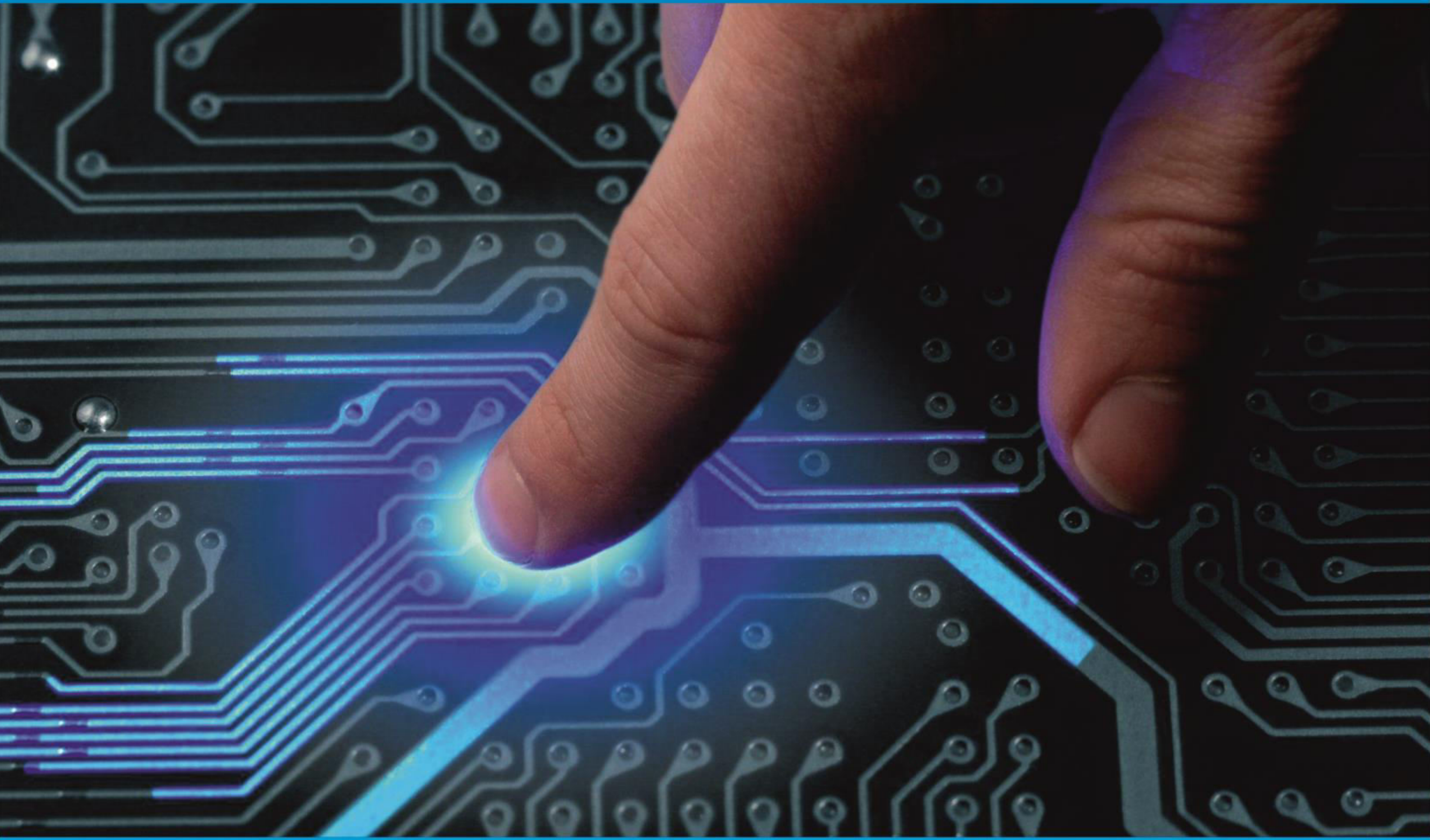
IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 9, Issue 3, March 2021

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.488

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903105|

Review on Parasitic Computing

Arunangshu Biswas [1] Dr. Vaishali Sangvilkar [2]

Student, M.C.A. P.E.S.'s Modern College of Engineering Pune, Maharashtra, India[1]

Head of the Department, M.C.A. P.E.S.'s Modern College of Engineering Pune, Maharashtra, India[2]

ABSTRACT: Reliable communication on the web is guaranteed by a typical set of protocols employed by all computers. The main motive of parasitic computing is to exploit these protocols and compute some problems with the help of communication infrastructure. In this way the internet is transformed into a distributed computer system in which servers unwittingly perform computations on behalf of a remote node. Consequently, Parasitic computing raises important questions on ownership of the resources connected to the web and challenges current computing paradigms. Most simply if we put it, this is a model in which one machine forces the target machine to solve a piece of a complex computational problem merely by engaging them in standard communication. The success of this whole process depends solely on the trust between all involved parties. Like for example solving nondeterministic polynomial-time complete problems a common technique will be to generate a large number of candidate solutions and test the candidate solution for its validity or adequacy. Because the candidate solutions can be tested in parallel and effective computer architecture for these problems is one that supports simultaneous evaluation of many tests.

KEYWORDS: Parasitic Computing, SAT Problem, Target node, hacking, checksum, TCP, HTTP, IP, network, exploit,NP complete problems, NP hard problems

I. INTRODUCTION

In this whole world there are many resources available that are ready to use but aren't being used to their full potential. Trusted communication on the internet is achieved and made sure of by a standard set of protocols used by all computers. These protocols can be exploited healthily for computing or evaluating some problems along with communicating side by side.

Parasitic computing - a programming technique where a program in normal authorized interactions with another program manages to urge the opposite program to perform computations of a fancy nature. It's a type of, a security exploit therein the program implementing the parasitic computing has no authority to consume resources made available to the opposite program.

It was first put forth by Albert Laszlo Barabasi, Vincent Freeh, Hawoong Jeong and Jay Brockman from the Univ. of Notre Dame, Indiana, USA, in the year 2001.[1]

Parasitic Computing is a technology that mainly exploits the transmission control protocol and makes other machines do the computation for its purpose keeping in mind that the machine which does the computation for the parasitic node does not get burdened and crashes.[3] At an equivalent time the target computers are unaware that they performed computation for the advantage of a commanding node. This raises a debate between the security of the mode which becomes the target in parasitic computing and also the benefit that could be achieved by saving a lot of time and resources. Many complex problems like satisfiability problems, NP- Complete problems which could take a long period can be done in a shorter time. The load for computations can be distributed among many target computers located in different geographical regions. Though this may not prove an ultimate solution to all problems, parasitic computing does contribute to meeting some demands in this fast-paced world.

II. LITERATURE SURVEY

Turing Machine - Deterministic/Non-Deterministic

A Turing machine is a theoretical machine that manipulates symbols on a strip of tape consistent with some set of rules. Despite its simplicity, a Turing machine is tailored to simulate the logic of any computer algorithm and is particularly useful in explaining the functions of a CPU inside a computer. In computational complexity discipline, a deterministic Turing machine is a theoretical machine that's utilized in thought experiments to look at the skills and limitations of algorithms. In a deterministic Turing machine , the group of rules impose at the most one action to be performed for any given situation. In a non-deterministic Turing machine , it's going to have a group of rules that prescribes one action for a given situation.

NP, P, NP-Complete Problems [4]

Let P is a set of problems that can be solved by a deterministic Turing or computing machine in Polynomial time.
NP is a set of decision problems that can be solved by a Non-deterministic Turing Machine or a theoretical machine in Polynomial time. P may be a subset of NP (any problem which will be solved by a deterministic machine in polynomial time also can be solved by a non-deterministic machine in polynomial time).
Informally, NP is a collection of decision problems which can be solved in a polynomial-time via a "Lucky Algorithm", a magical algorithm that always makes a right guess among the given set of choices
NP-complete problems are the toughest problems in the NP set. A decision problem T is called NP-complete if:

1. T is in NP (Any given solution for NP-complete problems are often verified quickly, but there's no efficient known solution).
2. Every problem in NP is reducible to T in polynomial-time.

A problem is NP-Hard if it follows property 2 mentioned above, doesn't need to follow property 1. Therefore, the NP-Complete set is additionally a subset of the NP-Hard set.
The above discussion can be supported by the diagram given below.

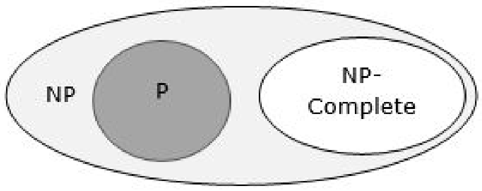


Figure 2.1[4]

For solving many NP - complete problems, like the satisfiability problem, a standard technique is to produce an ample number of possible solutions so test the individual solution for their adequacy. Because the candidate solutions can be tested in parallel, an effective computer architecture for these problems is one that supports simultaneous evaluation of the many tests. Consider the subsequent figure shown below.

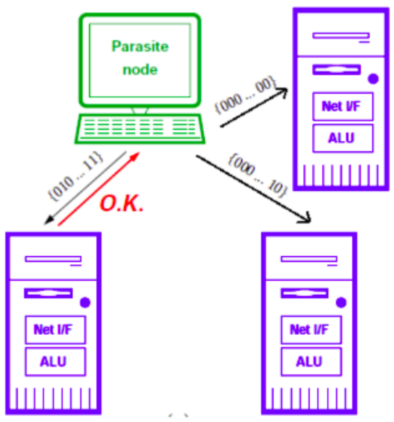


Figure 2.2[3]

In this figure we see that a parasitic computer selects a set of target nodes connected to a network where each of the target nodes consists of

1. Arithmetic Logical Unit

2. Network Interface

So these target nodes are much capable of sending and receiving data and as well as do computations. This will allow one parasitic node to initiate a computational process by directing them some data together with some computational instruction. The target nodes reply to the parasitic node which then tabulates the result for further implementation.

The forms of tests that every of the target nodes can perform, as well as the efficiency with which it can perform them, depends upon the capabilities of the arithmetic and logic unit; hence, the implementation of the parasitic computer should be appropriately matched to the appliance problem.

Since there are many layers involved within the receiving and interpreting message there are several internet protocols that can be exploited to perform a parasitic computation

An I.P. level implementation could force routers to unravel computational problems but this will put the load on network traffic which hinders the sending and receiving data on other nodes which aren't participating in any of the computation processes. That proves to be a bottleneck.

But in the truest sense if one wants to delegate the computations to an overseas target Transmission Control Protocol or may a higher level protocol must be used. Potential candidate protocols include TCP, HTTP, or encryption/decryption with SSL.

III. PARASITIC COMPUTING

3.1 Solving SAT Problem

During package transfer across the Internet, messages can be corrupted, i.e., the bits change. TCP contains a checksum that provides some data integrity of the message. To achieve this, the sender computes a checksum and transmits that with the message. The receiver also computes a checksum, and if it does not agree with the sender's, then the message is corrupted (see Figure 3.1). One property of the TCP checksum function is that it forms a sufficient logical basis for implementing any Boolean logic function, and by extension, any arithmetic operation.

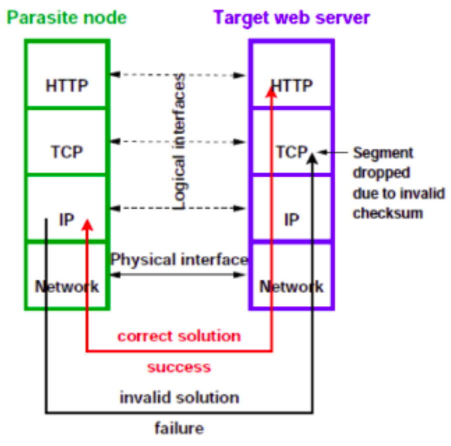


Figure 3.1

3.2 Example

Note: A very trivial problem is taken into consideration in this section as real computation problems are much larger and heavy weight. Main purpose of this example is to cover the basis on which the idea of parasitic computing is supported.

TCP checksum: The checksum is a simple function performed by all web servers (in TCP), which allows a recipient to check if the received message has been corrupted during transmission.[5] The sender (parasitic node) breaks the message and

Let $P(x_1, x_2, x_3, x_4)$ be a function such that

$P(x_1, x_2, x_3, x_4) = (x1 \oplus x2) \wedge (x1 \wedge x2)$

This is a simple equation so it has simple two of these solution which will evaluate the equation to be true that is 1

$x_1 = 1$, $x_2 = 0$, $x_3 = 1$, x_4 as $(1 \oplus 0) \wedge (1 \wedge 1)=1$ or

$x_1 = 0$, $x_2 = 1$, $x_3 = 1$, x_4 as $(0 \oplus 1) \wedge (1 \wedge 1)=1$

Now randomly any solution at a time can satisfy the equation and evaluate the equation as true\\
Let's discuss a problem in detail consider the table shown below:

X	Y	X⊕Y	X∧ Y	X+Y
0	0	0	0	00
0	1	1	0	01
1	0	1	0	01
1	1	0	1	10

Table 3.1

If we notice carefully the result column of X ⊕ Y and X + Y we find a unique mapping of true values of X ⊕ Y and X + Y that is 01 and similarly there exists a unique mapping of true value for X∧ Y and X + Y column that is 10. So we can use this fact for generating unique equations and use the idea of mapping the operators to create a user generated checksum and unique keys such that the sum of the user generated checksum and unique keys is always all 1s. This in turn will help to identify a user key uniquely and discard the non user's key whenever an attempt is made to fire a successful request by any non users. The table below shows the concept and idea discussed above. In the next discussion we will see how to implement this idea and put in use the technique of parasitic computing.

⊕	∧
X1	X3
X2	X4
01	10

Table 3.2

1] The technology is implemented based on the simple protocols used in http request response cycle. If we consider simple request from a client which the server responds to, on the server side server prepares the data first which the client requested and then divides it in some fixed groups of bits for transmitting the data as packets \\

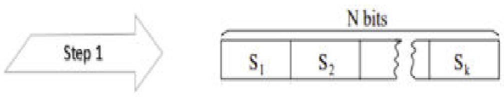


Figure 3.2

2] Next the sub-string of data bits are all summed up together See figure 5 and 6

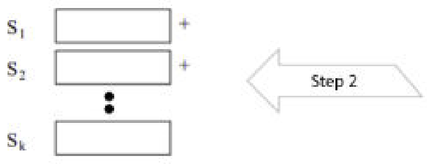


Figure 3.3

3] Next The complement of the result of binary addition is done for further processing

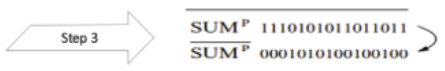


Figure 3.4

4] The complement of the sum is attached at the beginning of the packet as the checksum and sent to the network. The client receiving the response from the server can check the integrity and validity of the data packet sent.

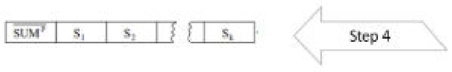


Figure 3.5

The client then add all the data frames sent by the sender and finally adds the total sum to the checksum transmitted at the header of the data, that is
$SUM^T = \underline{(\square\square\square^{\square})} + S_1 + S_2 + ... + S_N$
If all the bits after binary addition turns out to be 1s this means the data received is valid and correct then $SUM^T = 1111111111$

And even if there is at least 1 zero in the end result this means there is a problem or some data bits are lost during transmission then $SUM^T = 1110110111$
Though this is a rare possibility as there is a very good security in transmission control protocol but there may be the odds which may cause problems.

3.3 Real Action

The concept of TCP checksum for integrity of data discussed previously is modified such that , instead of transmitting the checksum evaluated from the sum of data segments, a checksum which all the solution should generate is passed as header along with the data. Thus the T.C.P. at the target side will evaluate the checksum for the whole message received. However, the data segments do not represent the correct solution (which is the entire data), then the checksum evaluated by the target TCP will not give the correct sum (111...11). The Transmission control protocol layer at the target side understands that the message has been damaged, and discards it. Refer Figure 3.3.2[3]

However if the data segments represent one of the correct solution to the satisfiability problem, the message is cleared by TCP and sent to HTTP, web server interprets the solution as an HTTP request; however, because it is not a valid HTTP request, the web server responds with a message saying something like "404 error" or bad request. This event is shown as a red arrow in Figure 3.3..1[3] Thus, every message to which the parasite node receives a response is a solution to the posed SAT problem.

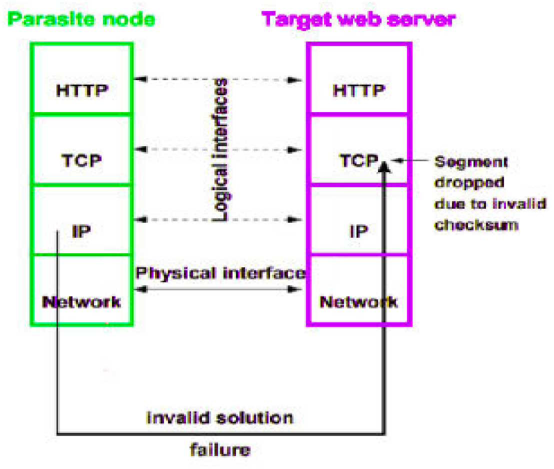


Figure 3.3.1

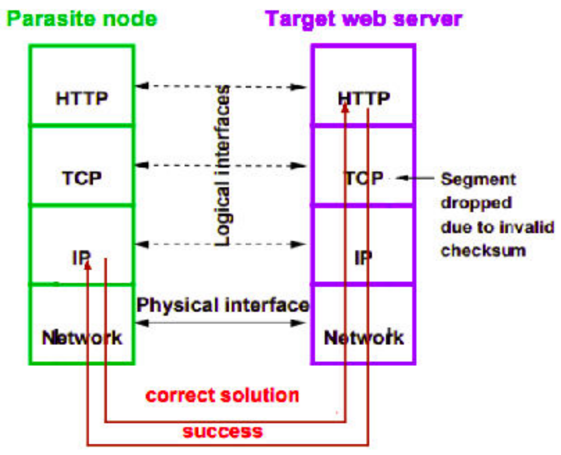


Figure 3.3.2

All this was about the basic working of parasitic computing.

IV. ADVANTAGES AND CONS

As we know no technology can escape challenges which rise due to rapid growth in the competition in this ever increasing demand of human beings, still there are certain advantages that parasitic computing highlights. They are as follows.

4.1 **Advantages**

1. Load for the computation on the parasitic node is reduced exponentially
2. Fake clients requesting service can be denied
3. Computational capability of remote computer is analysed
4. Parasitic computing does not compromise the security of targets, thus it is different than cracking[7]

4.2 **Disadvantages**

1. Parasitic computing could delay the services the target computer normally performs making it similar to denial of service attack
2. Because parasitic computing exploit basic internet protocols it is impos sible to stop a user from launching it
3. And changing or disrupting the functions that are exploited by it would simply eliminate the targets ability to
4. communicate with the rest of the internet.

V. CONCLUSION

To add to the conclusion if we see the example given in the original paper was two computers conversing over the Internet and the first computer is attempting to solve a large complex and very difficult 3-SAT problem by decomposing the original 3-SAT problem into a considerable number of smaller sub problems. Each of these sub-problems is then encoded or say processed as a relation between a checksum and a packet such that whether the checksum is accurate or not is also the answer to that smaller problem. The packet or checksum is then sent to another target computer.

The target computer when receives the packet to check whether the data is well-formed , create a checksum of the packet and see whether it is ditto the same as the provided checksum. If the checksum is not a valid one, it will then request a new packet from the original computer. The original or the parasitic node computer now knows the answer to that smaller problem based on the second computer's response received, and can transmit a fresh packet containing a different sub-problem. Eventually, all the sub-problems will be answered and the final answer will be easily calculated. The proof-of-concept is obviously not much sufficient and also inefficient as the amount of computation necessary to merely send the packets in the first place easily exceeds the computations leached from the other program. The 3-SAT problem would be solved much more quickly if it was just analysed locally by some other machine .Also in practice packets would probably have to be retransmitted occasionally when real checksum errors and network problems occur. However, parasitic computing on the level of checksums is a demonstration of the concept.[1] The authors suggest that as one moves up the application stack, there might come a point where there is a net computational gain to the parasite - perhaps one could break down interesting problems into queries of complex cryptographic protocols using public keys.[2] If there was a total gain, one could theoretically use a number of control nodes or machine for which many hosts on the Internet form a distributed computing network completely unaware

REFERENCES

[1]https://en.wikipedia.org/wiki/Parasitic_computing#:~:text=Parasitic%20computing%20is%20programming%20technique,computations%20of%20a%20complex%20nature.
[2]https://www.academia.edu/37124773/PARASITIC_COMPUTING
[3]Parasitic Computing
Albert-László Barabási , Vincent W. Freeh y , Hawoong Jeong , and Jay B. Brockman y
https://www.researchgate.net/publication/11818463_Parasitic_computing
[4]https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_np_hard_complete_classes.htm
[5]http://webcache.googleusercontent.com/search?q=cache:5hahWiQmgT4J:www.123seminarsonly.com/Seminar-Reports/013/76510610-Parasitic-Computing-Full-Report2.doc+&cd=1&hl=en&ct=clnk&gl=in
[6]https://www.geeksforgeeks.org/tcp-ip-in-computer-networking/
[7]https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.125.6164&rep=rep1&type=pdf#:~:text=Parasitic%20computing%20does%20not%20compromise%20the%20security%20of%20the%20target,denial%2Dof%2Dservice%20attack.
[9]https://www.slideshare.net/aritra6800018/parasitic-computing
[10]https://www.slideshare.net/veenajl/athilyass
[11]https://www.youtube.com/watch?v=GBGS1OQ7Dz0
[12]https://www.youtube.com/watch?v=sz98bXtTISY