Reputation based systems

The Serendipity paper suggests using reputation based trust in which nodes construct and share reputation information. Basically, reputation allows nodes to form an expectation of behaviour based on the judgements of others, bringing the significant benefit of being able to trust other nodes that are not known directly. Reputation can encourage good behaviour, as a node seek good reputation and benefit from it.

Following attacks are possible on reputation based systems:

a. *Whitewashing attack*:

Whenever the reputation of the malicious node reduces, the attacker resets a poor reputation by rejoining the system with new identity.

b. *Sybil attack*:

The attacker can create multiple identities (Sybil) and exploits them in order to manipulate its reputation score. Sybil attack is described later in detail.

c. *Impersonation and reputation theft:*

The attacker can masquerade the identity of some node and thus can steal its reputation.

d. *Denial of reputation*:

Attacker can damage the reputation of a node (In combination with Sybil attack or impersonation).

e. *Attack on underlying network*:

The reputation system can be attacked by targeting the underlying infrastructure. Reputation information can be attacked (manipulated/replayed/disclosed) both when stored in the nodes and when transported. It can be interrupted by, for example, a denial of service (DoS) attack or misrouting. An infrastructure which does not implement strong identity management may leave the reputation system open to impersonation threats including Sybils (so, for example, anti-spoofing techniques are needed in the anti-spam use-case). The centralised reputation system model suffers because the central entity can become a target and affect the whole system.

f. *Trust topology threats*:

Attacks may try to exploit trust relations among nodes. The attacker may, for example, target links which, if broken (e.g. by DoS), would have maximum effect. The attacker may also investigate which nodes have the highest reputation ratings and attack them, since their recommendations have the greatest impact. In peer-to-peer networks, hijacking nodes with a high reputation rating and therefore high throughput may strengthen DoS attacks. One way for the attacker to perform a trust topology attack is to extract the trust mapping by asking for reputation feedback [4].

Recommended fix

There are two methods for preventing Rushing attack. First is Secure Neighbour Detection in which two nodes detect each other as neighbours only if they can

communicate with each other or they are within maximum transmission range of each other. Second method for preventing Rushing attack is Randomized Message Forwarding in which the receiving node collects a number of RREQs and forwards only a randomly selected RREQ.

2.5. Sybil attack

Sybil attack [5] is basically a process of counterfeiting multiple identities with malicious intent. It is an attack wherein a system or network is subverted by a considerable number of forging identities in peer-to-peer networks. By illegitimately infusing false or biased information via the pseudonymous identities, an adversary can mislead a system into making decisions benefiting her.

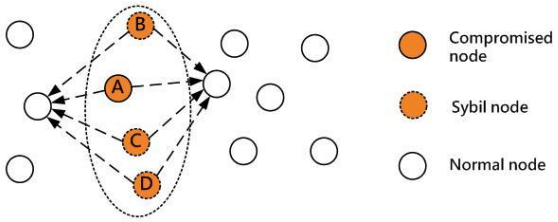


Figure 5

In above figure, initiator node allocates the tasks to nodes A, B, C, and D considering them as unique entities. But actually, all these identities are counterfeited by attacker. So, this will result in submission of all the tasks to the attacker and will hamper the computation and can also result in denial of service attack. This can also be used by the attacker in the reputation based serendipity system to affect the reputation of legitimate nodes and thus disrupting the sole purpose of the network.

Recommended fix

Sybil attack can be avoided using centralized or semi centralized trusted identity management authorities, by using specific system features based techniques, Sybil guard and other techniques as suggested in paper "A Survey of Sybil Attacks in Networks"[5].

2.6. Black Hole attack

Black hole [6] in MANET is a serious security problem to be solved. In this problem, a malicious node uses the routing protocol to advertise itself as having the shortest path to the node whose packets it wants to intercept. In flooding based protocol, if the malicious reply reaches the requesting node before the reply from the actual node, a forged route has been created. Hence, all the packets for that route will go through this malicious node. This malicious node then can choose whether to drop the packets to perform a denial-of-service attack or to use its place on the route as the first step in a man-in-the-middle attack.

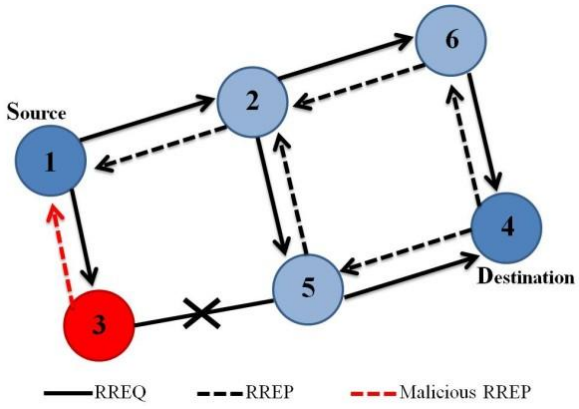


Figure 4

In serendipity system, attacker can use Black hole attack to launch Denial of Service (DoS) attack in which the attacker will forge the reply coming from the receiver and hence hampering further computation. Also, the attacker can advertise itself as a node with high computational power to get all the computational tasks and then can launch DoS attack. Also, the data which is passed from the sender to the receiver can be interrupted by the attacker to launch man in the middle attack.

Recommended fix

For preventing Black Hole attack, the node should wait and check the replies from all the neighbouring nodes to find a safe route to destination. A node can also monitor behaviour of its neighbouring nodes to prevent Black Hole attack.

2.7. Wormhole attack

In a wormhole attack, an attacker forwards packets through a high quality out-of-band link and replays those packets at another location in the network. It is also possible for the attacker to forward each bit over the wormhole directly, without waiting for an entire packet to be received. An attacker can create a wormhole even for packets not addressed to it, since it can hear them in wireless transmission and tunnel them to the attacker at the opposite end of the wormhole as shown in figure 7.

In serendipity system, the attacker can forward the data that it received for computation to a malicious node through a tunnel. It can also forward the result of the computation to the malicious node. If this data is confidential and the initiator node does not want the data to be disclosed to non participants, then this is an attack on confidentiality of the serendipity system.

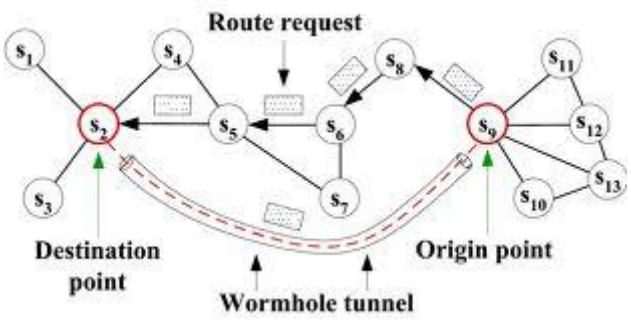


Figure 7: http://www2.engr.arizona.edu/~llazos/research.php

Recommended fix

Whenever the source node sends RREQ for finding the route to destination, if it does not receive a RREP packet within some specified period of time, it should add the route to wormhole list. Also, nodes should monitor the behaviour of neighbouring nodes for preventing wormhole attack.

2.8. Flooding attack

The Route Request (RREQ) Flooding Attack is a kind of denial-of-service attack, which aims to flood the network with a large number of RREQs to the destinations in the network. In this attack, the malicious node will generate a large number of RREQs, possibly in the region of hundreds or thousands of RREQs, into the network until the network is saturated with RREQs and unable to transmit data packets. In serendipity system, malicious initiator can flood many fake tasks and give these to its worker nodes for computation hence exhausting their computing power and resources. Flooding attack can be prevented using flooding attack prevention mechanism. In this mechanism, a trust function is used by the nodes to forward the RREQ packets. If a particular node shows malicious behaviour by flooding the network, its trust would be decreased and the RREQ packets from that node would not be forwarded.

2.9. Grey Hole attack

The grey hole attack is also a kind of DoS attack. Grey hole attack is an extension of black hole attack in which malicious node behaviours and activities are exceptionally unpredictable. This attack is difficult to detect than black hole attack as the malicious node behaves normally most of the time. A gray hole may exhibits its malicious behaviour in various techniques. It simply drops data coming from certain specific node in network, while forwarding all data for other nodes. Another type of gray hole attack is a node behaving maliciously for some time duration by dropping requests for some period. This attack may also exhibit a behaviour which is a combination of above two, making detection of attack more difficult.

In serendipity system, a malicious node can sometimes show malicious behaviour and other time normal behaviour. It can sometimes hinder the computation process by not returning the result of computation to the initiator node. Other times it shows normal behaviour which makes it difficult to differentiate malicious nodes from normal nodes.

2.10. Rushing attack

Rushing attack is an effective denial-of-service attack against all currently proposed on-demand ad hoc network routing protocols, including protocols that were designed to be secure.

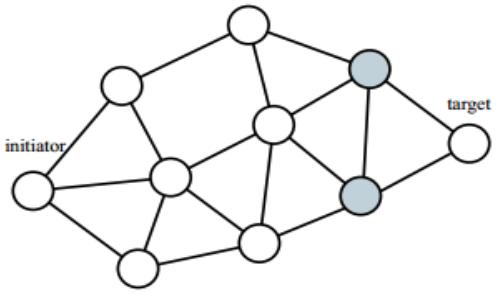


Figure 8

The initiator node initiates a Route Discovery for the target node. If the ROUTE REQUESTs for this discovery forwarded by the attacker are the first to reach each neighbour of the target, then any route discovered by this Route Discovery will include a hop through the attacker. That is, when a neighbour of the target receives the rushed REQUEST from the attacker, it forwards that REQUEST, and will not forward any further REQUESTs from this Route Discovery. When non-attacking REQUESTs arrive later at these nodes, they will be discarded. As a result, the initiator will be unable to discover any usable routes (i.e., routes that do not include the attacker) containing at least two hops (three nodes). An attacker that can forward ROUTE REQUESTs more quickly than legitimate nodes can do so, can increase the probability that routes that include the attacker will be discovered rather than other valid routes an attacker that can forward ROUTE REQUESTs more quickly than legitimate nodes can do so, can increase the probability that routes that include the attacker will be discovered rather than other valid routes as shown in figure 8.

A more powerful rushing attacker may employ a wormhole to rush packets. In this case, the attacker simply forwards all control packets (but not data packets) received at one node (the attacker) to another node in the network (e.g., a second attacker). This forms a tunnel in the network, where packets reaching one end of the tunnel are broadcast out the other end. If the tunnel provides

significantly faster transit than legitimate forwarders, nodes near one end of the tunnel generally will be unable to discover working routes to the other end of the tunnel, since it will generally discover routes through the tunnel.

In serendipity system, a malicious node uses rushing attack to become an intermediate node in the entire route discovered. Hence all the data communication takes place through the malicious node which can be intercepted, interrupted or dropped resulting in disruption or denial of service.

3. ATTACKS SPECIFIC TO SERENDIPITY

3.1 Malicious data attack

The experiment in the paper [1] divides a text-to-speech file into multiple PNP blocks and distributes these among the nodes. The nodes work on these blocks and return the results which are assimilated by the initiator. One cannot help but think about the nature of the piece of code that works on this conversion. What if the code is malicious in nature and the job initiator leverages lack of checking mechanisms to compromise nodes? Let us examine such an attack in greater detail.

The ecosystem that Serendipity has the most potential for deployment in is the Smartphone-ecosystem due to its ubiquity and ever increasing computing power. We all know that the current smart phone market is dominated by iOS and Android. Most of us are also aware that a huge number of users root or jailbreak their devices (unfortunately, there are no official sources for this. But, for example, CM for android has millions of downloads). As a result of this, a malicious job initiator can very easily run malicious code on any node with supervisor rights.

Let us first examine an attacker that directly injects malicious code to run on a node.

A number of attacks are possible even if the device is not jail broken:

a. Job that retrieves personal information stored on the device like contacts, location history. Consider this simple piece of code simply retrieves the contacts and sends it back to the job initiator:

```
Uri personUri = ContentUris.withAppendedId
(People.CONTENT_URI, personId);
Uri phonesUri = Uri.withAppendedPath(personUri,
People.Phones.CONTENT_DIRECTORY);
String[] proj = new String[] {Phones._ID, Phones.TYPE,
Phones.NUMBER, Phones.LABEL}
Cursor cursor = contentResolver.query(phonesUri, proj, null,
null, null);

(add code for returning to initiator)
```

b. A job that simply installs an ad-server on the device to bombard the device with advertisements.

c. A job that self-replicates and essentially infests the entire network.

However, attacks that utilize root privileges are a lot more dangerous:

a. Recently, there have been attacks [*http://research.nq.com/?p=391*] that modify the code for inbuilt commands to perform customized action. An example is to modify the code that runs during start-up to set up the malicious code even after the job initiator is out of the picture. A backdoor, if you will. Note that modification on system code is not allowed if the device is not rooted.

b. There was a recent attack [9] on the bitcoin client of many android devices. The flaw was improper random number generation because the program did not use /dev/urandom for seeding the generator. We also know that android heavily relies on /dev/urandom for seeding the random number generator in TLS/OpenSSL. A deadly attack that uses root privileges can be mounted using Serendipity as follows:

a. A job initiator creates PNP blocks. Each block essentially alters the /dev/urandom. The block returns some trivial result to indicate success.

b. The attacker knows which nodes are compromised. There is a good chance that anything that makes use of /dev/urandom in the compromised nodes can behave according to the attacker.

c. The attacker sniffs all wireless network at the compromised node. Any TLS traffic, like bank logins, are no longer secure due to the compromised /dev/urandom.

Now let us consider a job initiator that creates PNP blocks consisting of code that is vulnerable, intentionally or unintentionally. A similar attack has already been discussed [10] in an Android device that uses:

a. Android Scripting Environment that allows interpretation of multiple programming languages on the device.

b. A vulnerable app already installed on the device. In the case of Serendipity, this is replaced by the code inside that PNP block that the node has to execute.

c. The above are combined to mount a privilege escalation attack on the device. This can be further combined with the attacks previously discussed that leverage the fact that a device is jail broken.

Another popular attack that falls under the same category can be carried out using return oriented programming.

a. Job initiator A sends PNP block that contains code that returns the instructions already residing in the node's device memory. This forms the basis of the ROP attack. The returned result contains the return addresses that the attacker can use on each node.

b. A new set of PNP blocks are disseminated by A that contain vulnerable code and are then run on the nodes.

c. A colluding attacker B or another device owned by A then creates a job profile that contains PNP block with malicious code that are disseminated throughout the network. Based on the current node that executes each PNP block, the code uses the appropriate return addresses retrieved earlier and combines them with the vulnerable code that A sent earlier to hijack the stack execution flow of the device.

d. This can allow the attacker to do anything from spawning a remote shell to executing in built system code to do the attackers bidding.

Recommended fix

All the attacks discussed above are possible due to the fact that a node, upon receiving a PNP block, blindly runs it without first checking it for vulnerabilities and maliciousness. Thus, the best solution for this is to have a job checking mechanism in place in Serendipity to thwart such attacks.

Out of all the components of the architecture of Serendipity, the component that is best to incorporate the job checking is the Master as every PNP block as well as its result has to go through the Master.

The modified flow is as follows:

a. After receiving a job, the job engine constructs the job profile and starts a job initiator who will initiate a number of PNP-blocks and allocate their tasks.

b. The job engine disseminates the tasks to either local or remote masters.

c. The remote or local masters examine the job and check for vulnerabilities or malicious code.

d. After a worker finishes a task, the master sends back the results to the job initiator (6a, 6b), who may trigger new job PNP-blocks.

After all results are collected, the job initiator returns the final results and stops.

There are a number of methods to check a job:

a. Match the job signature with a database of vulnerable code and malicious code. Something similar to what antivirus do.

b. Use behaviour analysis to determine if the job is safe or not: the behaviour analyzed should be the network traffic behaviour and the behaviour of the system should also be examined to determine if the job is malicious or vulnerable.

A more elaborate and complex modification to the Serendipity to accomplish job checking is to modify the protocol. Let the network have n nodes. Then, using each node's reputation, a small subgroup of nodes is selected to function as "security guards". The job of the security guards is to make sure that the jobs being disseminated throughout the network are safe. However, this would mean that all the traffic is essentially bottlenecked through this subgroup of nodes and might adversely affect the performance.

However, the security guards need to employ multiple measures to ensure the safety of the jobs by:

a. Running the job in a sandboxed or virtual environment and examine its result, based on which it will classify it as safe or not and forward accordingly.

b. Employ intrusion detection on: the network traffic that the job initiates and how the job affects the system behaviour.

3.2 Denial of Service attack

After job initiator distributes the jobs, the nodes carry out the job as assigned by the initiator and once the job is completed the worker nodes send back the result to the initiator node. But the problem with this approach is while sending back the result; the initiator does not perform a check whether the result came from authentic node or is it someone who is just claiming to be the worker node (Malicious node). In this case the malicious node would return fake result to the initiator node resulting in an error.

Also, there could be a malicious node which is aware of the ongoing transmission (NAV field of RTS/CTS) it can interfere with the transmission of link layer frame by transmitting a small number of bits.

Malicious node can also inject random data packets in the network so, as to consume the resources (especially battery) of the other nodes to perform its useless task which will result into depletion of their resources and thus, they would not be able to provide service to some genuine nodes. Due to mobility of the malicious node it

can deplete resources of new nodes every time it comes in contact with new neighbours.

As shown in Figure 9(a) the malicious node first consumes the resources of v1,v2 and v3. Then due to mobility it comes in contact with v4,v5,v6,v7 and consumes their resources, hence causing denial of service for all the nodes that need the above nodes to perform their tasks.

The attacker instead of targeting the network can target a particular node which does most of the work and try to consume all of its resources. As this node is used by most of the other nodes to perform their tasks, the absence of it will degrade their performance and would lead the initiator to perform all the calculations locally and thus killing the sole purpose of the serendipity model.

Potentially the most dangerous attack that can be mounted is if the job initiator is an attacker. An attacker can deliberately create a job containing some malicious code packed in it. When the job gets distributed among the worker nodes and they start executing the job, the malicious code runs on these worker nodes which may result in malfunctioning or compromising of these nodes. This attack holds unless there is an effective job checking mechanism in place

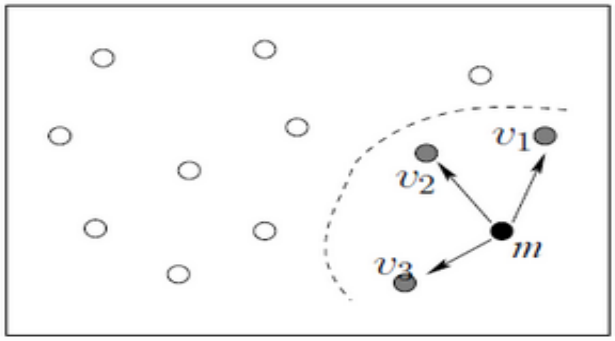


Figure 9 (a)

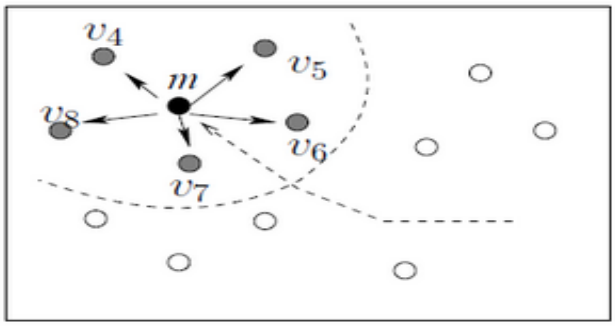


Figure 9 (b)

Malicious Code Execution: There is no check whether the job being performed contain some malicious code or not. So, if a job initiator itself is an attacker he can compromise his neighbours and these compromised neighbours now are new source of attack which attack their neighbours. In other words, these compromised neighbours can compromise their neighbours so effectively the whole network can come under the attacker's control as shown below in Figure 10.

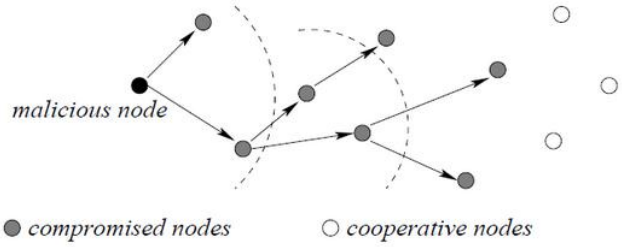


Figure 10

Attacker can even speed up the propagation of malicious code by utilizing the contact information and finding the nodes which are not its immediate neighbours but are 2 hop away and can perform the task faster. So, when initiator wants to propagate malicious code then it transfers the job containing malicious code to its neighbours which in turn transfer it to their neighbour without checking the job. Thus the network is compromised faster as the co-operative node will always find a path between the attacker and the 2-hop compromised node because of its cooperative nature. Thus an attacker can form an overlay network on the original network efficiently as shown below in Figure 11.

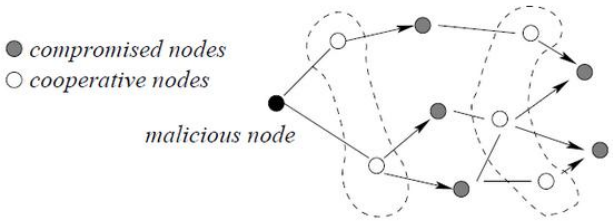


Figure 11

4. SIMULATION OF DENIAL OF SERVICE ATTACK

4.1 Experimental Setup

We use Network Simulator 2 to simulate an experiment with 5 nodes wherein one node is the job initiator, three nodes are legitimate wireless nodes and one node is a malicious wireless node. Given below are the generic parameters of the said experiment:

Medium Access Control	:	802_11
Routing Protocol	:	AODV
Energy Model	:	Energy Model
Number of Nodes	:	5

Given below are some of the generic node parameters:

Receiving Power : 1.0
Transmit Power : 1.0
IFQ Type : Queue/Droptail/Priority Queue
IFQ Length : 50
Initial Energy : 10
Channel : Channel/Wireless Channel

We assign the following properties to the malicious node:

Initial Energy : 1000
Movement Pattern : Same as that of the job initiator

Assumptions as per Serendipity paper [1]:

a. Considering the predictable contacts model, the job initiator can predict the future contacts in the network.

b. The job initiator is aware of the estimated energy consumption at each node.

c. The job engine follows the Water Filling (greedy) algorithm as a result of which it always disseminates the job to the node with the highest amount of energy at its disposal. Also, the malicious node can trivially advertise itself as having high computing power as it does for the amount of energy.

d. The malicious node has unlimited energy and computing power at its disposal.

e. The malicious node is aware of the job initiator's movement patterns.

Given below is the ns2 code for selecting the best node for task dissemination:

```
set maxenergy 0

set bestworker 0

set neighbors [$node_(0) neighbors]

foreach nb $neighbors {

  set en1 [$nb energy]

  puts $en1

  puts [$nb id]

  if {$en1 > $maxenergy} {

     set maxenergy $en1

          set bestworker $nb

  }

}
```

(Entire source code in Appendix A1.)

The Attack:

As per the greedy algorithm, the job engine will always disseminate the tasks to the node with the minimum task completion time (and maximum energy). This vulnerability is exploited by the malicious node by advertising that it always has unlimited computational power and energy at its disposal thus resulting in the less task completion time with high availability. Furthermore, it follows the job initiator's movement patterns. As a result of this, the job initiator is always in contact with the malicious node.

So, whenever the job engine has to disseminate tasks, it always gives preference to the malicious node over any other nodes that it may come in contact with. Thus the malicious node:

a. Monopolizes the task dissemination process through which it can mount a DOS attack on the job initiator.

b. Can manipulate the result of the tasks at a bigger scale.

The simulation is recorded in a video which is essentially what has been described above. Screenshots of the simulation and link for the video are added in the Appendix.

5. PROPOSED SOLUTION

Serendipity is an efficient and promising system for remote computing among intermittently connected mobile devices. But, it focuses on the functionality only assuming that all nodes are collaborative and trustworthy. In practical scenarios, some nodes may be selfish or even malicious for which the paper[1] suggests some notional credit based system or reputation based system. But, various deficiencies in such systems have been explained in the previous sections. Various attacks on the serendipity system (as discussed above) raises the requirement of a modified serendipity system which takes security into consideration along with the functionality.

This paper proposes a modified serendipity system, Secure Serendipity that addresses many of the security concerns and tries to minimize the attack surface.

A. Base Idea

The basic idea, for implementing security in serendipity, is to make some of the nodes in the network responsible for the security. So, some nodes are assigned with this responsibility by making them "trusted nodes". This idea is similar to the route reflectors concept where certain nodes become responsible for controlling the system making it more efficient or secure(in this case).

B. Assumptions

The proposed system assumes a network setting with predictable contacts i.e. the future contacts can be predicted efficiently. The time in the network is denoted by t and solution assumes that there are n nodes in the network. The network is mobile but not more than some p nodes join or leave the network at one time.

The basic requirement for the security of any system is to ensure confidentiality, authenticity and integrity. To fulfil this, some secure hybrid encryption scheme should be implemented. Many such schemes for mobile ad-hoc networks have been published and thus, can be used for encrypting all the communication among the nodes. The proposed solution assumes that the communication channel is encrypted.

We also assume that the malicious nodes are present in the network but are not in the majority. It can easily be argued that this assumption is very reasonable and practical. If the malicious or selfish nodes are present in the network they would try to compromise each other or get benefits from others with providing service. Such a network can't be used for remote computing and thus, there will be no requirement to provide security or even to run serendipity system in such a network.

C. Functionality

The functionality of 'Secure Serendipity' has been divided in three phases which have been explained as follows.

Phase I - Building the Trust

This is the initial phase when the network in newly built (t = 0) and no job execution has been carried out yet. The following actions are performed during this phase.

1. The normal serendipity system is run for some time, say from t=0 to $t=t_o$. During this period, each participating node maintains a trust matrix where it keeps track of the behaviour or trust factor of other nodes. The trust factor has to be computed by each node in following two scenarios.

a. If the node is the job initiator, it will maintain a trust matrix for the participating worker nodes on the basis of the behaviour and performance of each node. Successful completion of the assigned task, time taken to complete the task and the quality of the results are some of the parameters that can be taken into consideration while computing the trust factor.

b. If the node is a participating worker node, it will keep an entry of trust factor in the trust matrix for the job initiator node. The size of the job, the stability of the system and status of battery before and after execution of the job are some of the parameters that can be taken into consideration while computing the trust factor.

Whenever two nodes meet each other, they share their matrices with each other.

2. At time t_0, it is assumed that each of the participating nodes has trust matrices of all other participating nodes. So, at t_o, a weighted average of the trust factor values, for each node maintained by all the other participating nodes, is computed which becomes the trust factor for that node in the network.

3. (m+k) nodes with the highest trust factor are selected out of which m nodes are randomly picked to function as 'trusted nodes' in the network.

The m nodes are chosen in a manner such that they are evenly distributed across the ad-hoc network and each untrusted node has at least one trusted node. m should be chosen properly with respect to n as too small or too large value of m may adversely affect the efficiency and the security of the system.

Phase II - Job Execution

In this phase, the trusted nodes are responsible for the security of the network. Following are the requirements for the proper functionality of the trusted nodes:

1. The untrusted nodes function as the worker nodes and thus, can be assigned tasks for execution.

2. The trusted nodes are exempted from the responsibility of functioning as a worker node and thus, can't be assigned any job for execution.

Following are the responsibilities of a trusted node in the network:

a. **Sandboxing:** The trusted node is responsible for checking for the maliciousness of any request for task distribution or any result received. For this the trusted node maintains a simulated environment in which it runs the code partially to observe and analyze its behaviour.

b. **Blacklisting:** The trusted nodes are responsible for blacklisting any node with malicious intent. If the code running in simulated environment shows some malicious behaviour, the requesting node gets blacklisted and thus, would not be considered as part of the network further.

c. **Trust Matrix:** The trusted nodes maintain a trust matrix with entry for each participating/requesting node based on the behaviour of the nodes in the network.

d. **Job Hashes:** To improve the efficiency of the whole system, the trusted nodes maintain the hashes of the jobs already checked for the malicious intent along with the results. For each request received, the trusted nodes compute the hash of it and look for a match in the database. If found, it can take action accordingly without requiring the partial run of the code.

e. **Prevent DDoS**: The trusted nodes ensure that the network is not under denial of service attack at any point of time. For this, checks are done under following two scenarios.

i. Malicious node as job initiator: The node attempting DDoS may assign the tasks to all the efficient nodes in the network and thus, making them unavailable for the genuine request. The trusted nodes prevent this scenario by ensuring that a particular task distribution request covers only a fraction of nodes currently present in the network and thus, other nodes are available to serve as worker nodes for genuine requested. For this, a threshold value is maintained and if the number of 'worker nodes to be assigned the tasks' exceeds this value, the task distribution request is rejected.

ii. Malicious node as high energy worker node: In this scenario, the malicious node may attempts to launch the attack by advertising itself as the high energy and most efficient node and thus, pretending as the best candidate for task allocation. In this way, the malicious node may collect many tasks to be executed but wouldn't return the results. This would force the job initiator nodes to execute the tasks locally and thus, defeating the whole purpose of serendipity. The trusted nodes prevent this attack by ensuring that a particular worker node has not been assigned tasks more than a threshold value.

f. **Communication among trusted nodes:** Trusted nodes are responsible for communicating with each other to share the trust matrices, numbers of jobs assigned to each worker node, information about blacklisted nodes, task distribution requests and their predictable contacts.

g. **Availability of predictable contacts:** If requested, the trusted nodes should make their predictable contacts available to any of the node in the network.

Any job execution goes through following steps:

1. The node with some job to be completed in distributed manner, requests the nearest trusted node for its predictable contacts.

2. Job initiator divides the job into tasks based on the predictive contacts received from the nearest trusted node.

3. Job initiator sends a task distribution request to the nearest trusted node.

4. The trusted node runs any executable code partially in a virtual environment to ensure that it is not malicious. The trusted node also checks that the trust factor of requesting node is above some threshold value.

5. If the job is found malicious, the task distribution request is rejected and the job initiator adds the requesting node to the blacklist prohibiting it from making any requests in future. If the value of trust factor for the requesting node is below the threshold, the request is rejected.

6. If the job is not malicious, the trusted node disseminates the tasks just like in Serendipity, forwarding the tasks to the nodes on which the job initiator intended the job to be executed.

7. If the trusted node meets another trusted node before meeting some node to which the task is to be assigned and based on predictable contacts, if that trusted node is going to meet the worker node sooner, the task distribution responsibility is forwarded to it.

8. The tasks are executed on the worker nodes and returned to the originator through the trusted nodes.

During this process, the trusted nodes and untrusted nodes maintain trust matrices for each other based on the behavior and performance parameters.

Incentives for Trusted Nodes: The trusted nodes are responsible for the security of the network. This involves continuous involvement in the network activities along with some tasks to be executed locally on the trusted nodes. To motivate the trusted nodes, they are allowed to get their jobs executed with a higher priority over the jobs from other nodes. The only constraint is that if a trusted node wished to be a job initiator, it should request this to some other trusted node just like any untrusted job initiator. This helps in preventing an attack in case the trusted node is a malicious one as its task distribution request would be scrutinized by another trusted node.

Controlling New Members: As the nodes in the network wouldn't have an entry for a new node joining the network who has not performed any activity. Thus, this new node wouldn't be allowed to request task distribution as its trust factor wouldn't be available. The new node is supposed to act as a worker node for sometime, build the trust and then get the benefits from others. This ensures that the selfish nodes without having the trust are not misusing or disrupting the network activities.

Phase III - Rebasing the Trust

After a fixed time interval, all the matrices for trusted and untrusted nodes are shared and the selection process is repeated using the updated trust factor values of the nodes. In the selection process, (m+k) nodes with highest trust factor values are chosen as candidates and from these (m+k) nodes, m nodes are selected to work as the new trusted nodes in the network.

The selection of the trusted nodes takes place with following constraints:

1. The same node is not selected as the trusted node three times in a row.

2. The new nodes with the trust factor values can't participate in the selection process and thus, can't be chosen as the candidate trusted nodes.

D. Analysis and Security Aspects of Secure Serendipity

As discussed above, Secure Serendipity with encrypted communication channel ensures confidentiality, authenticity, integrity and availability. Sandboxing prevents execution of any malicious code on the worker nodes and thus, protect them against any compromise. Blacklisting helps in secluding any malicious node as soon as it is detected malicious. The system provides protection against denial of service attacks and also controls the selfish nodes.

If the malicious node gets selected as the trusted node, authenticated encryption ensures that this node is not able to inject some malicious content.

Even in the initial run of serendipity(without trusted node), malicious nodes can't affect trust factor computation and thus, the selection of trusted nodes as we are taking weighted average of the trust factor values maintained by all the nodes in the network. Therefore, the malicious nodes, which are in minority, won't be able to affect the results much.

Reason for Two-Phase Selection: The selection process involved picking (m+k) nodes with highest trust factor out of which m nodes are selected at random as the trusted nodes. The benefits for selecting in two phases are:
1. If we select the top m nodes as the trusted nodes unconditionally and some malicious node is able to maintain itself among these m nodes, it would be picked as the trusted node. So, randomization tries to minimize this possibility.

2. The trusted m nodes are essentially barred from executing serendipity tasks. Selecting the top m nodes with highest trust factor value (and high performance) would adversely affect the performance of the system.

Reason for Restriction on Three Times Continuous Selection of a Node: The same node is not selected as the trusted node 3 times in a row. The reasons for this approach are:

1. This ensures that high performance nodes are not always locked into being trusted nodes that never execute tasks.

2. We insist on 3 times because restricting the number to 2 will render the network with n-m nodes to select trusted nodes from, which might be too less and can be exploited by an attacker.

E. Limitations

Although Secure Serendipity facilitates remote computing in a secure environment, there are some limitations of the system.

1. Degrades performance of Serendipity: The availability of only (n-m) nodes as the worker nodes in the network, task distribution through trusted nodes and sandboxing degrades the performance of the serendipity system. Thus, this system is more useful in cases where large sized jobs are to completed in distributed manner.

2. Requirement of encrypted communication channel: The system assumes that all the communication takes place through an encrypted channel and this assumption provides confidentiality and authenticity. If the encrypted communication channel is not available, it may introduce new attack vectors.

3. Trade-off between security and efficiency: Security of any system always degrades its performance. Same principle applies in this case and thus, Secure Serendipity is not as efficient as serendipity. It is a trade-off between the security and the efficiently, and should used wisely as per the requirements. Secure serendipity is more useful in the networks where security is one of the primary requirements. For example, in military scenarios, this system can be really useful as the strategic planning provides the predictive contacts and the remote computing can be performed in a secure environment.

F. Future Work

The proposed solution works in the case of predictive contacts. The solution can be extended for more realistic scenarios where the contacts are not predictive. We would also like to implement this system to perform a comparative study of Serendipity System and Secure Serendipity System in terms of performance and security.

6. CONCLUSION

The system proposed in the paper [1] provides better performance by utilizing remote computational resources in mobile adhoc network, but there are many attacks which destroy the purpose of using this system. Denial of Service attack on Serendipity system was simulated in NS2 simulator. With this attack the job initiator was not able to assign tasks to other nodes which forced it execute the tasks locally. The paper suggests modification to the original Serendipity system which deals with the problem of handling malicious nodes.

7. REFERENCES

[1] Cong Shi et al. "Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices"

[2] S. Li and X. Wang, Enhanced security design for threshold cryptography in ad hoc network.

[3] B. Wu, J. Chen, J. Wu, and M. Cardei, "A survey of attacks and countermeasures in mobile ad hoc networks," Wireless Network Security, pp. 103–135, 2007.

[4] Rahman, A Framework for Decentralised Trust Reasoning, PhD Thesis, University College London, 2005

[5] A Survey of Sybil Attacks in Networks Wei Chang and Jie Wu

[6] Mohammad Al-Shurman and Seong-Moo Yoo et al, Black Hole Attack in Mobile Ad Hoc Networks

[7] Jian-Hua Song et al, Effective Filtering Scheme against RREQ Flooding Attack in Mobile Ad Hoc Networks

[8] Yih-Chun Hu et al, Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols

[9] http://www.extremetech.com/computing/164134-how-bitcoin-thieves-used-an-android-flaw-to-steal-money-and-how-it-affects-everyone-else

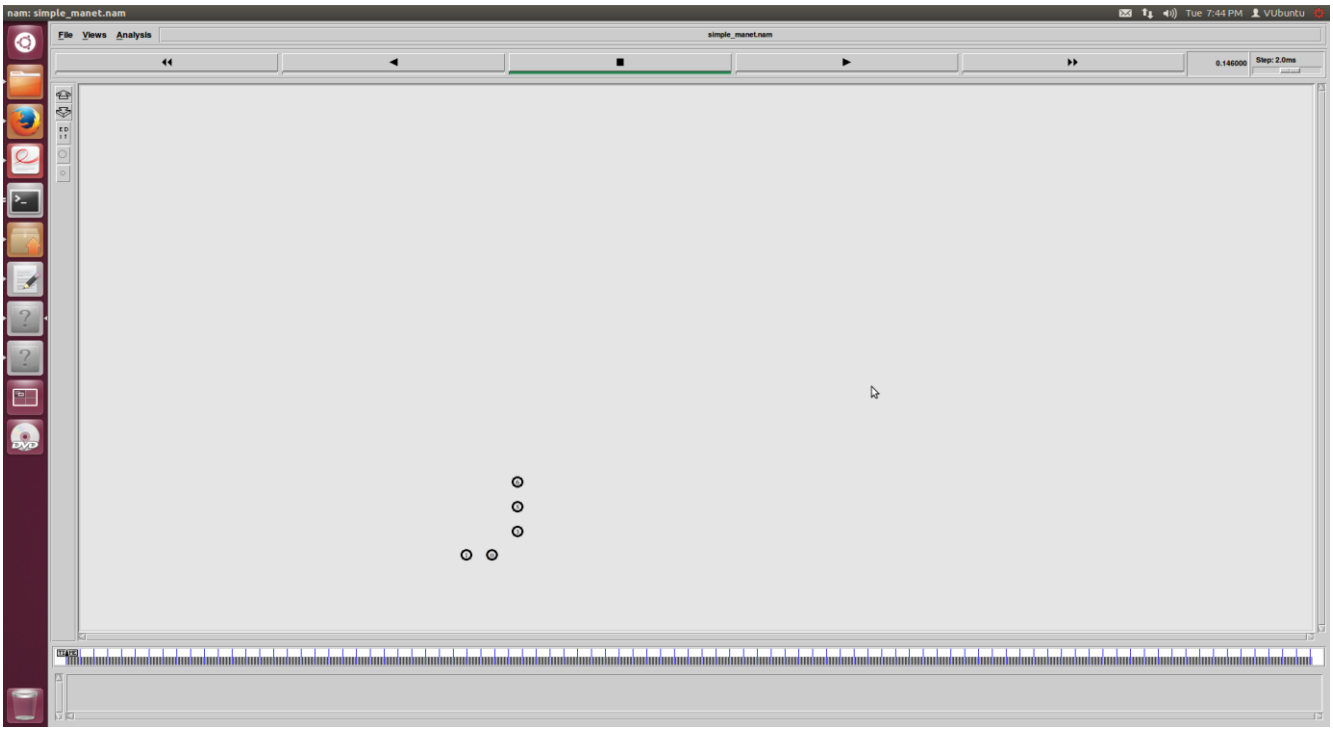
[10] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Marcel Winandy, Privilege Escalation Attacks on Android

[11] Fei Xing et al, Understanding Dynamic Denial of Service Attacks in Mobile Ad Hoc Networks

Appendix A

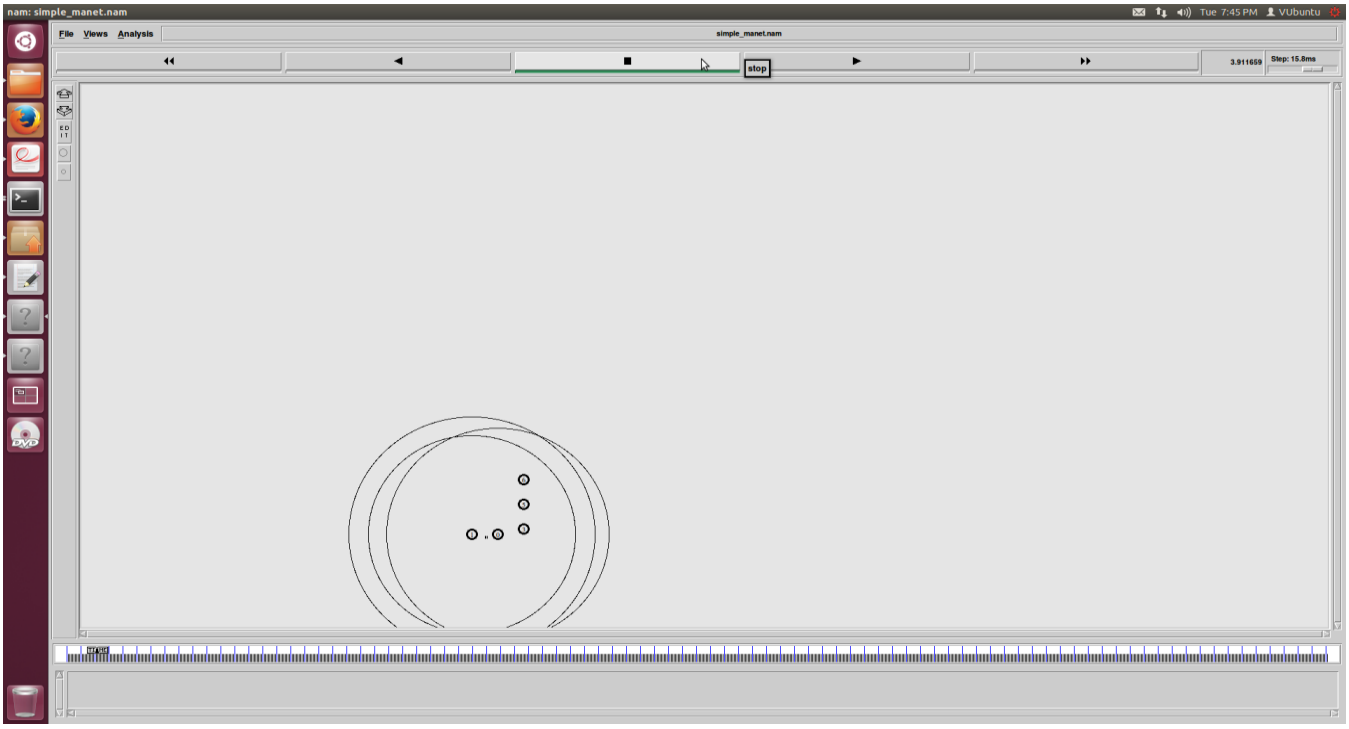
SCREENSHOTS OF SIMULATION

Screenshot 1



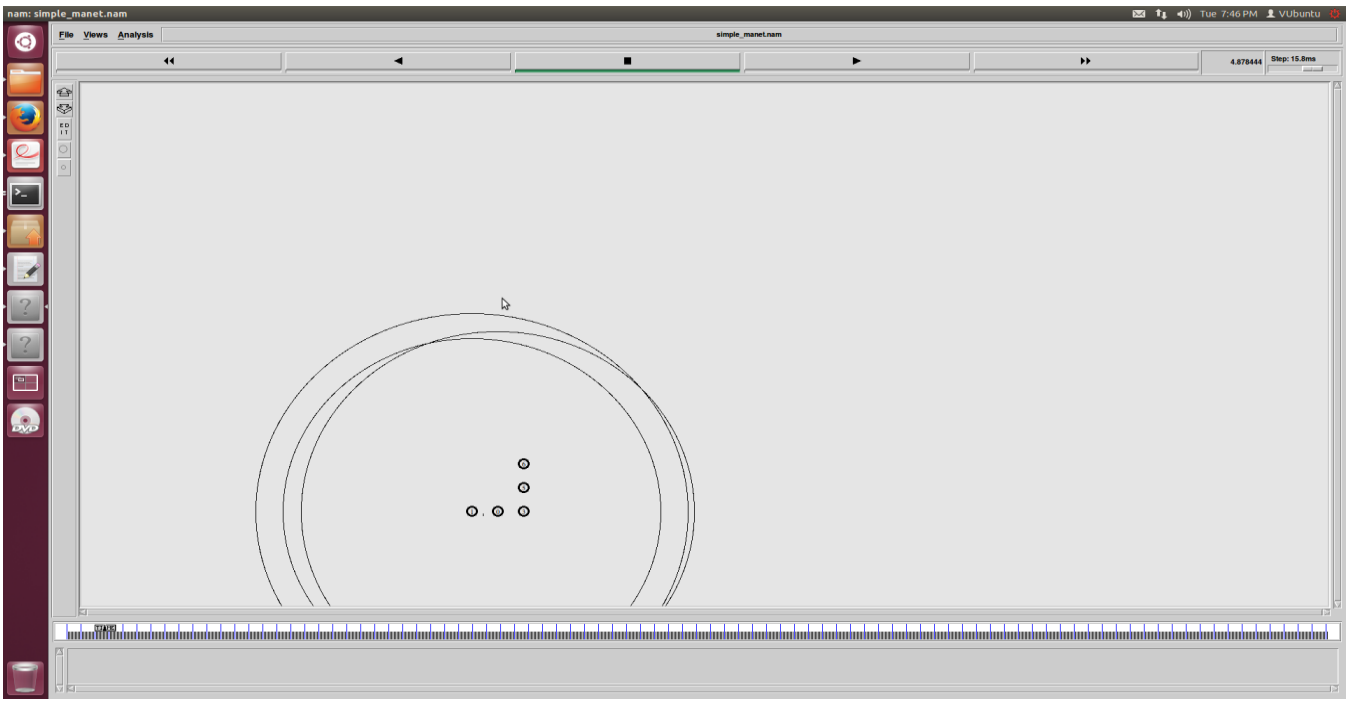
Above screenshot shows the initial position of the nodes at time t=0.

Screenshot 2



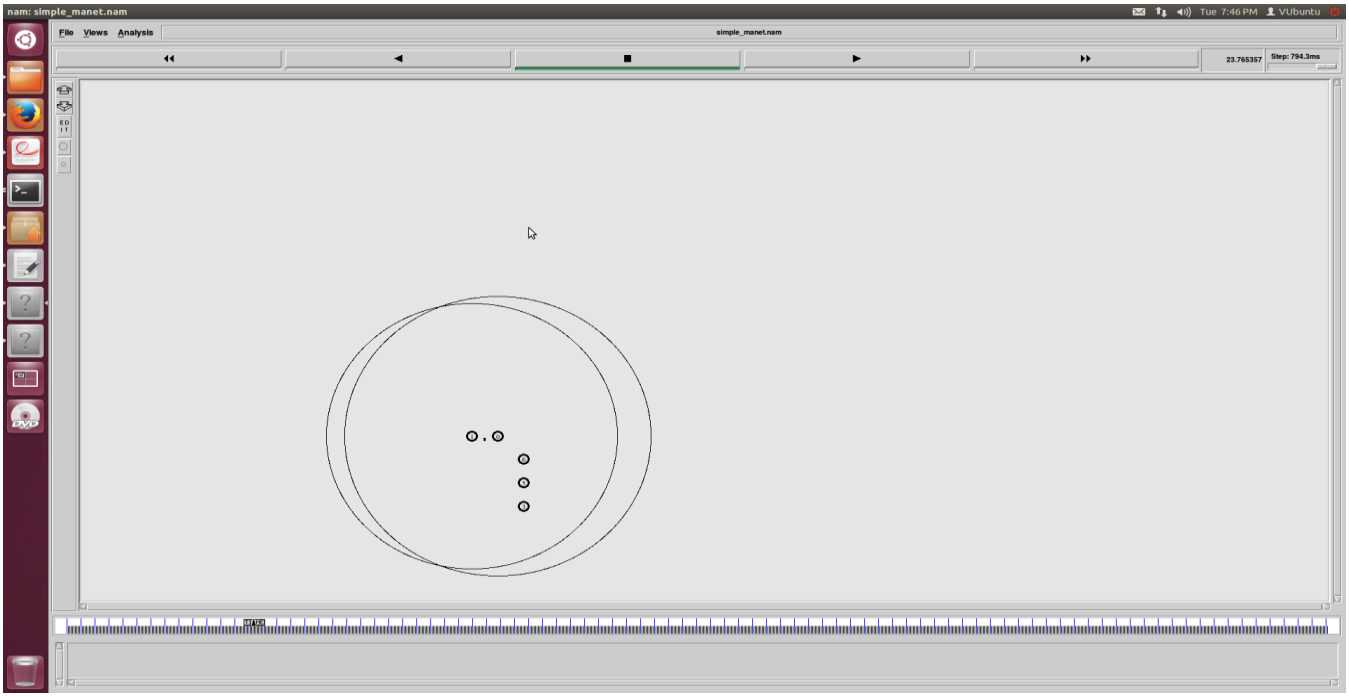
At time t=2, the three nodes i.e. job initiator, the malicious node and the candidate worker node come into contact with each other. The job initiator gives preference to the malicious node and assigns the task to it even though the legitimate candidate worker was in its contact.

Screenshot 3



The job initiator continues to transfer data to the malicious node.

Screenshot 4



The malicious node continues to advertise itself as high energy node and keeps getting data from the job initiator. So, it keeps preventing the job initiator from giving preference to any other node in the ad hoc network thus launching DoS attack.

VIDEO OF SIMULATION

The video of simulation is attached with the report.

APPENDIX B: NS2 Source code of the tcl file

```
set opt(chan)          Channel/WirelessChannel
set opt(prop)          Propagation/TwoRayGround
set opt(netif)         Phy/WirelessPhy
set opt(mac)           Mac/802_11                ;# MAC type
set opt(ifq)           Queue/DropTail/PriQueue
set opt(ll)            LL
set opt(ant)           Antenna/OmniAntenna
set opt(x)             800        ;# X dimension of the topography
set opt(y)             800            ;# Y dimension of the topography
set opt(ifqlen)        50             ;# max packet in ifq
set opt(nn)            5              ;# number of nodes
set opt(rp)            AODV       ;# routing protocol script
set opt(lm)            "off"          ;# log movement
set opt(energymodel)    EnergyModel     ;
set opt(radiomodel)     RadioModel     ;
set opt(initialenergy)  1000          ;# Initial energy in Joules
set val(x)    500
set val(y)    500

proc usage {} {
   puts {cbr_mobile: Usage> ns simple_manet.tcl [manet <DSR,AODV,TORA,OLSR,NRLOLSR,others> }
   puts {PARAMETERS NEED NOT BE SPECIFIED... DEFAULTS WILL BE USED}
   exit
}

set state flag
foreach arg $argv {
      switch -- $state {
            flag {
            switch -- $arg {
                  manet   {set state manet}
                  help    {usage}
                  default {error "unknown flag $arg"}
            }
            }
            manet   {set state flag; set val(rp) $arg}
      }
}

set ns_        [new Simulator]

$ns_ use-newtrace
# Create trace file
set tracefd     [open simple_manet.tr w]

# Write into trace file
$ns_ trace-all $tracefd

#Create Nam file
set namtrace [open simple_manet.nam w]

# Write into nam file
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

# Set up topography object
set topo       [new Topography]

$topo load_flatgrid 500 500

# Create God
```

```
create-god $opt(nn)

#Create the specified number of mobilenodes [$val(nn)] and "attach" #them to the channel.
set chan_1_ [new $opt(chan)]
      $ns_ node-config -adhocRouting $opt(rp) \
                       -llType $opt(ll) \
                       -macType $opt(mac) \
                       -ifqType $opt(ifq) \
                       -ifqLen $opt(ifqlen) \
                       -antType $opt(ant) \
                       -propType $opt(prop) \
                       -phyType $opt(netif) \
            -channel $chan_1_ \
                       -topoInstance $topo \
                       -agentTrace ON \
                       -routerTrace ON \
                       -macTrace ON \
                       -energyModel $opt(energymodel) \
                       -idlePower 1.0 \
                       -rxPower 1.0 \
                       -txPower 1.0 \
                       -sleepPower 0.001 \
                       -transitionPower 0.2 \
                       -transitionTime 0.005 \
                       -initialEnergy $opt(initialenergy)


      $ns_ set WirelessNewTrace_ ON

# Configure node using the parameters specified in “define options”
set chan_1_ [new $opt(chan)]

#Define node range and parameters. Nodes too.
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 9.21756e-11 ;#550m
Phy/WirelessPhy set RXThresh_ 4.4613e-10 ;#250m
Phy/WirelessPhy set bandwidth_ 512kb
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 2.4e+9
Phy/WirelessPhy set L_ 1.0
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 0.25
Antenna/OmniAntenna set Gt_ 1
Antenna/OmniAntenna set Gr_ 1
set node_(0) [$ns_ node]
$node_(0) random-motion 0

Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.74269e-10 ;#400m
Phy/WirelessPhy set RXThresh_ 1.08918e-9 ;#160m
Phy/WirelessPhy set bandwidth_ 512kb
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 2.4e+9
Phy/WirelessPhy set L_ 1.0
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 0.25
Antenna/OmniAntenna set Gt_ 1
Antenna/OmniAntenna set Gr_ 1
set node_(1) [$ns_ node]
$node_(1) random-motion 0
```

```
$ns_ node-config -initialEnergy 10
set node_(2) [$ns_ node]
$node_(2) random-motion 0

Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.74269e-10 ;#400m
Phy/WirelessPhy set RXThresh_ 1.08918e-9 ;#160m
Phy/WirelessPhy set bandwidth_ 512kb
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 2.4e+9
Phy/WirelessPhy set L_ 1.0
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 0.25
Antenna/OmniAntenna set Gt_ 1
Antenna/OmniAntenna set Gr_ 1
set node_(2) [$ns_ node]
$node_(2) random-motion 0

Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.74269e-10 ;#400m
Phy/WirelessPhy set RXThresh_ 1.08918e-9 ;#160m
Phy/WirelessPhy set bandwidth_ 512kb
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 2.4e+9
Phy/WirelessPhy set L_ 1.0
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 0.25
Antenna/OmniAntenna set Gt_ 1
Antenna/OmniAntenna set Gr_ 1
set node_(3) [$ns_ node]
$node_(3) random-motion 0

set node_(3) [$ns_ node]
$node_(3) random-motion 0

set node_(4) [$ns_ node]
$node_(4) random-motion 0

# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes
# Here node_(0) is the source.
$node_(0) set X_ 50.0
$node_(0) set Y_ 50.0
$node_(1) set X_ 25.0
$node_(1) set Y_ 50.0
$node_(2) set X_ 75.0
$node_(2) set Y_ 75.0
$node_(3) set X_ 75.0
$node_(3) set Y_ 100.0
# Node_(4) is the data collection point
$node_(4) set X_ 75.0
$node_(4) set Y_ 125.0

#Set Node neighbours
$node_(0) add-neighbor $node_(1)
$node_(0) add-neighbor $node_(2)

#Set destination format is "setdest <x> <y> <speed>"

# node_(0) is the phenominon.
$ns_ at 0.01 "$node_(0) setdest 50.0 150.0 5.0"
$ns_ at 0.01 "$node_(1) setdest 25.0 150.0  5.0"
```

```
$ns_ at 0.01 "$node_(2) setdest 20.0 200.0 0.0"
$ns_ at 0.01 "$node_(3) setdest 10.0 300.0 0.0"
$ns_ at 0.01 "$node_(4) setdest 1.0 300.0 0.0"

#Set Udp, cbr agent and attach those with nodes

set maxenergy 0
set bestworker 0
set neighbors [$node_(0) neighbors]
foreach nb $neighbors {
  set en1 [$nb energy]
  puts $en1
  puts [$nb id]
  if {$en1 > $maxenergy} {
     set maxenergy $en1
          set bestworker $nb
  }
}

set udp1 [new Agent/UDP]
$ns_ attach-agent $node_(0) $udp1
$udp1 set class_ 0

#$udp1 set fid_ 2
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packetSize_ 1000
$cbr1 set interval_ 0.02

# Attach null agent for sink
set null1 [new Agent/Null]
$ns_ attach-agent $bestworker $null1
$ns_ connect $udp1 $null1

set udp2 [new Agent/UDP]
$ns_ attach-agent $node_(2) $udp2
$udp2 set class_ 0

#$udp2 set fid_ 2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
$cbr2 set packetSize_ 1000
$cbr2 set interval_ 0.02

# Attach null agent for sink
set null2 [new Agent/Null]
$ns_ attach-agent $node_(3) $null2
$ns_ connect $udp2 $null2

# Start the traffic generator
$ns_ at 2.0 "$cbr1 start"

# Setting the time to stop the simulation
$ns_ at 160.0 "stop"
$ns_ at 160.01 "puts \"NS EXITING...\" ; $ns_ halt"

puts $maxenergy
puts $bestworker
proc stop {} {
   global ns_ tracefd namtrace
   $ns_ flush-trace
   close $tracefd
   close $namtrace
```

```
    exec nam -a simple_manet.nam &
    exit 0
}

# Run the simulation
$ns_ run
```