Asian Journal of Convergence in Technology *Volume1, Issue 5*
Issn No.:2350-1146, I.F-2.71

P2P Botnet Prevention With The Help of Sybil Attack

1st Avadhoot S. Joshi
Asst. Prof. at Computer Science & Engineering Dept.
VVPIET, Solapur
Solapur, India
joshisavadhoot@rediffmail.com

2nd Sagar M. Mane
Asst. Prof. at Computer Science & Engineering Dept.
VVPIET, Solapur
Solapur, India
sgrmane@gmail.com

3rd Vishal V. Bhanawase
Asst. Prof. at Computer Science & Engineering Dept.
VVPIET, Solapur
Solapur, India
vishal1234_2@yahoo.com

***Abstract*—"Botnet" is a network of computers that are compromised and controlled by an attacker. Botnets are one of the most serious threats to today's Internet. Most current botnets have centralized command and control (C&C) architecture. However, peer-to-peer (P2P) structured botnets have gradually emerged as a new advanced form of botnets. Without central C&C servers, P2P botnets are more resilient to defences and countermeasures than traditional centralized botnets. In this paper, we systematically study P2P botnets along multiple dimensions: bot candidate selection, network construction, C&C mechanisms and communication protocols, and mitigation approaches. We carefully study two defence approaches: index poisoning and Sybil attack. According to the common idea shared by them, we are able to give analytical results to evaluate their performance. We also propose possible counter techniques which might be developed by attackers against index poisoning and Sybil attack defences. In addition, we obtain one interesting finding: compared to traditional centralized botnets, by using index poisoning technique, it is easier to shut down or at least effectively mitigate P2P botnets that adopt existing P2P protocols and rely on file index to disseminate commands.**

Keywords—Botnet; P2P Botnet; Index poisoning; Sybil attack; Kademlia protocol.

I. INTRODUCTION

"Botnet" is a network of compromised computers (bots) running malicious software, usually installed via all kinds of attacking techniques such as Trojan horses, worms and viruses. These zombie computers are remotely controlled by an attacker (botmaster). Botnets with a large number of computers have enormous cumulative bandwidth and computing capability. They are exploited by botmasters for initiating various malicious activities, such as email spam, distributed denial-of service attacks, password cracking and key logging. Botnets have become one of the most significant threats to the Internet.

Today, centralized botnets are still widely used. In a centralized botnet, bots are connected to several servers (called C&C servers) to obtain commands. This architecture is easy to construct and efficient in distributing botmaster's commands; however, it has a weak link - the C&C servers. Shutting down those servers would cause all the bots lose contact with their botmaster. In addition, defenders can easily monitor the botnet by creating a decoy to join a specified C&C channel. Today several P2P botnets have emerged Just like P2P networks, which are resilient to dynamic churn (i.e., peers join and leave the system at high rates), P2P botnet communication won't be disrupted when losing a number of bots. In a P2P botnet, there is no central server, and bots are connected to each other and act as both C&C server and client. P2P botnets have shown advantages over traditional centralized botnets. As the next generation of botnets, they are more robust and difficult for security community to defend.

Researchers have started to pay attention to P2P botnets. However, in order to effectively fight against this new form of botnets, enumerating every individual P2P botnet we have seen in the wild is not enough. Instead, we need to study P2P botnets in a systematic way.

II. P2P BOTNET

In the following, we will discuss how pull and push C& mechanisms can be applied in P2P botnets.

A. Leveraging Existing P2P Protocols:

As we discussed above, both parasite and leeching P2P botnets depend on existing P2P networks. Thus it is natural to leverage the existing P2P protocols used by the host P2P networks for C&C communication. Besides, these protocols have been tested in P2P file-sharing applications for a long time, so they tend to be less error-prone than newly designed ones, and have nice properties to improve performance of P2P systems and mitigate network problems, such as link failure or churn. The following discussion is based on parasite and leeching P2P botnets, but bot-only botnet can adopt these protocols as well.

In P2P file-sharing systems, file index which is used by peers to locate the desired content, may be centralized (e.g., Napster), distributed over a fraction of the file-sharing nodes (e.g., Gnutella), or distributed over all or a large fraction of the nodes (e.g., Overnet). A peer can send out query message for the file it is searching for, and the message will be passed around according to the routing algorithm implemented in the

system. The search will terminate when query hits are returned or the query message expires.

Botmasters can easily adopt the above procedure to disseminate commands in pull style. They can insert records associated with some predefined file titles or hash values into the index, but rather than putting the content location information, botnet commands are attached. In order to get commands issued by botmasters, bots periodically initiate queries for those files or hashes, and nodes who preserve the corresponding records will return query hits with commands encoded. In other words, bots subscribe the content, i.e., commands, published by botmaster.

Take the early version of Storm botnet for example, it utilizes the Overnet, and implements pull C&C mechanism. In this botnet, every day there are 32 hash keys queried by bots to retrieve commands. These 32 keys are calculated by a built-in algorithm, which takes the current date and a random number from [0-31] as input. Therefore, when issuing a command, the botmaster needs to publish it under 32 different keys. Trojan.Peacomm botnet employs the similar C&C design.

Compared to pull mechanism, implementation of push mechanism on existing P2P protocols is more complex. There are two major design issues:

i. Which peers should a bot forward a command to?
ii. How to forward commands: using in-band (normal P2P traffic) or out-of-band messages (non-P2P traffic)?

To address the first issue, the simplest way is to let a bot use its current neighbouring peers as targets. But the problem of this approach is that command distribution may be slow or sometimes disrupted, because 1) some bots have a small number of neighbours, or 2) some peers in a bot's neighbour lists are not bot members in the case of parasite or leeching P2P botnets. One solution to this problem is that letting bots claim they have certain popular files available which are predefined, and forwarding commands to peers appearing in the search results for those files. Thus the chance of commands hitting an actual bot is increased. These predefined popular files behave as the watchwords for the botnet, but could give defenders a clue to identify bots.

For the second issue, whether using in-band or out-of-band message to forward a command depends on what the peers in the target list are. If a bot targets its neighbouring peers, in-band message is a good choice. A bot could encode a command in a query message, which can only be interpreted by bots, send it to all its neighbours, and rely on them to continue passing on the command in the botnet. This scheme is easy to implement and hard for defenders to detect, because there is no difference between command forwarding traffic and normal P2P traffic. On the other hand, if the target list is generated in other ways, like using peers in returned search results discussed above, bots have to contact those peers using out-of-band message. Obviously out-of-band traffic are easier to detect, and hence, can disclose the identities of bots who initiate such traffic.

The above discussion mainly focused on unstructured P2P networks, where query messages are flooded to the network. In structured P2P networks (e.g., Overnet), a query message is routed to the nodes whose node IDs are closer to the queried hash key, which means queries for the same hash key are always forwarded by the same set of nodes. Therefore, to let more bots receive a command, the command should be associated with different hash keys, such that it can be sent to different parts of the network.

B. Design A New P2P Communication Protocol

It is convenient to adopt existing P2P protocols for P2P botnet C&C communication, however, the inherited drawbacks may limit botnet design and performance. A botnet can be more flexible if it uses a new protocol designed by its botmaster.

The advanced hybrid P2P botnet and the super botnet are two newly designed P2P botnets, whose C&C communication are not dependent on existing P2P protocols. Both of them implements push and pull C&C mechanisms. In a hybrid P2P botnet, when a bot receives a command, it forwards the command to all the peers in the list (push), and those who cannot accept connection from others periodically contacts other bots in the list and try to retrieve new commands (pull). A super botnet is composed of a number of small centralized botnets. Commands are pushed from one small botnet to another, and within a small centralized botnet, bots pull the command from their C&C servers. Furthermore, the hybrid P2P botnet is able to effectively avoid bootstrap procedure, which is required by most of the existing P2P protocols, by 1) passing a peer list from one bot to a host that is infected by this bot, and 2) exchanging peer lists when two bots communicate.

The drawback of designing a new protocol for P2P botnet communication is that the new protocol has never been tested before. When a botnet using this protocol is deployed, the network may not be as stable and robust as expected due to complex network conditions and defences [9].

C. Functionalities of P2P Botnet

In this section we detail two key functionalities of P2P bot: C&C functionality and P2P functionality. And then, several features are proposed to describe P2P bot.

[1]. Functionalities of P2P Bot
i. Command-and-control Functionality

The defining characteristic of bots is the remote control mechanism, by which we can distinguish bots from conventional viruses and worms. And we usually call it "command and control", C&C for short. Command-and control functionality enables bots to request, send, interpret commands and return results.

- Request: a bot asks another bot for command information. The bot has to know whom it should request, and we call the destination "predecessor".

- Send: a bot send command information to another bot. The bot has to know to whom it should send command information, and we call the destination "successor".
- Interpret: command information is interpreted as concrete instructions which bots can execute.
- Return: a bot sends execution results to its controller.

ii. Peer-to-peer Functionality

The defining characteristic of P2P botnets is the peer-to peer communication style. Due to this communication style, P2P botnets are more resilient than centralized botnets.

Peer-to-peer functionality enables bots to construct and maintenance an overlay network, to route and locate, and to deal with joining, leaving and failure of peers. Generally P2P communication in P2P botnets is similar to that in common P2P applications, file-sharing systems for example.

[2]. Features of P2P Bot

Different functionalities correspond to different services for users. Features of a service consist of attributes including protocol, version and patch, and configurations including programs and parameters.

i. Command-and-control Features

C&C attributes consist of C&C-protocol, C&C-version and C&C-patch. C&C configurations consist of C&C programs and parameters. The later includes predecessor-table, successor-table, commands, and requests.

- Predecessor-table contains all predecessors a bot knows.
- Successor-table contains all successors a bot knows.
- Commands represent the set of command information.
- Requests represent the set of request information.

ii. Peer-to-peer Features

P2P attributes consist of P2P -protocol, P2P -version and P2P -patch. P2P configurations consist of P2P programs and parameters. The later includes boots-table and routing-table.

- Boots-table contains all boots-peers a peer knows. A peer joins in a P2P network by at least one peer existing in the network, i.e. boots-peer. The content of boots-table may vary from P2P protocol to P2P protocol.
- Routing-table contains several nearest peers' information. A peer asks these peers to route queries and to locate resources. The same as boots-table, the content of boots-table may be different due to different P2P protocols.

III. KADEMLIA PROTOCOL

3.1. Introduction

Kademlia is a distributed hash table (DHT) [21] for decentralized peer-to-peer computer networks designed by Petar Maymounkov and David Mazières in 2002. It specifies the structure of the network and the exchange of information through node lookups. Kademlia nodes communicate among themselves using UDP. A virtual or overlay network is formed by the participant nodes. Each node is identified by a number or *node ID*. The *node ID* serves not only as identification, but the Kademlia algorithm uses the *node ID* to locate values (usually file hashes or keywords). In fact, the *node ID* provides a direct map to file hashes and that node stores information on where to obtain the file or resource.

When searching for some value, the algorithm needs to know the associated key and explores the network in several steps. Each step will find nodes that are closer to the key until the contacted node returns the value or no more closer nodes are found. This is very efficient: Like many other DHTs, Kademlia contacts only O(logn) nodes during the search out of a total of $n$nodes in the system.

Further advantages are found particularly in the decentralized structure, which increases the resistance against a denial of service attack. Even if a whole set of nodes is flooded, this will have limited effect on network availability, since the network will recover itself by knitting the network around these "holes".

Kademlia is a communications protocol for peer-to-peer networks. It is one of many versions of a DHT, a Distributed Hash Table.

3.2. The Node

A Kademlia network consists of a number of cooperating nodes that communicate with one another and store information for one another. Each node has a nodeID, a quasi-unique binary number that identifies it in the network.

Within the network, a block of data, a value, can also be associated with a binary number of the same fixed length B, the value's key.

A node needing a value searches for it at the nodes it considers closest to the key. A node needing to save a value stores it at the nodes it considers closest to the key associated with the

3.2.1. NodeID

NodeIDs are binary numbers of length B = 160 bits. In basic Kademlia, each node chooses its own ID by some unspecified quasi-random procedure. It is important that nodeIDs be uniformly distributed; the network design relies upon this.

While the protocol does not mandate this, there are possible advantages to the node's using the same nodeID whenever it joins the network, rather than generating a new, session-specific nodeID.

3.2.2. Keys

Data being stored in or retrieved from a Kademlia network must also have a key of length B. These keys should also be uniformly distributed. There are several ways to guarantee this; the most common is to take a hash, such as the 160 bit SHA1 digest, of the value.

3.2.3. Distance: the Kademlia Metric

Kademlia's operations are based upon the use of exclusive OR, XOR, as a metric. The distance between any two keys or nodeIDs x and y is defined as

distance(x, y) = x ^ y

Where ^ represents the XOR operator. The result is obtained by taking the byte wise exclusive OR of each byte of the operands.

Kademlia follows Pastry in interpreting keys (including nodeIDs) as bigendian numbers. This means that the low order byte in the byte array representing the key is the most significant byte and so if two keys are close together then the low order bytes in the distance array will be zero.

3.2.4. The K-Bucket

A Kademlia node organizes its contacts, other nodes known to it, in buckets which hold a maximum of k contacts. These are known as k-buckets.

The buckets are organized by the distance between the node and the contacts in the bucket. Specifically, for bucket j, where 0 <= j < k, we are guaranteed that

2^j <= distance(node, contact) < 2^(j+1)

Given the very large address space, this means that bucket zero has only one possible member, the key which differs from the nodeID only in the high order bit, and for all practical purposes is never populated, except perhaps in testing. On other hand, if nodeIDs are evenly distributed, it is very likely that half of all nodes will lie in the range of bucket B-1 = 159.

3.2.4.1. Bucket Size

The Kademlia paper says that k is set to a value such that it is very unlikely that in a large network all contacts in any one bucket will have disappeared within an hour. Anyone attempting to calculate this probability should take into consideration policies that lead to long-lived contacts being kept in the table in preference to more recent contacts.

3.2.4.2. Contacts

A contact is at least a triple:

- the bigendian nodeID for the other node
- its IP address
- its UDP port address

The IP address and port address should also be treated as bigendian numbers. Kademlia's designers do not appear to have taken into consideration the use of IPv6 addresses or TCP/IP instead of UDP or the possibility of a Kademlia node having multiple IP addresses.

3.2.4.3. Sorting

Within buckets contacts are sorted by the time of the most recent communication, with those which have most recently communicated at the end of the list and those which have least recently communicated at the front, regardless of whether the node or the contact initiated the sequence of messages.

3.2.4.4. Updates

Whenever a node receives a communication from another, it updates the corresponding bucket. If the contact already exists, it is moved to the end of the bucket. Otherwise, if the bucket is not full, the new contact is added at the end. If the bucket is full, the node pings the contact at the head of the bucket's list. If that least recently seen contact fails to respond in an *(unspecified)* reasonable time, it is dropped from the list, and the new contact is added at the tail. Otherwise the new contact is ignored for bucket updating purposes.

In a large, busy network, it is possible that while a node is waiting for a reply from the contact at the head of the list there will be another communication from a contact not in the bucket. This is most likely for bucket B-1 = 159, which is responsible for roughly half of the nodes in the network. Behaviour in this case is unspecified and seems likely to provide an opening for a DOS (Denial of Service) attack.

3.2.4.5. Rationale

Experience has shown that nodes tend to group into two clearly distinguished categories, the transient and the long-lived. This update policy gives strong preference to the long-lived and so promotes network stability. It also provides a degree of protection from certain types of denial of service (DOS) attacks, including, possibly, Sybil attacks, discussed below [20].

3.3. Protocol Messages

The original Kademlia paper, says that the Kademlia protocol consists of four remote procedure calls ("RPCs") but then goes on to specify procedures that must be followed in executing these as well as certain other protocols. It seems best to add these procedures and other protocols to what we call here the Kademlia protocol.

Kademlia has four messages.

- PING — used to verify that a node is still alive.
- STORE — stores a (key, value) pair in one node.
- FIND_NODE — the recipient of the request will return the k nodes in his own buckets that are the closest ones to the requested key.
- FIND_VALUE — Same as FIND_NODE, but if the recipient of the request has the requested key in its store, it will return the corresponding value.

Each RPC message includes a random value from the initiator. This ensures that when the response is received it corresponds to the request previously sent.

3.3.1. PING

This RPC involves one node sending a PING message to another, which presumably replies with a PONG.

This has a two-fold effect: the recipient of the PING must update the bucket corresponding to the sender; and, if there is a reply, the sender must update the bucket appropriate to the recipient.

All RPC packets are required to carry an RPC identifier assigned by the sender and echoed in the reply. This is a quasi-random number of length B (160 bits).

Implementations using shorter message identifiers must consider the birthday paradox, which in effect makes the probability of a collision depend upon half the number of bits in the identifier. For example, a 32-bit RPC identifier would yield a probability of collision proportional to 2^-16, an uncomfortably small number in a busy network. If the identifiers are initialized to zero or are generated by the same random number generator with the same seed, the probability will be very high indeed.

It must be possible to piggyback PINGs onto RPC replies to force or permit the originator, the sender of the RPC, to provide additional information to its recipient. This might be a different IP address or a preferred protocol for future communications.

3.3.2. STORE

The sender of the STORE RPC provides a key and a block of data and requires that the recipient store the data and make it available for later retrieval by that key. This is a primitive operation, not an iterative one.

While this is not formally specified, it is clear that the initial STORE message must contain in addition to the message ID at least the data to be stored (including its length) and the associated key. As the transport may be UDP, the message needs to also contain at least the nodeID of the sender, and the reply the nodeID of the recipient. The reply to any RPC should also contain an indication of the result of the operation. For example, in a STORE while no maximum data length has been specified, it is clearly possible that the receiver might not be able to store the data, either because of lack of space or because of an I/O error.

3.3.3. FIND_NODE

The FIND_NODE RPC includes a 160-bit key. The recipient of the RPC returns up to k triples (IP address, port, nodeID) for the contacts that it knows to be closest to the key.

The recipient must return k triples if at all possible. It may only return fewer than k if it is returning all of the contacts that it has knowledge of. This is a primitive operation, not an iterative one.

The name of this RPC is misleading. Even if the key to the RPC is the nodeID of an existing contact or indeed if it is the nodeID of the recipient itself, the recipient is still required to return k triples. A more descriptive name would be FIND_CLOSE_NODES.

The recipient of a FIND_NODE should never return a triple containing the nodeID of the requestor. If the requestor does receive such a triple, it should discard it. A node must never put its own nodeID into a bucket as a contact.

3.3.4. FIND_VALUE

A FIND_VALUE RPC includes a B=160-bit key. If a corresponding value is present on the recipient, the associated data is returned. Otherwise the RPC is equivalent to a FIND_NODE and a set of k triples is returned. This is a primitive operation, not an iterative one [19] [20].

3.4. Possible Problems with Kademlia: The Sybil Attack

A paper by John Douceur, describes a network attack in which attackers select nodeIDs whose values enable them to position themselves in the network in patterns optimal for disrupting operations. For example, to remove a data item from the network, attackers might cluster around its key, accept any attempts to store the key/value pair, but never return the value when presented with the key.

A Sybil variation is the Spartacus attack, where an attacker joins the network claiming to have the same nodeID as another member. As specified, Kademlia has no defence. In particular, a long-lived node can always steal a short-lived node's nodeID.

Douceur's solution is a requirement that all nodes get their nodeIDs from a central server which is responsible at least for making sure that the distribution of nodeIDs is even.

A weaker solution would be to require that nodeIDs be derived from the node's network address or some other quasi-unique value [20].

Acknowledgment

With all respect and gratitude, I would like to thank all people who have helped me directly or indirectly for the completion of this paper.

I express our heartily gratitude towards Mr. M. S. Chaudhary for guiding me to understand the work conceptually and also for his constant encouragement to complete this paper on "P2P Botnet Prevention with the Help of Sybil Attack"

I also express our thanks to Prof. T. J. Parvat Head of department of Computer Engineering & PG Co-ordinator Prof. M. S. Chaudhary for providing necessary information and required resources.

With deep sense of gratitude I thank to our Principal Dr. M. S. Gaikwad and Management of the SIT, Lonavala for providing all necessary facilities and their constant encouragement and support.

Last but not the least we thanks to all the Teaching & Non-teaching staff members of Computer Engineering Department for providing necessary information and required resources.

I am ending this acknowledgement with deep indebtedness to my friends who have helped me.

References

[1] Abhijeet B. Potey, Prof.Anjali B.Raut "Defending Sybil Using Social Network", International Journal of Engineering and Computer Science (IJECS) Volume 2 Issue, Page No. 196-199, 2 Feb 2013.

[2] "Botnets - the evolving nature of adversaries, tactics, techniques and procedures" Georgia Tech Cyber Security Summit, 2011, Pages 6-7.

[3] Joseph Massi, Sudhir Panda, Girish Rajappa, Senthil Selvaraj and Swapana Revankar "Botnet Detection and Mitigation" Proceedings of Student-Faculty Research Day, Pace University, 2010.

[4] Andrew White, Alan Tickle, and Andrew Clark "Overcoming Reputation and Proof-of-Work Systems in Botnets" Fourth international Conference on Network and System Security, 2010.

[5] Zhou Hangxia "Mitigating Peer-to-Peer Botnets by Sybil attacks", International Conference on Innovative Computing and Communication and Asia-Pacific Conference on Information Technology and Ocean engineering © IEEE, 2010.

[6] Oliver Jetter, Jochen Dinger, and Hannes Hartenstein "Quantitative Analysis of the Sybil Attack and Effective Sybil Resistance in Peer-to-Peer Systems", IEEE ICC proceedings, 2010.

[7] Junfeng Duan, Jian Jiao, Chunhe Xia, Shan Yao, and Xiaojian Li "Descriptive Model of Peer-to-Peer Botnet Structures", International Conference on Educational and Information Technology (lCEIT), 2010.

[8] Ping Wang, Sherri Sparks, and Cliff C. Zou "An Advanced Hybrid Peer-to-Peer Botnet", IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 7, NO. 2, Pages 113-127, APRIL-JUNE 2010.

[9] Ping Wang, Lei Wu, Baber Aslam and Cliff C. Zou "A Systematic Study on Peer-to-Peer Botnets" IEEE, 2009.

[10] Carlton R. Davis, Jos´e M. Fernandez, and Stephen Neville "Optimising Sybil Attacks against P2P-based Botnets", 2009.

[11] Ping Wang, Lei Wu, Baber Aslam and Cliff C. Zou "A Systematic Study on Peer-to-Peer Botnets" IEEE, 2009.

[12] Duc T. Ha, Guanhua Yan, Stephan Eidenbenz, and Hung Q. Ngo "On the Effectiveness of Structural Detection and Defence Against P2P-based Botnets", 2009.

[13] Thibault Cholez, Isabelle Chrisment and Olivier Festor "Evaluation of Sybil Attacks Protection Schemes in KAD", published in 3rd International Conference on Autonomous Infrastructure, Management and Security – AIMS, 2009.

[14] Robert F. Erbacher, Adele Cutler, Pranab Banerjee, and Jim Marshall "A Multi-Layered Approach to Botnet Detection", 2008.

[15] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling "Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm", 2008.

[16] Reinier Schoof and Ralph Koning "Detecting peer-to-peer botnets", 2007.

[17] Antti Nummipuro "Detecting P2P-Controlled Bots on the Host" Seminar on Network security, 2007.

[18] Daniel Stutzbach and Reza Rejaie "Improving Lookup Performance over a Widely-Deployed DHT", infocom, 2006.

[19] "Kademlia: A Design Specification", the XLattice Project, 2003-2006.

[20] John R. Douceur "The Sybil Attack", in the proceeding of first international workshop on peer-to-peer systems (IPTPS), Pages 251-256, 2002.

[21] Petar Maymounkov and David Mazi`eres "Kademlia: A Peer-to-peer Information System Based on the XOR Metric".

[22] Benedikt Westermann, Andriy Panchenko, and Lexi Pimenidis "A Kademlia-based Node Lookup System for Anonymization Networks".

[23] Isabel Pita and Adrian Riesco "Specifying and Analysing the Kademlia Protocol in Maude*".

[24] Manoj Rameshchandra Thakur "Distributed and Cooperative Approach to Botnet Detection Using Gossip Protocol".

[25] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack "Exploiting KAD: Possible Uses and Misuses".

[26] M. Patrick Collins, Timothy J. Shimeall, Sidney Faber, Jeff Janies, Rhiannon Weaver, and Markus De Shon "Using uncleanliness to predict future botnet addresses".

[27] Kademlia - Wikipedia, the free encyclopedia.

[28] Distributed hash table - Wikipedia, the free encyclopedia.