Conducting and Optimizing Eclipse Attacks in the Kad Peer-to-Peer Network

Michael Kohnen, Mike Leske, and Erwin P. Rathgeb

University of Duisburg-Essen, Institute for Experimental Mathematics,
Ellernstr. 29, 45326 Essen
Michael.Kohnen@iem.uni-due.de, Mike.Leske@stud.uni-due.de,
Erwin.Rathgeb@iem.uni-due.de

Abstract. The Kad network is a structured P2P network used for file sharing. Research has proved that Sybil and Eclipse attacks have been possible in it until recently. However, the past attacks are prohibited by newly implemented security measures in the client applications. We present a new attack concept which overcomes the countermeasures and prove its practicability. Furthermore, we analyze the efficiency of our concept and identify the minimally required resources.

Keywords: P2P security, Sybil attack, Eclipse attack, Kad.

1 Introduction and Related Work

P2P networks form an overlay on top of the internet infrastructure. Nodes in a P2P network interact directly with each other, i.e., no central entity is required (at least in case of structured P2P networks). P2P networks have become increasingly popular mainly because file sharing networks use P2P technology.

Several studies have shown that P2P traffic is responsible for a large share of the total internet traffic [1, 2]. While file sharing probably accounts for the largest part of the P2P traffic share, also other P2P applications exist which are widely used, e.g., Skype [3] for VoIP or Joost [4] for IPTV. The P2P paradigm is becoming more and more accepted also for professional and commercial applications (e.g., Microsoft Groove [5]), and therefore, P2P technology is one of the key components of the next generation internet.

P2P networks can be categorized into two main types, the unstructured and the structured ones. The structured networks have gained popularity recently in science and are becoming increasingly popular also in practice. One of the most popular structured P2P networks is the file sharing network Kad which is used, e.g., by eMule [9] and aMule [10].

Structured networks introduce identifiers for both content and nodes and a notion of closeness between those identifiers (see below). Thereby, they define which nodes are responsible for a certain content item, e.g., a file provided for sharing in the network. The logical closeness metric is completely independent from physical closeness.

The structured approach offers more precise and comprehensive lookup compared to unstructured networks as it ensures that all content that exists will actually be found. The well-defined responsibility of nodes for specific content, however, also introduces specific weaknesses. Users can create multiple identifiers [13] and select them such that they control all nodes that are responsible for a certain content item. If they succeed, they can control access to it and can also limit its availability, which is called Eclipse attack.

In [7], Steiner et al. state that an Eclipse attack is easily possible in the Kad network, as the nodes can freely choose their node identifiers, and present results of successfully performed Eclipse attacks. However, since the publication of [7], new versions of eMule and aMule have been released which include security measures against these kinds of attacks. Steiner himself states on [14] that the changes affect the practicability of his attacks.

Motivated by Steiner's work, we analyzed the security measures built into the state-of-the-art Kad clients and developed a way of circumventing them in an efficient way. In this paper, we describe the basic approach for the improved attack as well as ways to make it highly efficient. The experiments described provide a proof of concept and also give an indication of the effort required for the attack in a realistic scenario.

2 How the Kad Network Works

Kad is based on the Kademlia algorithm which was presented by Maymounkov and Mazières [8]. It is only defined by its implementations in several file sharing applications such as eMule [9], aMule [10] and MLDonkey [11] – no formalized protocol specification exists. Details of the aMule Kad implementation can be found in [12]. This paper concentrates on eMule, as it is the most widespread application.

Every file that is available in the Kad network has a *file ID* that is derived by calculating an MD4 hash value of the contents (not the name) of the file. Every Kad node has a *node ID* that is – normally – randomly generated the first time the node enters the Kad network and retained afterwards. Both IDs are 128 bits long. The logical distance between two identifiers is calculated using the XOR operation. Hence, the more bits match at the beginning of the IDs, the smaller is the distance between them.

The nodes with the smallest distance from a certain file ID are responsible for that file. The set of responsible nodes is not statically defined, but varies as nodes join and leave the network. Therefore, the set of responsible nodes needs to be determined again during every lookup process.

Nodes that offer a file store their contact information (IP address, port numbers, node ID) on the responsible nodes. Nodes that look for a file ask the responsible nodes for source nodes for that file. This concept is called indirect storage, as the responsible nodes do not store the file itself, but only pointers to nodes that store it.

To download a file from the network, its ID must be known. The default approach to retrieve the ID of the desired file is to use the keyword search mechanism of Kad, the other way is to use specific websites which list file IDs of popular files, thus bypassing the keyword search function of Kad.

When a node offers a new file, it stores a pointer to itself (*source information*) at the nodes responsible for the file ID, which allows other nodes to retrieve the file.

Additionally, it generates keywords by dividing the filename into parts (e.g., single words). It calculates hash values for every keyword (*keyword IDs*) and stores pointers at the nodes that are responsible for each keyword ID which contain the whole filename and the ID of that file. Each process of storing information for a file or keyword ID is called *publication process*.

If another user performs a *keyword search*, the client software calculates the keyword ID of the first keyword provided by the user and looks for the nodes that are responsible for that ID. In the subsequent request to those nodes, it includes the other keywords in clear text. The polled nodes use the provided keywords to filter their pointer lists and answer with matching files including their respective file IDs. The user can select the file(s) he wants to download from that list.

After the file ID of the desired file has been determined, the node performs a *source search* by asking the nodes that are responsible for the file ID for source nodes for that file and starts to download the file (parts) directly from those nodes.

Attacking the source search mechanism is our main target, as it is possible to eclipse files from the network this way regardless of how the user determined the file ID. During our analyses, we decided to attack the publication process as well, because only this way it can be ensured that only our nodes know about the content (see below). The attack against the publication process can be directed against both keyword and file IDs. The keyword search mechanism behaves the same way the source search process does, so no specific attack is necessary.

2.1 Kad Message Types

Two versions of Kad messages exist; we only use version 1 message types. All current versions of the client applications understand both versions of the messages. By using the older version, we circumvent the necessity to implement the obfuscation and advanced handshake features of newer versions in our tools. The usage of version 1 messages only has no impact on our tests.

Kad messages are transmitted via UDP. Every Kad message begins with a one-byte identifier indicating that the message is a Kad message, which is followed by a one-byte opcode. Message parameters are appended, if applicable.

Information about other peers is transmitted in a *peer information* data structure which can be found in many message types. This data structure contains contact information about the node, such as its ID, its IP address and the UDP port used for Kad communication, and various other pieces of information.

Hello Requests are used to check the availability of other nodes. We use the fact that Hello Request messages can also be used to insert a node into another node's routing table for our attack. The queried node is supposed to reply with a *Hello Response* message. Both messages only contain the peer information data structure of the respective sending peer.

Request messages are used to retrieve information about other nodes during a lookup process – explained in detail in section 2.2. They contain a Type field identifying the type of lookup (e.g., source search, keyword search, publication, etc.), the target ID and the ID of the queried node. A *Response* message contains the queried target ID and a peer information data structure for each matching node the queried node knows.

Publish Requests are used during a publication process to store information about offered files or keyword information on the responsible nodes. They contain the target ID and information about the published objects such as the filename or the bit rate of audio files. A node acknowledges a successful publication with a *Publish Response*.

During a source or keyword search, *Search Requests* are used to ask other nodes for sources or files matching a keyword. This message type contains the target ID and a field indicating whether the ID is a file or a keyword ID. In case of a keyword search with a search term consisting of more than one part, the second to last part of it are appended in clear text to the message. The queried node replies with a *Search Response* message containing the target ID and a result list.

2.2 The Lookup Process

A node performs a lookup process every time it requires (more) sources for a file, when it publishes a file or when the user performs a keyword search in order to find the currently responsible nodes for that file or keyword ID. When these nodes are identified, the specific action is performed, e.g., a query for sources.

As several lookup processes may run concurrently, eMule uses independent search processes for each lookup which are controlled by the so-called SearchManager. SearchManager invokes search processes and terminates them when their maximum run time is exceeded (default: 45 seconds) or their maximum number of results has been achieved (default: 300 sources during a source search, 10 nodes during a publication).

Each search process is split into two phases: In phase 1, the responsible nodes are identified. Up to three nodes are queried in parallel in order to accelerate the lookup process and to cope with non-responding nodes. The number n of required nodes depends on the action: During a publication process, a node contacts ten nodes to store the source information on them. A source search queries as many nodes as necessary until either the maximum number of sources or a timeout is reached. During phase 2, the specific action is performed on the identified nodes.

Phase 1: Locating the responsible nodes. A search process manages four lists: POSSIBLE, BEST, TRIED and RESPONDED. All lists are sorted by distance to the target with the closest node at the top.

At the beginning of a lookup, POSSIBLE will be filled with the 50 currently known nodes that are closest to the target ID. BEST stores the three closest nodes that have been found so far, TRIED stores the nodes that have already been queried, and RESPONDED stores all nodes that have responded to queries.

After initializing, the three BEST nodes are queried for closer nodes and copied to TRIED. The nodes answer with a list of nodes that are closer to the target. Those nodes are inserted into POSSIBLE and also into BEST, if applicable. Nodes in BEST are queried immediately for closer nodes.

Eventually, this process starves, as no closer nodes will be found. BEST then contains the three currently closest nodes to the target. Fig. 1 illustrates this process.

A search process cannot transition to phase 2 on its own. SearchManager periodically evaluates the status and transitions the search process to phase 2 if no response has been received for more than three seconds.

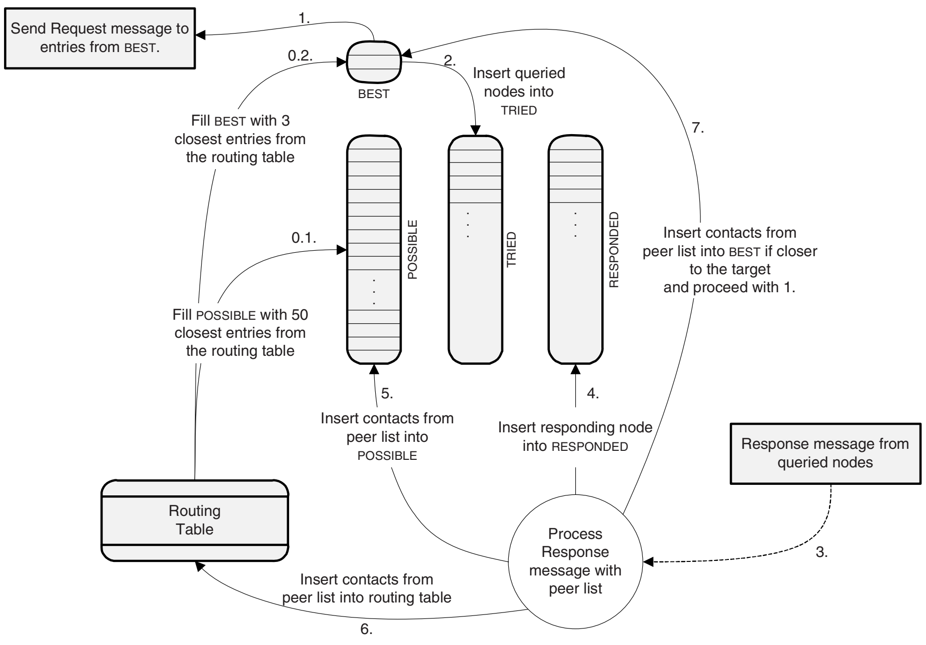
Send Request message to entries from BEST.
1.
0.2.
BEST
2.
Insert queried nodes into TRIED
Fill BEST with 3 closest entries from the routing table
7.
0.1.
POSSIBLE
TRIED
RESPONDED
Insert contacts from peer list into BEST if closer to the target and proceed with 1.
Fill POSSIBLE with 50 closest entries from the routing table
5.
Insert contacts from peer list into POSSIBLE
4.
Insert responding node into RESPONDED
Response message from queried nodes
Routing Table
Process Response message with peer list
3.
Insert contacts from peer list into routing table
6.

Fig. 1. Phase 1 of the lookup process. The initialization procedures are displayed as "0.x". All lists are sorted by distance to the target.

Phase 2: Performing the specific action. In phase 2, the search process evaluates the nodes contained in POSSIBLE, starting with the first node. If the node is also contained in TRIED and RESPONDED, the specific action is performed and the node is removed from POSSIBLE; if it is not in TRIED and RESPONDED, the search process falls back into phase 1 and queries this node for closer nodes.

2.3 Security Issues: Sybil and Eclipse Attacks and Countermeasures

Kad is susceptible to both the Sybil and the Eclipse attacks. A Kad node selects its own ID autonomously. Therefore, a malicious user can select specific IDs instead of randomly generated ones. Additionally, he can use multiple IDs concurrently, which is called a Sybil attack [13]. If an attacker thereby provides or controls the n closest nodes to the victim ID, all queries for this ID will converge on his nodes, as the nodes whose IDs are closest to a queried ID are responsible for managing content identified by it. This way, the attacker performs an Eclipse attack, as he is able to control the victim ID and can decide whether and how to respond to incoming requests.

As a consequence, the eMule developer community has designed security measures against this attack that have been implemented in eMule 0.49a [9]. Prior versions do not include any countermeasures. The new countermeasures constrain the routing table as follows:

- One IP address must not have more than one node ID assigned.
- A Routing Bin[1] must not contain more than two nodes from the same /24 IP subnet.
- The whole routing table must not contain more than ten nodes from the same /24 IP subnet.

These constraints only apply to the routing table, so they are only checked when nodes are added to it. They are not applied during lookup processes, so lookups are as susceptible to Sybil and Eclipse attacks as before.

Without these restrictions, an attacker could easily create an arbitrary number of Sybil nodes using one IP address and therefore perform an Eclipse attack using only a single machine. The new restrictions force the attacker to use several machines located in different subnets, so the attack effort rises significantly.

Modifications ("mods") of eMule are developed besides the official eMule application. SafeKad (included in the modification MorphXT [15], e.g.) implements a similar countermeasure that is also applied during the lookup process: It only allows one node per /20 IP subnet in a single response message. This mechanism leads to a more secure lookup process, as an attacker now needs nodes that reside in different /20 IP subnets. However, we will show a way to also circumvent this security measure. SafeKad is not developed further, though, as its developers regard the security measures of the official client as superior [15].

3 An Improved Eclipse Attack on the Kad Network

Based on the attack described in [7], we developed an improved attack procedure that circumvents the new security measures of eMule 0.49a and SafeKad. To demonstrate its feasibility, we developed a tool suite consisting of specialized Kad clients conducting the attacks. We attack both the publication process and the source search process, as they differ in the amount of queried nodes (ten vs. as many as necessary to obtain the maximum amount of sources or until the timeout is reached). Both processes perform a lookup process to determine the nodes they need to query.

3.1 Basic Attack Concept

We define the *Target ID* as the Kad ID which we want to eclipse. For the source search process, this ID represents a file. The publication process attack can either be directed against a keyword ID or a file ID, depending on what the attacker desires to eclipse (e.g., keyword "ubuntu" vs. file "ubuntu-8.04.2-desktop-i386.iso").

The Target ID resides in what we call a *CrawlArea* – the part of the Kad ID space surrounding the Target ID that we analyze for nodes that need to be manipulated. The size of the CrawlArea is defined by the length of the common prefix of the CrawlArea nodes and the Target ID; e.g., a "/20 CrawlArea" means that the first 20 bits of the CrawlArea nodes' IDs are the same as the first 20 bits of the Target ID.

[1] The routing table of a Kad node is divided into Routing Bins which contain up to 10 nodes and represent up to 6.25% of the Kad address space, depending on their position in the node's routing table. For details refer to [12].

In order to perform the attack, we use a chain of *Sybil nodes* (Sybil 0 to Sybil n). The Sybil nodes have different, specifically chosen Kad IDs where Sybil 0 has the largest distance to the Target ID and Sybil n the smallest. We call this order of nodes "decreasing order". The chosen node IDs and the Target ID have the first 125 bits in common, so it can be assumed that our nodes have smaller distances to the Target ID than any other node in the network. As a result, our nodes – and only our nodes – are responsible for the Target ID.

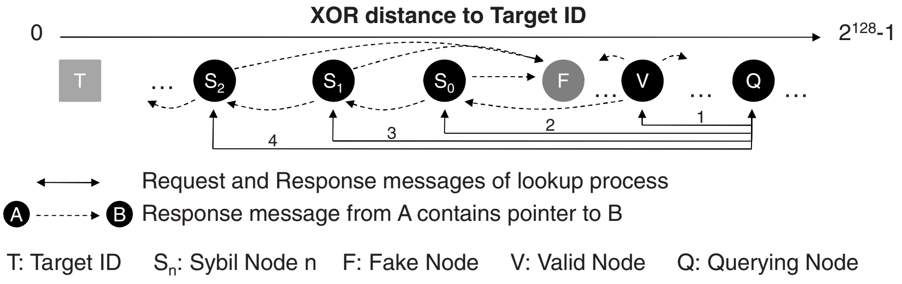
XOR distance to Target ID
0
2^128-1
T
S2
S1
S0
F
V
Q
1
2
3
4
Request and Response messages of lookup process
A
B
Response message from A contains pointer to B
T: Target ID
Sn: Sybil Node n
F: Fake Node
V: Valid Node
Q: Querying Node

Fig. 2. Illustration of how a querying node is lured into the Sybil node chain. The Sybil nodes are arranged in decreasing order. All IDs are arranged according to their distance to the Target ID.

The new security measures of the official eMule client and the SafeKad modification require new attack techniques: To circumvent eMule 0.49a's restriction that one Routing Bin may only contain up to two nodes from the same /24 IP subnet, only Sybil 0 is actively inserted into other nodes' routing tables. As this check is not applied during lookup processes, we use Sybil 0 to successfully direct the querying node to our other Sybil nodes in the same /24 subnet.

SafeKad's security measures prevent the usage of a single Response message containing all Sybil nodes, because they would discard all except one of them as they reside in the same /20 IP subnet. Hence, we include only one Sybil node and another randomly generated faked node in a Response message[2]. The random node's ID is chosen to be farther away from the Target ID than the Sybil's one.

When a node performs a lookup process, it will eventually be directed to Sybil 0 by other valid nodes that have been manipulated. Sybil 0 will respond with a Response message containing Sybil 1 and a faked node, which is illustrated in Fig. 2. When receiving this message, the node will immediately put Sybil 1 into its POSSIBLE and BEST lists, as it is closer than all previously found nodes, and query it for closer nodes. The Sybil node in turn answers with the next Sybil node and a faked node. This process is repeated.

Attacking a publication process requires ten Sybil nodes to eclipse a file or a keyword by deceiving the publishing nodes. This number is fixed, as a node stores the information always on ten nodes.

During a source search, more Sybil nodes can be required: When a Kad node tries to find sources for a file, it queries as many nodes as possible until it has found the

[2] The faked node is required because Kad expects a Response message to contain at least two nodes.

maximum amount of sources or reaches a timeout. Using the first condition, in theory, only one Sybil node is required: The search terminates when 300 sources have been found (default value). If the first queried node replies with 300 randomly generated, non-existent sources, no further Sybil nodes are required. The drawback of this approach is that it is visible to the user, as the client application will first report 300 sources which will then drop to 0 after all result nodes have been queried. We chose a more stealthy approach using the second termination criterion, the timeout. We try to keep the querying node busy long enough so that the search is terminated due to a timeout. To perform this attack, we introduced a delay after the reception of a Request message before sending the Response message. The number of required Sybil nodes is variable and depends on the delay interval (see 4.1).

Summarizing, our attack bases on a chain of Sybil nodes of which only the first node (Sybil 0) is actively proclaimed. Sybil 0 points querying nodes to the next node in the chain which then points to the next Sybil node and so forth. The Sybil nodes answer after a configurable delay with only the next Sybil node and an arbitrarily chosen fake node.

3.2 Optimizations

During our test phases, we discovered optimizations for the attacks on the publication and the source search processes.

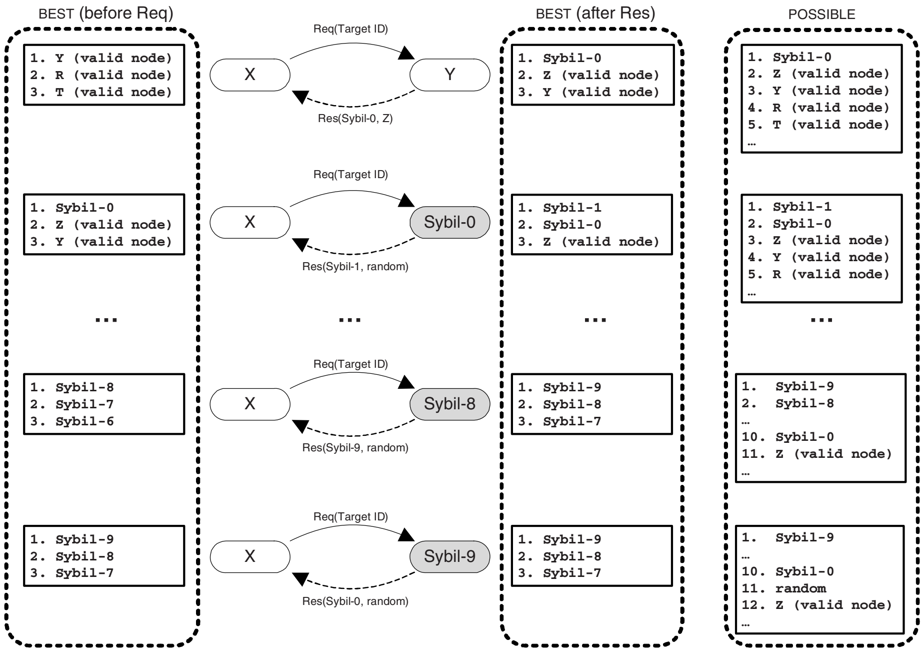

Fig. 3. The BEST and POSSIBLE lists before and after each message of the Sybil node chain

Accelerating the publication attack. To have a node enter phase 2 of the lookup process more quickly when we attack the publication process, we have Sybil 9 send Sybil 0's contact information instead of Sybil 10's ones. This way, the node notices that there are no new closer nodes and that it has found the ten currently closest nodes to the Target ID – which are our Sybil nodes 0 to 9 – and will start to send the Publish Requests to them immediately. Additionally, we chose the node ID of the fake, non-existent node included in Sybil 9's response message to be only a little farther away than our Sybil nodes to place it between the last Sybil and the first valid node in the list, in case one of the Sybils' answers does not arrive in time. If this happened, the attacked node would send a Publish Request to the first valid node on its list. This way, the request is sent to the non-existent node. The sending node does not receive an acknowledgement, so it tries the closest non-queried node on its POSSIBLE list, which should contain all ten Sybil nodes by then. Fig. 3 shows the status of the BEST and POSSIBLE lists before and after each message exchange when a node is lured into our Sybil node chain.

Minimizing the number of required Sybil nodes for the source search attack. During our test runs, we discovered that the number of required Sybil nodes for the attack on the source search can be minimized by reversing the order of the Sybil nodes so that Sybil 0 is the closest node to the target and Sybil n is the farthest one ("increasing order") – Fig. 4 illustrates this order of Sybil nodes. This change causes a node to repeatedly transition between phases 1 and 2 during the lookup process: The node tries to find closer nodes by only querying the currently known closest nodes. As Sybil 0 is the closest node, the process starves when Sybil 0 is queried and returns Sybil 1 which is farther away. The starvation causes a transition to phase 2 for the first time. The node queries Sybil 0 for sources, does not obtain any and discovers that it has not found the maximum amount of sources yet. Therefore, it continues its lookup by also querying the next node in POSSIBLE – which is Sybil 1 – for closer nodes and thereby transitions back to phase 1. This process repeats until the timeout is reached.

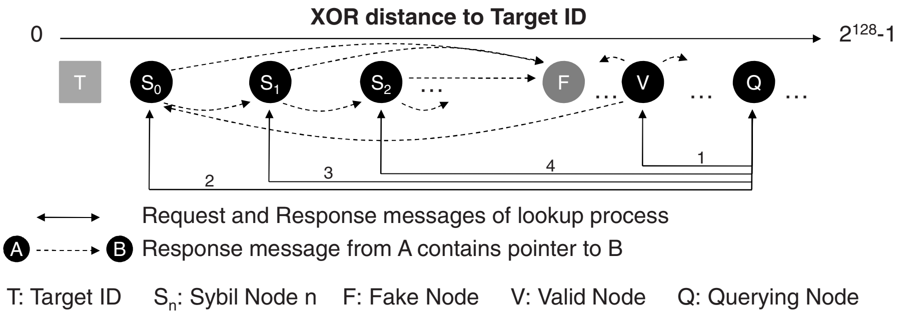
XOR distance to Target ID
0
2^128-1
T
S0
S1
S2
F
V
Q
1
2
3
4
Request and Response messages of lookup process
A
B
Response message from A contains pointer to B
T: Target ID
Sn: Sybil Node n
F: Fake Node
V: Valid Node
Q: Querying Node

Fig. 4. Illustration of the increasing order of the Sybil nodes

3.3 Tool Suite

To perform our analyses, we developed a tool suite consisting of three parts: *KadCrawler*, *Capo* and *SybilNode*. KadCrawler is responsible for finding all nodes in the CrawlArea and storing them in a database. The area of the address space containing the nodes that are poisoned is a subset of the CrawlArea which we call *PoisonArea*.

The routing tables of the nodes in the PoisonArea are then poisoned by the second tool, Capo: It selects the PoisonArea nodes from the database and sends a Hello Request message to them containing peer information about itself, so Capo takes the role of Sybil 0. Using the Hello Requests, Capo tries to insert itself into the routing table of the receiving node. Unfortunately, the reception of a Hello Response message cannot be interpreted as a successful insertion into the routing table, as a node replies before checking if the node should be inserted, so there is no way of determining whether the insertion was successful.

Capo performs the poisoning process every ten minutes to ensure that the currently closest nodes are manipulated. It also responds to incoming Request messages querying for closer nodes for a given ID. If the queried ID is the Target ID, Capo responds by sending information about Sybil 1 and a faked node to lure the querying node into our chain of Sybil nodes. Hello and Publish Requests for arbitrary IDs are answered positively and logged; Search Requests are logged, but not answered.

The third tool, SybilNode, is a functionally reduced Capo as it does not actively poison any nodes. It only reacts to incoming messages the same way Capo does. The Sybil nodes 1 to n are instances of this tool.

4 Results

Using our tool suite, we performed two test series. First, we performed short tests using one file with random content and name and ten test runs for each test scenario. The scenarios vary by the order of the Sybil nodes (decreasing vs. increasing) and the answering delay.

For the second test, we created 12 such files and tested for 90 hours to conduct the efficiency analysis. We refrained from eclipsing file or keyword IDs of content offered by third parties due to legal reasons.

4.1 Proof of Concept

For the test with one file, we selected a 12-bit PoisonArea. The test runs were performed using eMule 0.48a with SafeKad and eMule 0.49b as client applications. After each run, all configuration files were reset to defaults.

All attacks against the publication process were successful, as all ten Publish Requests of each test run were sent to our Sybil nodes. Both eMule 0.48a with SafeKad and eMule 0.49b were susceptible to our attacks.

The attacks against the source search process were conducted in both decreasing and increasing order of the Sybil nodes. We varied the answering delay of the Sybil nodes using values of 0.0, 0.5, 1.0, 1.5 and 2.0 seconds. Longer delays are not reasonable, as a lookup process is transitioned to phase 2 after a timeout of 3 seconds. If this would happen before the node's lookup timeout had been reached, the node would query valid nodes as well which would avert the attack. Table 1 presents the average number of Sybil nodes required for the attack and shows that our optimization of the order of the Sybil nodes remarkably reduces the amount of necessary Sybil nodes.

For delays < 1.0s and an decreasing order of Sybil nodes, more than 30 Sybil nodes are required for the attack to be successful. The number of required Sybil nodes decreases with increasing delay. The optimal configuration is to arrange the Sybil nodes in increasing order and to use a delay of 2.0 seconds.

Table 1. The average number of required Sybil nodes to let the source search process reach the timeout

		Answering Delay [s]				
		0.0	**0.5**	**1.0**	**1.5**	**2.0**
Decreasing Order	**eMule 0.48a (SafeKad)**	>> 30	> 30	24.7	16.9	13.0
	eMule 0.49b	>> 30	> 30	24.8	14.9	11.0
Increasing Order	**eMule 0.48a (SafeKad)**	8.0	7.0	5.5	5.5	5.1
	eMule 0.49b	7.2	6.5	5.8	5.1	4.6

4.2 Efficiency Analysis

After proving that our tools work, we analyzed the required amount of nodes that need to be poisoned in order for the publication attack to be successful. We chose to attack the publication process although it requires more Sybil nodes and messages and therefore resources, because only by attacking the publication, an attacker can ensure that he controls all nodes that store information about the content item. If only the search process would be attacked, there would exist valid nodes that store the information and which could be found by chance, e.g., due to network failures or misbehaving client applications.

Each of the 12 test files was published every 20 minutes. We only used an unmodified eMule 0.49b as a client application as we had proved before that also eMule 0.48a with SafeKad does not avert our attack.

We used 13- to 24-bit masks as PoisonArea sizes. The results can be divided into three groups: Attacks using 13- to 20-bit masks were immediately successful. The ones with 21- and 22-bit masks were successful after 60 and 5 hours respectively. The attacks using 23 and 24 matching bits failed.

Immediately successful attacks. The attacks using 13- to 19-bit masks were immediately successful. Every publication process performed by one of our eMule clients succeeded. By contrast, 2 out of 270 attacks using 20-bit masks failed. The reason was that the lookup process of the publishing eMule client starved before it had found a poisoned node and therefore transitioned to phase 2 before our Sybil nodes had been found. This led to our Sybil nodes receiving only 8 of 10 Publish Requests.

Delayedly successful attacks. As we limited the amount of nodes chosen to be poisoned by demanding that the first 21 resp. 22 bits of the Target ID and the node ID match, the attacks on these two identifiers were not successful at first, because no matching nodes were found. This is not surprising, as [7] assumes that the Kad network contains up to 4 million nodes: If the node IDs were uniformly distributed, 0.95 to 1.91 nodes should match our criterion on an average (e.g., $4 \cdot 10^6 / 2^{22} = 0.95$).

After 57 hours, the poisoning process for the 21-bit mask area was successful for the first time; the last success occurred after 67 hours. In total, only six different Kad nodes were poisoned during 15 out of 490 poisoning processes. However, after the first node was poisoned, all subsequent publication processes of our eMule clients until the end of the measurement were successfully directed to our Sybil nodes, although no nodes were poisoned during the last 23 hours. This proves that the information of Capo's presence was spread in the target area by the poisoned nodes. This information persists in the routing tables as long as Capo responds to Hello Requests. So the attack is effective without requiring a re-poisoning as long as the poisoned nodes are available.

The poisoning process using the 22-bit mask was successful after 5 hours of measurement. For the same reason the 20-bit mask attack failed two times, 3 of 256 attacks failed.

Unsuccessful attacks. The attacks on the 23- and 24-bit mask areas never succeeded. For the 24-bit mask area, no matching node was found. Information about a single node having the first 23 bits in common with the Target ID was indeed found, but the node did not respond to Capo's Hello Request.

4.3 Scalability

Due to legal reasons, we only attacked self-generated content that was demanded only by our clients. However, by chance, one of our generated IDs lay close to the ID of a keyword that was accessed by other nodes. We received information about 75,000 content items in more than 125,000 Publish Requests during the 90 hour measurement. Only 14,714 Search Requests were sent to our Sybil nodes (and left unanswered), so the keyword was published more frequently than demanded.

The queries caused by the keyword did not have any impact on our tools or our other attacks. We therefore suppose that the only factor limiting the success of the attack is bandwidth: If requests of publishing nodes are not answered due to packet loss or timeouts, the nodes will publish on other nodes and the attack fails. Assuming an average Kad packet size of 100 bytes, 20 messages per direction and a duration of 5 seconds per publication process results in a network load of 3.2 kbit/s on average. Adding 50% of overhead for the poisoning and Hello and Search Request messages results in a load of 4.8 kbit/s. Using a symmetrical 2 MBit/s connection, more than 400 publications per second would be necessary for the attack to fail. The attack would fail then because the querying nodes would ask other than the Sybil nodes, as their query packets to the Sybil nodes would be (partially) lost due to congestion. As eMule's default re-publish interval is 5 hours, this would require 7.5 million users to publish the same file or keyword, which is 87.5% more users than the number of 4 million found by [7].

5 Conclusion and Outlook

We demonstrated that the Kad network is still susceptible to eclipse attacks despite the inclusion of security measures into the current official eMule client. The tool suite we presented is able to eclipse arbitrary popular content items using little resources.

We showed that it suffices to poison one single – namely the currently closest – valid Kad node to be able to perform the attack.

In order to protect against our attack, we recommend to further harden the lookup process by having the IP subnet checks ignore nodes even though they are closer. However, if the attacker controls nodes in different IP subnets, these checks would not help. Hence, we will continue our research by implementing and evaluating the concept of disjoint routing paths. Research in this direction might also help to identify means of disabling "botnets" such as the Storm Worm network, as they also employ P2P technology for communication and receiving orders [6].

References

[1] ipoque GmbH: ipoque Internet Study (2007), `http://www.ipoque.com/resources/internet-studies/internet-study-2007/`

[2] Haßlinger, G.: ISP Platforms Under a Heavy Peer-to-Peer Workload. In: Steinmetz, R., Wehrle, K. (eds.) Peer-to-Peer Systems and Applications. LNCS, vol. 3485, pp. 369–381. Springer, Heidelberg (2005)

[3] Skype website: `http://www.skype.com/`

[4] Joost website: `http://www.joost.com/`

[5] Microsoft Groove website: `http://www.microsoft.com/groove/`

[6] Holz, T., Steiner, M., Dahl, F., Biersack, E., Freiling, F.: Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, pp. 1–9 (2008)

[7] Steiner, M., En-Najjary, T., Biersack, E.W.: Exploiting KAD: possible Uses and Misuses. SIGCOMM Computer Communication Review 37(5) (2007)

[8] Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)

[9] eMule website: `http://www.emule-project.net/`

[10] aMule website: `http://www.amule.org/`

[11] MLDonkey website: `http://www.mldonkey.org/`

[12] Brunner, R.: A performance evaluation of the Kad-protocol. Master's thesis, University of Mannheim and Institut Eurécom, Sophia-Antipolis, France (2006)

[13] Douceur, J.R.: The Sybil attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002)

[14] Computer Communication Review Online, Editorial Zone of [7]: `http://ccr.sigcomm.org/online/?q=node/288`

[15] MorphXT website: `http://www.emule-mods.de/?mods=morphxt/`