Verme: Worm Containment in Peer-to-Peer Overlays

Filipe Freitas* Rodrigo Rodrigues* Carlos Ribeiro* Paulo Ferreira* Luís Rodrigues†
*INESC-ID / Technical University of Lisbon †University of Lisbon

Abstract

Peer-to-peer overlays provide an ideal substrate for worm propagation. P2p-assisted worms have the potential to spread faster than traditional scanning worms because they have knowledge of a subset of the overlay nodes, and choose these nodes to propagate themselves; and also because they can avoid traditional detection mechanisms.

We present a novel approach for containing p2p-assisted worms based on the fact that some overlay nodes may not have common vulnerabilities, due to their platform diversity. By properly reorganizing the overlay graph, this can lead to the containment of p2p-assisted worms in small islands of nodes with common vulnerabilities that only have knowledge of themselves or nodes running on distinct platforms.

We present the design of Verme, a p2p overlay based on Chord that follows this approach, and we discuss several interesting issues that arise in Verme's design. We argue that this new overlay may help containing, or at least slowing down the propagation of p2p-assisted worms, and raise the difficulty level of writing them.

1 Introduction

In recent years, we have witnessed the outbreak of several Internet worms that have not only caused inconvenience to many users, but also a large societal impact. Most of these are instances of "scanning worms" [13]. This means that once the worm has infected a host, it propagates itself by probing random IP addresses for new nodes to infect. In some cases this choice of IP addresses is biased by some heuristic that increases the chances of finding a IP address that is in use.

Recently, some authors have pointed out the potential problems raised by peer-to-peer assisted worms [18]. Such worms could take advantage of the topological information maintained by the p2p overlay by choosing its overlay neighbors as the next target to infect.

We must consider the possibility of two types of p2p-assisted worms. We can either have a worm that exploits a vulnerability in the p2p client application, or a worm that is not related to the p2p application, but uses knowledge from this application (e.g., by inspecting open TCP connections) to choose where to propagate. In this paper we consider the latter case, since it is more general in the sense that our defenses will also work for the former type.

P2p-assisted worms are potentially a more serious threat than scanning worms, since they can (1) spread faster as they do not have to probe random IP addresses, most of which are unused, (2) avoid traditional detection mechanisms, which are based on anomalous IP traffic patterns [10, 12] (as they do not generate many failed connections and can disguise as normal p2p traffic), and (3) avoid being detected by honeypots [10] (surveillance machines for early warning and detection that listen in unused IP addresses).

Although we have not seen specific instances of fast-spreading p2p-assisted worms, there is some indication that this is a pending problem. For instance, there have been reports of vulnerabilities in p2p client applications like eDonkey and KaZaA that would allow for the execution of arbitrary code on the client [1, 2]. Also, there have been some instances of viruses that use file sharing overlays to assist in their propagation by making themselves available for download [3]. Thus we could argue that it may be serendipity that worm authors have not written a fast-spreading p2p-assisted worm.

In previous work, researchers have pointed out the existence of this problem [18], and even quantified how much faster p2p-worms can propagate using simulations [5] and analysis [17]. In this paper we take the next step of proposing that peer-to-peer overlays should be modified to incorporate defenses that contain or at least slow down the propagation of p2p-assisted worms.

We present a series of general principles that should guide the design of overlays to achieve the aforementioned goals. Then we present a new overlay called Verme that is designed with these principles in mind. Verme is an extension of Chord [14], designed to contain p2p-assisted worms in small "islands" of nodes that may have common vulnerabilities. We designed Verme such that nodes inside each island do not have knowledge of other nodes with common vulnerabilities. As a consequence, the worm can be contained within the island. Furthermore, Verme is designed to maintain the

good properties of Chord, namely its good lookup performance and low overhead.

In the design of Verme a series of interesting problems have arisen, like how to address Sybil attacks [8] (in this case a Sybil attacker could join the overlay with identities of the wrong platform type, and use them to obtain addresses of nodes it should not have access to), and how to address load-balancing issues that arise from the uneven distribution of platforms that nodes run on. In this paper we also discuss possible ways to address these problems.

While not claiming to have found a panacea, our new insights and overlay design may contribute to containing, or at least slowing down the propagation of p2p-assisted worms, and raising the difficulty level for writing them.

2 Design Principles

We propose that the design of peer-to-peer overlays should be guided by the need to incorporate defenses that limit the propagation of p2p-assisted worms. The main insight behind our proposal is that overlays contain many different types of nodes, running on different platforms, or using different versions of p2p client software. This diversity can be used to contain the propagation of p2p-assisted worms, since the vulnerabilities in one particular implementation or platform may not affect the entire population. For instance, the SQL-Slammer worm only affects Windows machines running SQLServer 2000 applications. On the other hand the Linux/Lupper worm spreads by exploiting web servers hosting vulnerable PHP/CGI scripts, but it targets only i386 machines running Linux because it is distributed as an i386 ELF program. In case the vulnerability is found in the p2p client applications it is also not likely to be present in different client implementations.

Therefore we propose that existing overlay designs must be modified such that the overlay graph (i.e., the graph formed by the overlay routing state) forms small "islands" of nodes that are running on the same platform (or otherwise have common vulnerabilities). The nodes in each island may be adjacent to other nodes from the same island, or to nodes from islands of distinct types (i.e., that do not have common vulnerabilities), but may not be adjacent to nodes from other islands of the same type.

Figure 1 gives an example of a system with two types of nodes that do not have common vulnerabilities. The overlay graph forms small islands of nodes of the same type (enclosed within the dashed circles). The nodes within an island may have edges among themselves

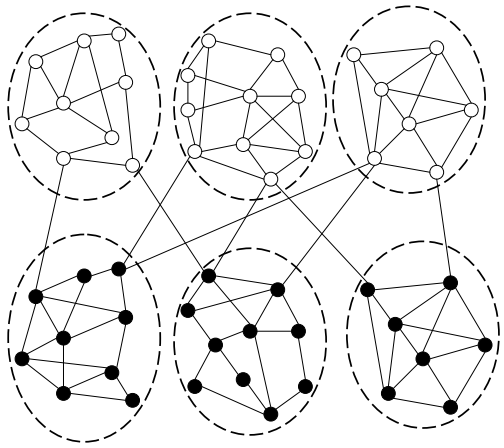

Figure 1: Generic structure of an overlay graph designed for worm containment

(i.e., they may be present in each other's routing tables) which may lead to the propagation of a p2p-assisted worm within an island. Nodes may also have edges to nodes that belong to distinct islands of other types, but never to nodes of distinct islands of the same type. Therefore a p2p-assisted worm will be confined to an island.

Modifying the overlay graph is not enough to succeed in preventing the propagation of p2p-assisted worms. For instance, a worm could use overlay maintenance messages or perform lookups to discover the network addresses of nodes of the same type from distinct islands. In the following sections we discuss other modifications to the peer-to-peer protocols and applications that are necessary.

3 Verme Design

In this section we present the design of Verme, an extension of Chord [14] that follows the design principles presented above.

In this presentation we rely on some assumptions that we will revisit in subsequent sections to discuss how reasonable they are or how they can be enforced.

First, we assume that each node is assigned a certificate that binds its node identifier to the public key that speaks for its principal, type, and possibly (depending on the level of security that is provided, as we will discuss next) its IP address.

To simplify our presentation, we will assume that nodes may be of two distinct types without common vulnerabilities (generalizing our design to support more than two types of nodes without common vulnerabilities is relatively straightforward). Furthermore we assume that nodes are evenly distributed among the two types.

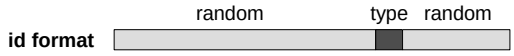

Figure 2: Identifier structure in Verme

In Section 5 we discuss how to mitigate the negative effects of uneven type distributions.

3.1 Chord overview

Chord [14] is a peer-to-peer routing overlay that provides a scalable lookup primitive that allows applications to find data stored in a peer-to-peer system.

In Chord nodes have identifiers that are 160-bit integers assigned in such a way that they are uniformly distributed (e.g., as the output of a SHA-1 function applied to the network address and port number of the node).

Chord designates the n nodes whose identifiers immediately follow a key (called the successor nodes) as responsible for that key. Lookups map a 160-bit key (the identifier of the data item) to the list of successors of that key.

Each Chord node maintains a small amount of routing state (small enough to keep its maintenance overhead low). This consists of a list of successors (i.e., the ids and IP addresses of the nodes that follow it in the ring) and a finger table, consisting of the IP addresses and identifiers of nodes that follow it at power-of-two distances in the identifier space.

Lookups requests travel through a sequence of nodes (either iteratively or recursively), where each node in this sequence answers or forwards the request to the node from its finger table with highest id still smaller than the desired key. The lookup will conclude when the successor of the id is reached, which happens with high probability after $O(log\ N)$ routing hops.

3.2 Id Assignment

The id assignment scheme used by Chord does not obey the principles mentioned in Section 2, since the list of successors of any given node will typically contain nodes of both types. Therefore we modify the way ids are assigned such that the ring is divided into sections, where each section only contains the ids of nodes of a particular type. Furthermore, neighboring sections must always belong to different types. This will cause nodes of the same type from the same section to have knowledge about themselves (through their successor lists) but no knowledge of nodes of the same type in other sections (provided that the number of nodes in each section is large enough that successor lists never cross more than one section).

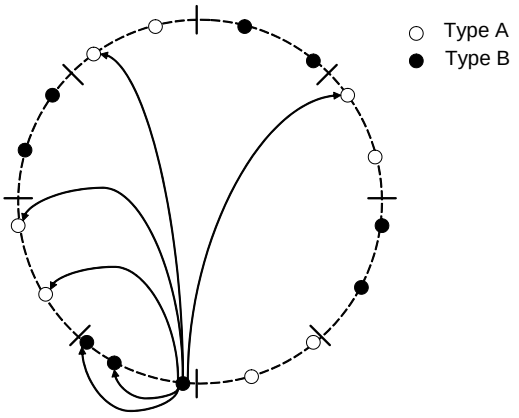

Figure 3: Finger and successor pointers in Verme

Verme's id assignment achieves this by dividing the node id in three parts, as depicted in Figure 2. The lower bits are assigned randomly, and the number of bits employed here specifies the length of the section. By adjusting this number properly we can ensure that, with high probability, successor lists do not cross more than one section. The middle bits are fixed according to the node type. With our simplifying assumption of having only two types in the system we could use a single bit. The higher bits are also assigned randomly and they specify the section number that the node is in.

3.3 Successors and Fingers

Each node maintains pointers to a successor list that are used and maintained just like in Chord. However, finger table entries must be modified to point to a node that is not of the same type as the node itself, to respect the design principles presented in Section 2.

Thus we need to change the way that fingers are defined. Instead of a finger table entry pointing to the node that follow it at power-of-two distances in the identifier space, the finger entry will now correspond to the first successor of the ids at the same distance that belongs to the opposite type. Figure 3 shows a Verme ring with the successors and fingers of a node.

3.4 Lookups

Lookup is the crucial abstraction provided by the routing overlay. In Chord (as in most peer-to-peer overlays) any node in the system can issue a *lookup(id)* operation. As mentioned, this returns the address of a node (or set of nodes) that are responsible for the data with that id (in this case these are the successors of the id).

This is used not only by applications, but also in the overlay maintenance protocols: finger table entries are

refreshed periodically by performing a lookup to the appropriate point in the id space; and joins of nodes incoming to the overlay are also initiated by performing a lookup to the id of the incoming node, who then contacts its new successor to update its routing information. We will begin by discussing how lookups are modified for overlay maintenance operations, and we discuss how applications can use lookups in Section 4.

Lookup is an operation that poses high risk for worm propagation. The current abstraction allows a worm to crawl the overlay, by making lookups with different ids, to obtain addresses to attack. We address this issue by changing several aspects of the lookup operation. First, the lookup message must carry the certificate of the node that is performing the lookup. This will allow the successor of the id to verify the legitimacy of the initiator in looking up this id. When lookups are being used for joining the overlay or calculating finger table entries, this is straightforward: the node must verify if it is the successor or a correct finger of the id in the certificate.

The second aspect we need to address in lookups is that they cannot be iterative, since many nodes in a lookup path have the same type as the node performing the lookup. Therefore we change the lookup to be either recursive (i.e., the reply travels back through the reverse lookup path) or transitive (i.e., the forward path is identical to a recursive lookup, but the replier contacts the initiator directly). If the lookup is recursive, the reply must be encrypted with the public key of the initiator (present in the certificate sent with the lookup) to keep the IP address in the reply from being disclosed to the nodes in the lookup path.

Transitive lookups are more efficient, but in this case the certificate sent in the lookup must contain the IP address of the initiator node, to allow the replier to contact him. This will open an avenue for an infected node to collect a large number of IP addresses of any type, simply by inspecting the IP addresses in certificates that are sent through it. Since overlay maintenance messages are small and relatively infrequent, we made the choice of using recursive lookups for this kind of operations. We discuss lookups performed by applications next.

4 Upper Layers

The layers above the lookup substrate also need to be modified to preserve the design principles and properties subjacent to our scheme, like not propagating network information about peers.

In this section we will focus on a particular layer that uses the lookup infrastructure: a distributed hash table (DHT) that supports get and put operations, similar to the DHash layer built on top of Chord lookups [7]. We believe this is representative of how other applications can be adapted.

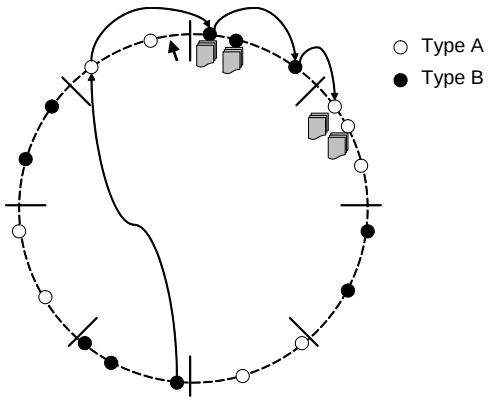

Figure 4: Replication of data items in a DHT

DHash uses lookups to get or put data in the overlay. In this system data is replicated in the set of n successors of the identifier of the data item. Get and put operations are preceded by a lookup that returns the address of one or more nodes responsible for the data. Then these nodes are contacted directly to store or retrieve the data.

This poses a risk to worm propagation because the responsible node may be from the same type of the node making the request. The first step to address this problem is to change the way that replicas are assigned. Instead of replicating in the n successors of the identifier of the data item, we make $n/2$ replicas at that point in the id space, and another $n/2$ in the same position of the subsequent section of the ring (of the opposite type).

This design is depicted in Figure 4. In this case the lookup will only return the addresses of the replicas that are of the opposite type of the node that issued the request. Then these nodes can be contacted directly to perform the *get* operation. In the case of puts, the client node should not contact directly the replicas of its own type. Instead, it contacts the remaining replicas, which are then responsible for replicating the data on the replicas of same type as the client.

This design feature also has the advantage of increasing reliability, since a worm outbreak that affects nodes from one type will not be able to wipe out all copies of a given object. We intend to further explore the increased availability and reliability guarantees of our system.

In the next section we discuss some implications of this design when the assumptions we stated previously (namely when nodes may obtain many or incorrect certificates) are not met.

5 Discussion

In this section we question the validity some of the assumptions made previously, and discuss other issues that remain open.

5.1 Sybil Attacks

In our design we assumed that nodes had certificates containing a correct indication of the type of node.

Issuing such certificates and limiting Sybil attacks [8] are issues that have been solved with some degree of success in deployed systems like Credence [15] (by asking joining nodes to download a large file or solve cryptographic puzzles). Also, in some cases where the client hardware allows it, we can use remote attestation to verify the identity and platform where the client is running.

However, a more problematic type of impersonation would happen if even a single bad node would obtain a certificate for a type that is different from the vulnerable machines (even, if needed, by acquiring a machine of that type), and perform lookups to several ids to collect a series of IP addresses of the intended type. For instance, a single node pretending to be running Linux could obtain many IP addresses of Windows machines by joining the overlay and looking up ids at random.

Our take on this problem is that we cannot solve it entirely, as there always may be a small fraction of impersonating nodes in the system, so we should minimize the damage caused by these nodes.

The impact of this attack can be limited if we only use recursive operations (i.e., an operation request is routed recursively like a lookup, and the reply travels back through the reverse lookup path), precluding lookups that return an IP address. This way, a node's knowledge of nodes of the opposite type is limited to its successors, fingers, and the nodes to whom it may be a successor or finger. The number of such nodes is $O(log\ N)$, and therefore is quite limited.

Note, however, that there is a tradeoff between performance and security: Performance is maximized if the responsible node replies directly to the initiator of the lookup, while security is maximized if lookups carry an operation request with them, and the reply travels back through the reverse lookup path (without any information about the address of who produced the reply). In between these two design points we can have intermediate solutions, like choosing a single intermediate node to relay the reply back to the client, i.e., the reply takes two hops to travel back to the client.

5.2 Uneven Distribution of Node Types

Another aspect that will complicate our design is the uneven distribution among distinct types: e.g., Windows XP represented 51% of the operating systems used to access Google's web site in 2004 [4]. This raises a load-balancing problem: the "islands" of the most common types will be much more populated than those of other types. Consequently, there is much more work to do by nodes of less common types.

A possible way to alleviate this problem is to perform an automatic classification of nodes in different types according to the output of remote OS fingerprint tools [9]. Such tools enable the remote collection of information about the software environment each node is running (like operating system and services that are running).

This information could be collected when a node joins the overlay (and audited periodically) and would enable a more reliable classification of nodes, and also the distinction between nodes that may be running on similar platforms but are less likely to have common vulnerabilities due the types of services that they run. The latter might be used to improve the load balancing by partitioning the most populated sets of node types.

5.3 Generalizing to Other Overlays

Even though DHTs are gaining in popularity, many popular p2p applications are based on unstructured overlays. We believe the design principles stated in section 2 can also be applied to modify the design of unstructured overlays.

For instance, in BitTorrent the tracker is responsible for assigning neighbors for peers to download content, hence for forming the overlay graph. So, if we assume that the tracker is not vulnerable to worm infection (e.g., it will not run any services, run behind a firewall, etc.), then it will be able to assign neighbors in a way that forms an overlay graph with the generic structure of Figure 1.

6 Related Work

The containment of p2p-assisted worms is a recent research area.

One of the first papers to point out the existence of this problem was the work of Zhou et al. [18]. In this workshop paper, the authors motivate the problem, and propose as their main research direction populating p2p overlays with guardian nodes. These are special nodes that are running worm-detection software (which the authors later proposed in a separate paper [6]) that tracks

how information from untrusted sources propagates itself in memory. These have to be special purpose nodes, since this detection considerably slows down the execution. This differs from our vision of a true p2p system where all nodes have common responsibilities, and where the overlay graph is modified to contain the propagation of the worms. In this paper, Zhou et al. also mention how the existence of immune nodes could slow down the propagation, but do not propose any reorganization of the overlay to achieve contention.

Yu et al. [17] propose a model for p2p-assisted worms, and analyze the propagation of these worms depending on the attack model (e.g., whether the worm uses the overlay topology or not), and on the structure of the overlay. They point out that these worms propagate much faster than traditional scanning worms, and that unstructured overlays can also lead to faster propagation. They do not propose, but mention as future work, the design of defense systems.

Ramachandran and Sikdar [11] have proposed an analytical model for the dissemination of worms in p2p overlays. They conclude that an accurate model needs to take into account user characteristics and communication patterns.

Chen and Gray [5] have also studied the propagation of worms in p2p overlays using simulations, but, unlike the previous two papers, they have considered a dynamic peer population instead of a static overlay graph. They also propose a detection mechanism based on the observation that worms distort node popularity, reflected in changes in connection rates.

We contrast with the previous papers in that they focus on a better understanding of the problem using models and simulations, whereas our proposal focuses on the defenses required to contain p2p-assisted worms.

A Warhol worm [16] is a different kind of fast-spreading worm that uses several optimizations in the scanning routines like precompiled IP lists. Our work focuses on worms that also have the potential for fast spreading, but the reasons for that are quite different.

7 Conclusion

This paper presented a novel overlay called Verme that is designed to contain, or at least slow down the propagation of p2p-assisted worms. Verme extends Chord by reorganizing the overlay graph, taking into account the fact that some overlay nodes may not have common vulnerabilities, due to their platform diversity. Verme's design still presents a large number of challenges and open issues, some of which were discussed here. In the future we intend to address these issues and implement a prototype of the new overlay.

References

[1] http://www.securityfocus.com/bid/4951.

[2] http://www.securityfocus.com/bid/6747.

[3] http://www.cert.org/incident_notes/IN-2004-01.html.

[4] Operating systems used to access Google - June 2004. http://www.google.com/press/zeitgeist/zeitgeist-jun04.html.

[5] G. Chen and R. S. Gray. Simulating non-scanning worms on peer-to-peer networks. In *Proceedings of 1th International Conference on Scalable Information Systems (INFOSCALE 2006)*, Hong Kong, China, May 2006.

[6] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of internet worms. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP 2005)*, Brighton, United Kingdom, Oct. 2005.

[7] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating System Principles*, Banff, Canada, Oct. 2001.

[8] J. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, MA, USA, Mar. 2002.

[9] R. Farrow. System fingerprinting with nmap. *Network Magazine*, Nov. 2000.

[10] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *Proceedings of HotNets 2003*, Cambridge, MA, USA, Nov. 2003.

[11] K. Ramachandran and B. Sikdar. Modeling malware propagation in gnutella type peer-to-peer networks. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece, Apr. 2006.

[12] S. Singh, G. V. C. Estan, , and S. Savage. Automated worm fingerprinting. In *Proceedings of 6th Symposium on operating design and implementation 2004 (OSDI 2004)*, San Francisco, CA, USA, Dec. 2004.

[13] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Proceedings of USENIX Security Symposium 2002*, San Francisco, CA, USA, Aug. 2002.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, San Diego, CA, USA, Aug. 2001.

[15] K. Walsh and E. G. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *3rd Symposium on Networked Systems Design and Implementation (NSDI 06)*, San Jose, CA, USA, May 2006.

[16] N. Weaver. Warhol worms: The potential for very fast internet plagues. http://www.iwar.org.uk/comsec/resources/worms/warhol-worm.htm.

[17] W. Yu, C. Boyer, S. Chellappan, and D. Xuan. Peer-to-peer system-based active worm attacks: Modeling and analysis. In *IEEE International Conference on Communications (ICC)*, Seoul, South Korea, May 2005.

[18] L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa, and S. Chien. A first look at peer-to-peer worms: Threats and defenses. In *Proceedings of the 4th International Workshop on Peer-To-Peer Systems (IPTPS'05)*, Ithaca, NY, USA, Feb. 2005.