Peer-to-Peer Botnets

Ethan Pailes

December 13, 2017

1 Abstract

One of the chief signatures of botnet malware is the list of control servers that bots use to phone home and receive new directives. These centralized control servers provide a central point of failure for the network, allowing mitigators to take down the whole network by blocking or taking down the control servers. To get around this restriction, some of the more sophisticated botnets use peer-to-peer command and control protocols. Rather than a single central node, the bots themselves implement command and control logic. In designing peer-to-peer C&C protocols, a malware author must consider trust and sinkhole attacks. This paper contains a review of known peer-to-peer C&C protocols and their mitigation strategies, as well as a new peer-to-peer C&C protocol designed to be resistant to sinkhole attacks.

2 Introduction

Both botnets and peer-to-peer networks have a long, and often intertwined history. Both technologies consist of a swarm of machines coordinating to achieve some means. In order to be successful, both must think seriously

about adversarial situations. Good designers of peer-to-peer networks assume an untrustworthy community, and good designers of botnets assume that their bots will be attacked by mitigators. A botnet can leverage the work on trustless protocols and the removal of a single point of failure offered by existing work on peer-to-peer technology to greatly improve its resiliency.

In 1999 Shawn Fanning and Sean Parker released an mp3 sharing application called Napster (Halter n.d.). Napster was not a true peer-to-peer network, as it used a centralized server to coordinate file search and peers had no real way to coordinate without the central server, but it grew in popularity so quickly that system administrators were soon complaining that mp3 sharing was consuming up to 60of university network bandwidth (Fusco 2000). It proved that there was a real demand for services that peer-to-peer technologies were uniquely well suited to provide. Peer-to-peer networks have a natural scaling technology, where the more people use the service, the more resources are available to service new requests. This stands in stark contrast to the traditional client-server architecture where companies must buy new servers to service the needs of more clients. It allowed Napster to roll out very quickly (it only operated between June 1999 and July 2001).

A year after Napster shut down Peter Maymounkov and David Maziéres invented the Kademlia distributed hash table, a core technology for designing peer-to-peer networks (Maymounkov and Mazières 2002). Kademlia uses the XOR function to impose a notion of distance between nodes on the network. It would be impossible for each node to remember all the details of the network topology, so instead nodes are only required to remember information about other nodes which are close according to this metric (note that XOR closeness is not correlated with physical closeness).

Meanwhile, the BitTorrent protocol was created in 2001 by Bram Cohen

as the spiritual successor to napster. BitTorrent uses a special descriptor file format to describe files to be shared in a cryptographically secure way. Peers with one of these descriptor files share data with each other in chunks, enabling download pausing and speeding the spread of a new file. Network coordination can proceed in either a centralized way using a tracker, which fills a role quite similar to Napsters centralized servers, or via a distributed hash table called Mainline DHT (*Azureus Introduces DHT Layer* 2005) in the style of Kademlia.

Malware authors were not ignorant of the developments in peer-to-peer technologies during the late 90s and early oughts. As early as 2003, the Sinit bot design was published (Wang, Sparks, and Zou 2007). While the Sinit bot was not fully peer-to-peer, instead using a hybrid architecture, it demonstrated how peer-to-peer technology might be used to increase the resilience of a botnet. By the mid oughts peer-to-peer botnets were popping up with increasing frequency (Grizzard et al. 2007).

The 2007 Peacom bot is a good example of a hybrid peer-to-peer botnet. This pattern is best illustrated through the example of the Peacom bot (Grizzard et al. 2007). The Peacom bot implemented a control protocol layered on top of the existing Kademlia DHT. Each day the bots were programmed to poll a new set of locations in the DHT. The botmaster would leave an encrypted note at the fixed locations with instructions for the day. This ensured that only the botmaster could leave instructions for the bot, though mitigators could prevent the botmaster from communicating with their bots.

Most peer-to-peer botnets are taken down by security researchers using a technique called sinkholing (Espiner 2012). The Kaspersky Labs takedown of the Hlux/Kelihos botnet is one example of such a mitigation scheme (Or-

tloff 2012). Mitigators stood up a sinkhole server which emulated a bot on the network botnet, then began trying to convince as many bots as possible to fully populate their peer lists with the sinkhole host. This meant that the sinkhole became the single-point of truth for an increasingly large share of the malware network, allowing the bots to be cleanly decommissioned. It is this possibility of a sinkhole attack that designers of peer-to-peer botnets must be most worried about.

3 To the Community

An awareness of peer-to-peer botnets is important because of the potential of peer-to-peer technology to make mitigation more difficult, and reduce the infrastructure burden on botmasters. As more and more manufacturers produce internet connected devices which have little hope of receiving long term security support, or even property security in the first place, the size of botnets will only increase. The Mirai botnet is one recent example of the destructive power that the internet of things is making available to botmasters (Krebs 2016). The war between botmasters and mitigators will only heat up, and understanding the tools available to botmasters will be important for people on both sides of the conflict.

4 A Peer-To-Peer Botnet Control Protocol

4.1 Background on Kademlia

A full explanation of the Kademlia(Maymounkov and Mazières 2002) distributed hash table is out of scope for this paper, but it will be hard to discuss the algorithm presented here without some background on the technology. A good place to start developing a richer understanding of the

protocol is the explanation of XOR distance and basic routing presented by the MaidSafe Foundation, a company developing technology on top of the Kademlia routing table (*XOR Distance and Basic Routing* 2014).

A Kademlia DHT is composed of a set of nodes, each of which possessing a random numerical identifier. Identifiers are assumed be be unique using the same justifications used for UUIDs (*Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components* 2005). It would be impossible for a single node to remember the location of all the other nodes in the network, so instead it remembers more nodes close to itself, and just a few nodes further away from itself. This similar to the way that a Tufts could tell you precisely how to navigate from Tisch Library to Davis Square, but would likely have a much vaguer notion of how to get to Monte Carlo from Paris. They might know that the journey would involve going south, but would probably not know the specific names of the roads involved.

For a human navigating in the real world, what is meant by distance is fairly straightforward, but it is not so clear how we determine distance between two numbers. One of the key insights of the Kademlia protocol is the fact that we can use XOR as a distance metric. The full justification and explanation of why this is a good idea is out of scope, but it is still worth noting a few friendly properties of XOR. Any number XORed with itself is zero. This matches our intuition that the distance between any point and itself is zero. Additionally, any two numbers not equal to one another XORed together is non-zero. This matches our intuition that two different points have some distance between them.

One key mechanic for any peer-to-peer network is the bootstrapping process, that is the process by which a node becomes a peer on the network from some minimal set of information. In the case of Kademlia, in order to bootstrap a node needs to know about at least one other node that is already on the network. The new node must perform a series of lookups on the network in order to learn about the nodes closest to it in XOR space. You can think of this process as something like what happens when someone new moves into a neighborhood and starts getting to know their neighbors and the local landmarks.

4.2 Design Goals

The chief benefits of a peer-to-peer network are resilience and reduced infrastructure cost. Bots ought to be hard to detect, and the network ought to be hard to take down. Control of the network should not require a large C&C infrastructure. In particular, botmasters ought to be able to control their network with only a private key and the ip address of a single bot. They would likely want to remember more than just a single ip address, but in principle a single infected host should act as a gateway to the whole network. This stands in contrast with hybrid peer-to-peer networks which separate bots into a worker caste and a control node caste.

4.3 Network Topology

In order to organize themselves into a distributed hash table, our bots will each need a random numerical identifier. Given that they will be operating in a hostile environment, it is almost certain to be useful for the bots to be able to encrypt traffic with their own key pair. Fortunately, public keys appear to be random data, so the two needs can be met at once by assigning

each bot a key pair and using the public key as the Kademlia identifier. Stock Kademlia calls for 180 bit identifiers, but the algorithm works just as well for larger identifiers though there is a performance cost. If we use AES256 as the chosen asymmetric encryption scheme, the entire key can be used without much fear of performance issues. A nice benefit of using keys as identifiers is that traffic between bots can be entirely encrypted, because whenever it is possible to talk to a particular bot you also have its public key.

4.4 Infection Sequence

An important part of any bots life cycle is looking for new hosts to infect. The particular exploits used to gain access to a host are out of scope for this design, but how a new payload is generated is important. A copy of the bot will be instantiated with a fresh key pair, a list of peers to use for bootstrapping onto the network (drawn from the parent bots own peer list), and the current command sequence number. Any obfuscation steps such as string mangling or random nop injection will be performed. At this point the new bot is ready to be used as a payload in an attack.

After infection, the bot must bootstrap onto the network. All that is required for bootstrapping is a single node on the network, and the parent bot will have provided the child bot with a number of options. The child will iterate through its bootstrap list until one of the peers responds, and then it will go through the standard Kademlia bootstrapping process.

4.5 Commands

In order to control the network, botmasters must have some way to issue commands. There are a number of ways this might be accomplished from

sending a dynamic library over the wire and invoking RPCs against it to embedding a scripting language such as Lua in the bots to providing a login shell. The particulars are out of scope for this design. Instead we focus on the network behavior required to send the command payloads, whatever they might be.

4.6 Single Commands

The simplest type of command is a command sent to a single machine. Given a bot identifier, the botmaster can execute a command on that bot. Bots all possess the public keys of the botmasters, so it is straightforward to encrypt the command payload with the bots public key (its identifier), and sign it with the botmasters key so that the bot can be sure that it is receiving a command from the right folks.

4.7 Broadcast Commands

A more complicated type of command entails asking all bots on the network to perform some action. An important principle of Kademlia is that no one peer has a global view of the network, so it isnt feasible to collect the address or identifier of each bot and then execute a single command. Instead, the bots must be relied upon to forward the command to their peers. Naively, we might just have each bot execute the command and then forward it to each of its peers. The problem with this is that any one bot is almost certainly on the peer list of several other bots. Then this approach would result in commands traveling around the network forever in cycles, repeatedly executing on each bot. Obviously, this is a bad thing, so we need to come up with some way to kill cycles.

We might consider just maintaining a hashset of all previously executed

commands and refusing to re-execute them, but this has problems of its own. It is quite likely that the botmasters will want to execute the same broadcast command on more than one occasion (such as a command that polls all bots to determine network health). The other issue is the size requirements of the hashset if the bots are long lived.

Instead, we use a broadcast command sequence number. The botmasters are responsible for remembering the number of commands that they have broadcast, and annotating each command with its index on broadcast. Each bot remembers the sequence number of the last command it executed, so it can just drop any commands which come from the past on the floor without forwarding them. Any commands which come from the future (more than one sequence number past the last executed command), can be buffered. This has the added benefit of guaranteeing in-order execution of commands, a very useful semantic property to have. Without this property botmasters would have to batch up blocks of commands that rely on being executed in order into a single payload. In order to deal with the case where a bot missed a particular command, previous commands will be stored as data objects on the Kademlia network, so that any bot can used the DHT to fill holes in its command stream. Without the Kademlia DHT bots would have to buffer past commands to allow replay, but the DHT provides an elegant solution to the problem. The storage will be amortized between all the bots, so no one bot will have a large storage burden.

4.8 Mitigation Concerns

If this botnet gained any amount of success, it is inevitable that mitigators would gain control of one or more bots. After they reverse engineered all the shenanigans used to obfuscate the code, this means they would have

full access to anything the bot knew. This means they would have access to its private key, and the ability to completely control its behavior on the network.

4.9 Command Injection

If migitgators could inject a broadcast command into the network, they could take down the whole network. Fortunately, because bots will always refuse to execute commands not encrypted by the botmasters private key, this is impossible.

4.10 Command Hiding

A captured bot might try to edit the command sequence to cause downstream peers to execute commands out of order. Command sequence numbers prevent this. A captured bot can still refuse to forward all commands, effectively taking itself off the network, but it is impossible to prevent this once a bot is captured.

4.11 Sinkhole Attacks

The main mitigation approach for peer-to-peer botnets is the sinkhole attack. This involves filling the peer-lists of each bot with entries pointing towards a sinkhole machine or set of sinkhole machines. In the context of a Kademlia structured network, each peer-list is highly structured, and must follow the pattern of containing more peers that are close by and fewer peers that are far away. This means that in order to sinkhole a particular peer, mitigators must gain possession of a particular pattern of bots, something which is difficult to do. This is made even harder by the fact that nodes assemble their peer list by talking to many machines, so it is impossible to

sinkhole anyone with just one node. Thus mitigators must stand up a large number of captured bots in order to begin making any progress on sinkholing the network. Additionally, the captured bots must behave like ordinary bots in order to avoid being kicked off the network.

4.12 Sybil Attacks

All peer-to-peer networks are vulnerable to Sybil Attacks (Douceur 2002), where the attacker creates a large number of fake identities, and uses them to subvert the consensus mechanisms by which the network organizes itself. If the attacker can control a majority of the identities on the network (or in some cases a supermajority), they can override control of the network. In the context of the network presented, an attacker would be unable to execute commands because controlling a majority of peers would not give them access to the botmaster key. However, it would still be possible to disrupt the functioning of the Kademlia network and prevent peers from talking to one another. Thus a Sybil attack on the network would be the same thing as a sinkhole attack.

5 Conclusion

Peer to peer botnets have the potential to increase the resiliency of malicious networks, and reduce the infrastructure cost required to run them. This report presents one design for a peer-to-peer botnet using standard peer-to-peer primitives and leveraging common crypto technology. It raises the possibility of a botnet controlled entirely with a single key pair and a lightweight client program. Without any centralized C&C infrastructure to attack, it becomes much more difficult to take down an entire botnet. The only remaining solutions for mitigators involve directly attacking the

network via a Sybil or Sinkhole attack. Another avenue of attack could be finding an implementation flaw in the software itself, or deploying cleaning software capable of seeing through any obfuscation techniques to any potentially infected machines.

References

Azureus Introduces DHT Layer (2005). http://www.slyck.com/news.php?story=772. Accessed: 2017-12-13.

Douceur, John (JD) (2002). "The Sybil Attack". In: *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*. URL: https://www.microsoft.com/en-us/research/publication/the-sybil-attack/.

Espiner, Tom (2012). *Second Kelihos botnet steps into a sinkhole*. http://www.zdnet.com/article/second-kelihos-botnet-steps-into-a-sinkhole/. Accessed: 2017-12-13.

Fusco, Patricia (2000). *The Napster Nightmare*. https://web.archive.org/web/20111019152028/http://www.isp-planet.com/politics/napster.html. Accessed: 2017-12-13.

Grizzard, Julian B. et al. (2007). "Peer-to-peer Botnets: Overview and Case Study". In: *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*. HotBots'07. Cambridge, MA: USENIX Association, pp. 1–1. URL: http://dl.acm.org/citation.cfm?id=1323128.1323129.

Halter, Burk. *Napster: Then and Now*. http://iml.jou.ufl.edu/projects/spring01/burkhalter/napsterhistory.html. Accessed: 2017-12-13.

Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components (2005). Standard. International Organization for Standardization.

Krebs, Brian (2016). *Source Code for IoT Botnet 'Mirai' Released*. https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/. Accessed: 2017-12-13.

Maymounkov, Petar and David Mazières (2002). "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. London, UK, UK: Springer-Verlag, pp. 53–65. ISBN: 3-540-44179-4. URL: http://dl.acm.org/citation.cfm?id=646334.687801.

Ortloff, Steven (2012). *Botnet Shutdown Success Story again: Disabling the new Hlux/Kelihos Botnet*. https://securelist.com/botnet-shutdown-success-story-again-disabling-the-new-hluxkelihos-botnet-49/32623/. Accessed: 2017-12-13.

Wang, Ping, Sherri Sparks, and Cliff C. Zou (2007). "An Advanced Hybrid Peer-to-peer Botnet". In: *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*. HotBots'07. Cambridge, MA: USENIX Association, pp. 2–2. URL: http://dl.acm.org/citation.cfm?id=1323128.1323130.

XOR Distance and Basic Routing (2014). https://www.youtube.com/watch?v=w9UObz8o8lY. Accessed: 2017-12-13.