On Reconnaissance with IPv6: A Pattern-Based Scanning Approach

Conference Paper · August 2015

DOI: 10.1109/ARES.2015.48

CITATIONS
37

READS
302

4 authors, including:

Peter Kieseberg
Fachhochschule Sankt Pölten
106 PUBLICATIONS **1,756** CITATIONS
SEE PROFILE

Edgar Weippl
University of Vienna
334 PUBLICATIONS **5,246** CITATIONS
SEE PROFILE

On Reconnaissance with IPv6: A Pattern-Based Scanning Approach

Johanna Ullrich, Peter Kieseberg, Katharina Krombholz, Edgar Weippl
SBA Research
Vienna, Austria
Email: (firstletterfirstname)(lastname)@sba-research.org

***Abstract*—Today's capability of fast Internet-wide scanning allows insights into the Internet ecosystem; but the on-going transition to the new *Internet Protocol version 6* (IPv6) makes the approach of probing all possible addresses infeasible, even at current speeds of more than a million probes per second. As a consequence, the exploitation of frequent patterns has been proposed to reduce the search space. Current patterns are manually crafted and based on educated guesses of administrators. At the time of writing, their adequacy has not yet been evaluated. In this paper, we assess the idea of pattern-based scanning for the first time, and use an experimental set-up in combination with three real-world data sets. In addition, we developed a pattern-based algorithm that automatically discovers patterns in a sample and generates addresses for scanning based on its findings. Our experimental results confirm that pattern-based scanning is a promising approach for IPv6 reconnaissance, but also that currently known patterns are of limited benefit and are outperformed by our new algorithm. Our algorithm not only discovers more addresses, but also finds implicit patterns. Furthermore, it is more adaptable to future changes in IPv6 addressing and harder to mitigate than approaches with manually crafted patterns.**

I. Introduction

Internet-wide scanning experienced a tremendous boom in recent years; and today, scanning the entire Internet takes not more than an hour to complete [1], [2]. This evolution has provided a number of insights into the Internet ecosystem: *Heninger et al.* investigated the cryptographic protocol TLS and SSH, and found hosts using the same keys as others [3]. *Durumeric et al.* studied the HTTPS certificate system and showed that the majority of trusted certificates were controlled by three organizations [4]. *Rossow* scanned for nodes which were vulnerable for reflection attacks and found millions of them [5]. Even further, scanning plays a vital role in vulnerability mitigation: By handing scanning results over to CERTs, *Kührer et al.* measured a drop of 92 % of hosts vulnerable to NTP reflection attacks [6]. *Durumeric et al.* measured a 47 % decrease of servers vulnerable to Heartbleed after reporting [7]. Apart from Internet-wide scanning, scanning a certain subnet has always been a part of penetration testing to discover potential victims.

The presented approaches have in common that they probe every address within a certain range. However, this method and an on-going development collide with each other: the replacement of the Internet's main protocol IPv4 by IPv6 [8] due to address scarcity. The new version has an increased address range making it impossible to scan even the smallest subnet in the common way [9]. As a consequence, practitioners and researchers looked out for alternative ways of reconnaissance[1], and the resulting development can be best described in two steps that followed different premisses:

(1) If one cannot probe all addresses, one has to use other sources to gain valid addresses. In a first step, researchers and practitioners accessed systems that stored addresses for their intended application. Thereupon are public archives, centralized application servers or various ways of leveraging the Domain Name Systems [10], [11]. The drawback of this approach is that nodes that are not participating or not listed are never discovered.

(2) If one cannot probe all addresses, one has to include more information to synthesize promising addresses. In a second step, researchers and practitioners targeted to reduction of address space through address patterns. Thereby, all addresses containing a certain pattern are probed. Known patterns arise from standardization (e. g. *Modified EUI format*) or striking structures in addresses (e. g. low-byte addresses) [10], [11]. This approach also has its drawbacks. The benefit of patterns that are inferred from standards is unknown because these standards are not without alternatives and other IPv6 addressing schemes exist. Striking patterns are typically crafted manually, based on educated guesses and might not be as wide-spread as they might seem to humans. To the best of our knowledge, no evaluation of pattern-based approaches is available.

In this paper, we overcome this gap and assess pattern-based scanning in IPv6 in an experimental set-up. We evaluate not only currently known patterns with respect to their applicability to reconnaissance, but also develop a pattern-based scanning algorithm. This algorithm automatically extracts patterns from a small training set in a first step, and generates addresses based on these findings for later scanning. Our results imply that pattern-based scanning is a feasible approach for the discovery of IPv6 hosts, but known patterns are of limited benefit. They are outperformed by our novel algorithm. Our algorithm's results vary with respect to certain parameters, however, we are able to pre-estimate the quality of our results.

We consider our approach worthwhile for the following reasons: (1) Our evaluation bases on three data sets of real-world addresses. The data sets include addresses from a large number of organisations and are considered to be representative for address assignment habits of administrators. (2) The

[1] In this paper, we use the term *reconnaissance* for the discovery of unknown hosts in a network. The term *(network) scanning* is used for the discovery of unknown host through sending requests in await for responses. According to this definitions, *scanning* is a means of reconnaissance.

algorithm extracts patterns automatically from a training set and generates addresses based on these findings. As it is likely that addressing habits might change in the future due to new technologies and standardizations, the algorithm is adaptable to future changes. (3) No response is ambiguous in real-world scanning: There might be no host listening to a certain address, or alternatively there might be no response to the requested service (port/protocol). With our experimental set-up, we are able to eliminate the second, non-IPv6 specific alternative. This way, we are able to isolate the impact of IPv6 address generation from non-IPv6 factors.

The remainder of the paper is structured as follows: Section II introduces related work considering aspects of reconnaissance in both protocol versions IPv4 and IPv6. Section III presents the considered scenario for scanning, explains our novel pattern-based algorithm for scanning and current, manually crafted patterns. In Section IV, we cover experiments to evaluate pattern-based scanning. The results are discussed in Section V, Section VI concludes this work.

II. Related Work

Our research is based on two foundations: First, we discuss networking scanning with the predecessor version IPv4 and highlight why approaches of more sophisticated address generation serve a different goal than increasing the number of discovered hosts. Second, we highlight ways of reconnaissance with IPv6, show the advantages of scanning with version 6 and further highlight current scanning approaches.

A. Scanning with IPv4

The de-facto standard IPv4 scanner is the open-source tool *nmap* [12]. This and similar tools are crafted for scanning small address ranges. Due to maintaining a connection-wise state they are not capable of sending high numbers of packets. Internet-wide scanning with a tool like *nmap* requires a high number of nodes, lots of time and/or money [3]. Their counterparts are specialized scanning tools that are optimized for high traffic rates: *IRL scanner* was the first in 2010 [13] and covered the whole Internet in 24 hours using a single machine. As this tool had never been released to the community, the issue was brought up again in 2013 and two new tools were presented: *ZMap* [2] and *masscan* [1]. *ZMap* is a modular, open-source tool that predominates in academia and enabled a variety of insights into the global Internet ecosystem, e. g., [3], [4]. *masscan* is also open-source and claims to be faster. Being both released to the public, both tools were found to be used in the wild [14].

With respect to this paper, the tools' address generation is of interest: *nmap* iterates through all addresses of a range in ascending order and starts with the lowest. This method imposes the drawback of possibly overloading a destination because close addresses are likely to be also topologically close. Internet-wide scanners thus aim to balance the traffic by scrambling the address order. *IRL scanner* uses a reversed linear congruential generator permutation, *ZMap* iterates over a multiplicative group of integers modulo a prime slightly larger than 2^{32}, and *masscan* encrypts an incrementally increased index by means of a hash function. Although the latters' address generation is more sophisticated, they still target to probe every single address in a range.

B. Reconnaissance with IPv6

Initially, the focus drifted to ways of reconnaissance beyond scanning [10], [9] due to the myth of IPv6's unscannability. On the one hand, sources that store addresses could be used: (1) Querying the DNS for known domains reveals addresses, and unhandy IPv6 addresses might be more likely listed than their IPv4 counterparts. (2) Different answers of certain DNS server implementations allowed the reduction of the address space because the server's response differs for empty non-terminal from other errors. (3) A variety of other services might also be used, e. g., Node Information Queries, log files or centralized application servers. On the other hand, (4) some IPv6 implementations responded to requests to multicast addresses with their unicast address and allowed reconnaissance for local adversaries. On the bottom line, no approach seemed more promising than scanning: The attacker actively invokes a response and is thus independent of the victim's networking customs. Scanning is locally as well as globally applicable and can base on a variety of protocols. Scanning exploits the protocols' intended functionality that cannot be fully prevented without an impact on regular networking. In return, one has to deal with the fact that not all addresses can be probed.

An early analysis of IPv6 addresses provided the insight that they include extra expressiveness due to their increased length [15]. *Reversing* this expressiveness is an approach to create actually used IPv6 addresses. *Gont et Chown* [11] searched through addressing standards for exploitable patterns for address reduction and also proposed patterns for manually crafted addresses. *Gont* implemented these patterns in the scanning tool *scan6* [16]. The idea behind pattern-based scanning is the reduction of search space as a consequence of the fact that not all addresses in IPv6 can be probed. Thus, one aims to probe more likely addresses prior less likely ones as opposed to the IPv4 approaches that use a changed address order solely to prevent destination overloading. These patterns are manually crafted based on educated guesses, and have never been evaluated with respect to their applicability. Further, manual crafting needs manual updates in case the underlying addressing schemes change, or do not go beyond obvious patterns.

This paper aims to overcome these issues: It assesses whether pattern-based approaches are feasible in general, evaluates the current approaches in detail and compares them to our novel pattern-based algorithm. This algorithm automatically discovers addresses, and generates new addresses for scanning based on these findings.

III. Scanning Design

In this section, we present our considered scenario for scanning, explain the recursive design of our novel pattern-based algorithm and finally describe the manually crafted patterns from the literature in detail.

A. Considered Attack Scenario

We consider hosts that reside in the same IPv6 network prefix, and an adversary that resides at an arbitrary location on the Internet without local access to the targeted network. The adversary aims to discover as many hosts as possible. He/She is aware of manually crafted address patterns and further has a representative sample of addresses in this prefix[2]. This scenario is typical for penetration tests or adversaries targeting a certain

[2] An IPv6 address consists of a 64-bit network prefix, and a 64-bit interface identifier. Technically speaking, the adversary aims to discover interface identifiers as the prefix is known, but we stick to the term addresses for comprehensibility.

organization unit. Internet-wide scanning consists of a multitude of such scenarios with different prefixes. The assumptions are realistic insofar as manually crafted patterns are publicly available ([16], [11]). The address sample might be gained from the organization unit itself, e. g., insider information, but might also be derived from a similarly organized network.

B. Recursive Algorithm

This dual-purpose algorithm automatically discovers patterns in a training set of addresses, and generates addresses based on these patterns for scanning. The algorithm for pattern discovery is recursive and refines a given pattern through the determination of an additional bit per recursion. This additional bit is chosen in a way that the refined pattern covers the highest number of addresses among all pattern candidates. With every recursion the number of determined bits increases by one, thus decreasing the number of undetermined bits by one. If the number of undetermined bits falls below a given threshold, address generation is started. Then, all addresses that contain the current pattern are generated in ascending order. In addition to the following textual representation, our algorithm is depicted in Algorithms 1 and 2 in Appendix A.

Refined Pattern: The refined pattern covers the highest number of addresses among all candidate patterns. To find this pattern, rules are created and their key performance indicator *support* is calculated. In detail, for every undetermined bit b_u two rules are generated: *Rule 1: The address is appropriate to the current pattern. ⇒ The undetermined bit b_u is zero.* and *Rule 2: The address is appropriate to the current pattern. ⇒ The undetermined bit b_u is one.* The support of a rule is the ratio of the number of addresses fulfilling the rule to the number of addresses fulfilling the current pattern. Applying the rule with the highest support to the current pattern provides the refined pattern that is provided to the next recursion.

Inverse Rule and Pattern: The recursion is not only recalled with the refined pattern, but also with its inverse pattern. In case the best rule of a recursion is *The address is appropriate to the current pattern ⇒ The undetermined bit b_v is zero.* Its inverse rule is *The address is appropriate to the current pattern. ⇒ The undetermined bit b_v is one.* Applying the inverse rule to the current pattern leads to the inverse pattern, and another recursion is called with this inverse pattern. This guarantees that all patterns are included in the manner of a binary search tree and more likely addresses are probed prior less likely.

Initialization: A pattern with at least one determined bit is necessary for initialization. The algorithm has to be started $2^{number\,of\,start\,bits}$ times to fully cover the search space.

Stop Condition: If the number of undetermined bits falls below a certain threshold, the recursive pattern generation is stopped. Based on the current pattern, all appropriate addresses are iterated in ascending order for scanning. The number of generated addresses is $2^{threshold-1}$.

The start bits and the threshold are parameters of our pattern-based algorithm. Start bits should be chosen in a way that does not impede pattern finding. The threshold of the stop condition defines the transition from pattern discovery to address generation, and thus defines the degree of exploitation of known combinations in addresses versus the flexibility to find slightly different addresses.

C. Manual Patterns

In this section we discuss manually crafted patterns as defined in [16]. The major difference between this work and our approach is that in [16] the list of scanned patterns bases on experience instead of sample analysis and is fixed. This means that an upgrade has to be done manually by releasing a new version of the scanner. With the approach outlined in this work, trends in the selection of address deployment will be incorporated into the reconnaissance. The manually crafted patterns are as follows.

Low-byte: It was detected that many administrators simply select low numbers for the two low bytes of addresses, so-called *low-byte* addresses. The scanned address range is 2001:db8::0-100:0-1500, i. e., 1.381.889 addresses in total.

Ports: Several different ports are defined as standard ports for services. Administrators use simple schemes to map these service ports into the last bytes of an IPv6 address. The port pattern uses 23 different port numbers of popular services to create four address ranges per port. By the example of FTP (port 21), these ranges are: 2001:db8::0-5:21, 2001:db8::21:0-5, 2001:db8::0-5:15 and 2001:db8::15:0-5[3].

OUIs: IPv6 Interface Identifiers in *Modified EUI format* contain the three byte Organizationally Unique Identifier (OUI), a fixed pattern of two bytes and another three free bytes. This pattern iterates through all 2^{24} addresses of a certain OUI, e. g., 2001:db8::1234:56ff:fe(00-ff):0-ffff with the showcase OUI 1234:56. [16] further mentions a *vendor* pattern consisting of all OUIs of a certain vendor, and a *virtual machines* pattern taking OUIs usually used by virtualization software like *VMWare* and *vbox*. We consider them as a particular case of the *OUI* pattern.

IV. Experiments

This section describes our experimental set-up and included data sets. Further, our gained results are provided and it is shown that our algorithm's performance can be predicted to a certain degree.

A. Experimental Setup

Our set-up consists of two *Python* scripts. One implements our recursive pattern-based algorithm and generates an address list. This address list represents the addresses that are scanned. A data set represents the addresses that are used by hosts in this network. The second script compares the generated list with this data set to evaluate the number of actually discovered hosts. We decided for this approach instead of scanning real-world networks in order to gain non-ambiguous results: No response in real-world scanning might have various reasons, like no host listening to this address or that the requested service is not supported. Our set-up allows to isolate IPv6 address generation from non-IPv6 factors.

For every run, two data sets are required: Our pattern-based algorithm requires a training data set to find patterns, while the evaluation algorithms requires a test data set to evaluate the number of successfully discovered hosts. Both data sets are created from our entire data (see below) by a 10-folds cross validation. Typically, the entire data set is split in ten portions of equal size, nine portions form the training data set, one portion the test data set [17]. We believe that using the smaller data set to find patterns, and discover addresses

[3] 21 in decimal is 15 in hexadecimal.

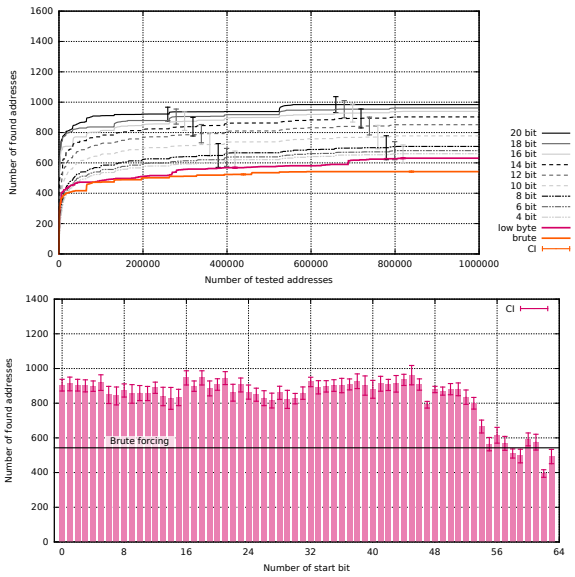

Fig. 1. *Routers:* Number of found addresses in dependence of probed addresses (top) and start bit (bottom). Comparison of our pattern-based algorithm, the *low-byte* pattern and brute-forcing.

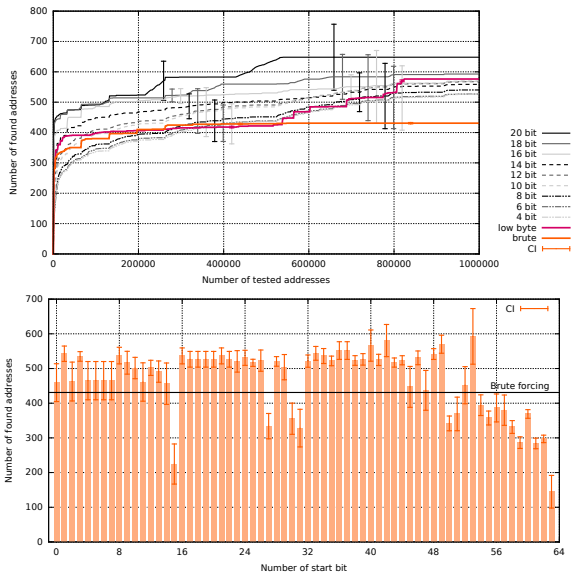

Fig. 2. *Servers:* Number of found addresses in dependence of probed addresses (top) and start bit (bottom). Comparison of our pattern-based algorithm, the *low-byte* pattern and brute-forcing.

in a larger data set represents the process of reconnaissance more adequately. Thus, we took the smaller set for training, the large for reconnaissance. For the creation of all subsets, we used *WEKA*'s *stratified remove folds algorithm* [18] and the addresses' LSB as classifier. The manually crafted patterns were evaluated by the same evaluation algorithm and on the same test data sets for comparability. The list of addresses were extracted from *scan6* by means of *tcpdump*, and another *Python* script extracting the addresses. In total, we ran ten runs per data set.

For our experiments, we were able to access three real-world data sets that represent a different node type each. *Client* addresses were gained from logs of RIPE's IPv6-enabled homepage *www.ripe.net*. In total we had 167 347 addresses. Requesting AAAA records of the *Alexa Top Million* revealed 16 644 *Server* addresses. Tracerouting the path to these servers revealed 12 982 distinct *Router* addresses. Both data sets were collected from a *Rackspace* cloud instance in the *Dallas* region. These data sets include addresses from a high number of organisations and are thus assumed to be representative for address assignment habits of administrators.

B. Results

Figures 1, 2 and 3 provide the results for the three host types routers, servers and clients. The upper sub-figure respectively shows the number of found addresses in dependence of the number of probed addresses for different thresholds. The results of our recursive algorithm are contrasted with the results of the low-byte pattern and brute-forcing[4]. The lower sub-figure shows the number of addresses found by the recursive algorithm in dependence of the start bit with a threshold of 18. The graphs average ten runs, and also provide the confidence interval (CI).

Routers: Our recursive algorithm outperforms brute-forcing and the low-byte pattern. The higher the threshold, the higher the number of found addresses. With a threshold of 20, the algorithm reveals plus 442 addresses in comparison to brute-forcing, and plus 353 in comparison to the low-byte pattern. A threshold of 4 still reveals plus 118 respectively 29 addresses. The low-byte pattern discovers 88 addresses more than brute-forcing. Considering initialization, bit 0 to 46 and bit 48 to 53 result in more than 800 discovered addresses. The maximum is 962 addresses (bit 45). Starting with bit 56 to 63 results in less than 620 addresses, the minimum is 395 addresses (bit 62).

Servers: Higher thresholds perform better, and our recursive algorithm outperforms brute-forcing. However, low thresholds (4, 6 and 8 bits) are below brute-forcing in the beginning, but are outstripping brute-forcing within the first third of probes. With a threshold of 20, the algorithm discovers plus 217 addresses in comparison to brute-forcing, and plus 72 in comparison to the low-byte pattern. The low-byte pattern reveals 145 more addresses than brute-forcing, and experiences a steep increase not only in the beginning, but also at about 500.000 probes. Considering initialization, the maximum is 593 addresses starting with bit 53. Its neighbor bits 52 and 54 however result in only 452 and 294 addresses, but bit 16 to 26 all result in more than 500 addresses. Results for bit 54 to 63 are below brute-forcing.

Clients: The recursive algorithm with thresholds above 16 reveals more addresses than brute-forcing and the low-byte pattern. With a threshold of 20, the recursive algorithm reveals plus 387 addresses in comparison to brute-forcing, and plus 417 addresses in comparison to the low-byte pattern. Noticeably, low-byte performs slightly worse than brute-forcing and reveals 31 addresses less. Considering initialization, only bit 17 and 19 to 21 provide more than 1000 addresses. Starting with bit 1, 14, 15 and 26 to 63 performs worse than brute-forcing.

Manual Patterns: The port pattern includes only 552 probes. Table I shows that these probes reveal 44 client, 53 router and 36 server addresses. The table compares these results to the number of discovered addresses in the first 552 probes of the low-byte pattern, brute-forcing and our recursive algorithm with a threshold of 18. All of them reveal roughly four-times the addresses of scanning with the port pattern, or even more.

[4]In this paper, *brute-forcing* is considered as probing addresses in ascending order and starting with the lowest.

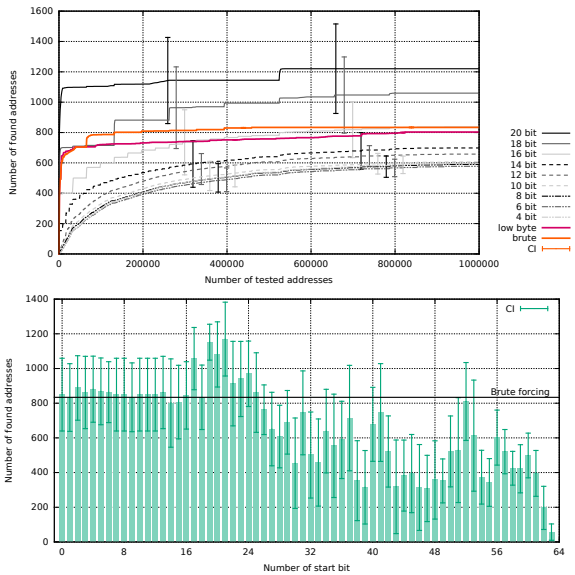

Fig. 3. *Clients:* Number of found addresses in dependence of probed addresses (top) and start bit (bottom). Comparison of our pattern-based algorithm, the *low-byte* pattern and brute-forcing.

	Routers	Servers	Clients
Port Pattern	52.6 (CI 1.0)	35.5 (CI 0.9)	43.8 (CI 2.0)
Low-Byte Pattern	204.1 (CI 0.8)	192.8 (CI 1.0)	209.6 (CI 0.8)
Brute-Forcing	203.1 (CI 0.8)	191.8 (CI 1.0)	208.6 (CI 0.8)
Our Algorithm	407.3 (CI 1.7)	274.3 (CI 1.7)	386.1 (CI 81)

TABLE I. PORT PATTERN IN COMPARISON TO ALTERNATIVES

	Routers	Servers	Clients
Top 1	8.1 (CI 0.5)	163.8 (CI 2.7)	707.4 (CI 6.0)
Top 2	6.3 (CI 0.5)	41.4 (CI 0.9)	311.0 (CI 3.8)
Top 3	4.7 (CI 0.3)	18.0 (CI 0.7)	147.4 (CI 2.5)
Top 4	4.5 (CI 0.4)	13.5 (CI 0.8)	140.9 (CI 2.4)

TABLE II. OUI PATTERN: BEST RESULTS

Table II shows the four most successful scanning attempts of the OUI pattern per host group. Every attempt requires 2^{24} probes. For routers and servers a low number of addresses is discovered: The most-frequent OUI reveals only 8 router addresses and 164 server addresses. The most-frequent OUI in clients reveals 707 addresses, but remains a singular result. The second-frequent reveals 311 address, and the remaining below 150 addresses. These are all rather low results considering the roughly 16 million probes per OUI.

C. Parameter Prognosis

An adversary using our pattern-based algorithm for host discovery likes to know in advance whether a certain start bit is a good choice because the algorithm's results are heavily dependent on the initialization. We claim that the results are dependent on the bit ratio as shown in Figure 4. This ratio indicates the part of addresses with this certain bit set to one. Low ratios provide better results, than ratios close to 50 %. Starting with such ratios of about 50 % postpones a pattern with a rather high number of appropriate addresses. This impedes our algorithm's intention of prioritizing frequent patterns.

Fig. 4. Bit Ratio: Ratio of addresses with a certain bit set to one.

	Routers	Servers	Clients
linear	2519	2148	9424
quadratic	2389	1974	9424
exponential	3641	2738	10109

TABLE III. RESIDUAL VARIANCES OF REGRESSION ANALYSIS

We performed regression analysis to investigate this. Quadratic regression fitted best among linear, quadratic and exponential approaches evaluated by the residual variance, as shown in Table III. The coefficient of determination *R2* is 0.58 for routers and 0.48 for servers. This means that roughly half of the scanning results variance is determined by the starting bit's bit ratio. Residual variances for client nodes are higher in comparison and regression fitting is of less quality. *R2* is only 0.13 for quadratic regression. The influence of the initial bit on the overall result is minor. Figure 5 shows the scatter diagrams for routers and servers respectively. Every crossing indicates a bit ratio and its related scanning result. The resulting quadratic regression curves are added to the scatter diagrams. We refrained from depicting regression for clients due to space constraints.

V. DISCUSSION

Our results on pattern-based scanning approaches are twofold: On the one hand, the manually crafted patterns that are known from the literature turned out to be of limited benefit. On the other hand, we proposed a pattern-based algorithm that automatically discovers patterns in a sample and generates addresses based on the found patterns. This novel algorithm is able to outperform brute-forcing and manually-crafted patterns. Focusing on manually-crafted patterns, the *low-byte* pattern solely performs well with servers, and is even worse than brute-forcing for clients. The *port* pattern finds a fourth or less nodes than all other approaches within their first 552 probes, and the *OUI* pattern typically finds less than half of the nodes of our algorithm but requires 16 times the number of probes. Return to our pattern-based algorithm, it outperforms the other evaluated approaches, but its overall performance is dependent on the start bit that is set during

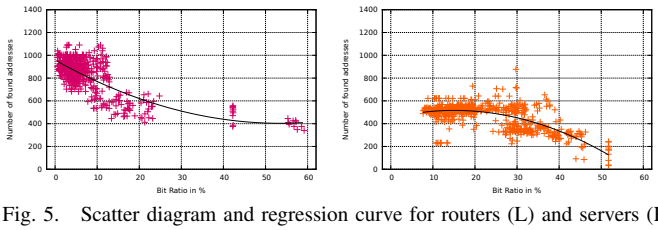

Fig. 5. Scatter diagram and regression curve for routers (L) and servers (R)

initialization. Taking a neighbor bit might already impact the performance negatively. However, we have shown that the algorithm's performance in dependence of the start bit can be pre-estimated by means of simple bit-wise statistics for servers and routers to a certain degree. Pre-estimation with respect to clients is of less quality, and might be a consequence of the high amount of random addresses generated by the privacy extension in the sample data sets. A removal of these temporary addresses from the data set is likely to improve the results on pre-estimation, but also on overall scanning results as their pseudo-randomness might negatively impact the pattern discovery. [15] proposes an algorithm to identify such addresses. Hosts using a temporary address are anyway also reachable via a stable address that might be even easier to guess.

The success of our pattern-based algorithm also highlights that address scanning is feasible with a comparable low effort due to address-inherent patterns, and defense-in-depth in the context of IPv6 reconnaissance seems incomplete; notwithstanding, that already early RFCs as of 2008 advise the assignment of addresses *"that are not obvious to guess"* [19] as low node density alone does not guarantee to be undiscoverable. This points the way towards mitigation of pattern-based scanning approaches. Random or pseudo-random addresses, e. g., [20], repel the threat of finding implicit patterns. Mitigation against our algorithm nevertheless requires more effort than against manually-crafted patterns. To mitigate the threat of manual patterns, choosing an address beyond the patterns is enough. For example, 2001:db8::5:21 is probed by the port pattern, but 2001:db8::6:21 is not.

VI. Conclusion and Future Work

In this work we proposed a new methodology for enabling scanning for active IPv6 addresses based on rule mining. The fundamental idea behind this approach lies in the observation that administrators do not select random addresses when migrating their services to the IPv6 world, but rather rely on patterns. While, opposed to IPv4, probing every single address is not possible for IPv6 due to the sheer amount of existing addresses, this approach uses prediction of patterns based on a sample set of addresses in order to rearrange the scanning order to enable faster retrieval of addresses. While this does not allow scanning the whole IPv6 range, this technique opens up great changes for fast retrieval of a large amount of used addresses. Contrary to approaches based on experience, this method allows the regular recalculation of the most likely patterns in order to detect changes in the typical selection of addresses. Future work includes the implementation of this techniques into a scanning tool, more results are needed with respect to the performance of the algorithm, as well as further research on speeding up searches.

Acknowledgment

The authors thank Rene Wilhelm (RIPE) for the provision of the data set. This research was funded by COMET K1, FFG - Austrian Research Promotion Agency.

References

[1] R. Graham, "masscan," https://github.com/robertdavidgraham/masscan, Accessed: 2015-03-04.

[2] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide scanning and its security applications," in *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[3] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices," in *Proceedings of the 21st USENIX Security Symposium*, 2012.

[4] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in *Proceedings of the 2013 Internet Measurement Conference*, 2013.

[5] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse," in *Symposium on Network and Distributed System Security*, 2014.

[6] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Exit from Hell? Reducing the Impact of Amplification DDoS Attacks," in *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[7] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer *et al.*, "The matter of Heartbleed," in *Proceedings of the 2014 Internet Measurement Conference*, 2014.

[8] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Internet Requests for Comments, RFC Editor, RFC 2460, December 1998.

[9] J. Ullrich, K. Krombholz, H. Hobel, A. Dabrowski, and E. Weippl, "IPv6 Security: Attacks and Countermeasures in a Nutshell," in *8th USENIX Workshop on Offensive Technologies*, 2014.

[10] T. Chown, "IPv6 Implications for Network Scanning," Internet Requests for Comments, RFC Editor, RFC 5157, March 2008.

[11] F. Gont and T. Chown, "Network Reconnaissance in IPv6 Networks," Work in Progress, Internet-Draft, January 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-opsec-ipv6-host-scanning-05

[12] G. Lyon, "nmap," http://nmap.org, Accessed: 2015-03-04.

[13] D. Leonard and D. Loguinov, "Demystifying Service Discovery: Implementing an Internet-Wide Scanner," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 109–122.

[14] Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-wide view of Internet-wide scanning," in *Proceeding of the 23rd USENIX Security Symposium*, 2014.

[15] D. Malone, "Observations of IPv6 addresses," in *Passive and Active Network Measurement*, 2008, pp. 21–30.

[16] F. Gont, "IPv6 Toolkit," https://github.com/fgont/ipv6toolkit, Accessed: 2015-03-04.

[17] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining, Pratical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[19] G. V. de Velde, C. Popoviciu, T. Chown, O. Bonness, and C. Hahn, "IPv6 Unicast Address Assignment Considerations," Internet Requests for Comments, RFC Editor, RFC 5275, December 2008.

[20] F. Gont, "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)," Internet Requests for Comments, RFC Editor, RFC 7217, April 2014.

APPENDIX A
PATTERN-BASED ALGORITHM

Our recursive algorithm for pattern discovery and address generation is depicted in Algorithms 1 and 2. This additional description emphasizes the textual representation of Section III.

Algorithm 1 `doRecursionWith` determines a further bit in every recursion.

Input: $pattern$ is a bit pattern with determined and undetermined bits

```
1: if count(undetermined bits) < threshold then
2:     iterateAddresses(pattern)
3:     return false
4: end if

5: rule = findBestRule(pattern)
6: pattern = apply(pattern,rule)
7: doRecursionWith(pattern)

8: alternativeRule = inverse(rule)
9: alternativePattern = apply(pattern,alternativeRule)
10: doRecursionWith(alternativePattern)
```

Algorithm 2 `findBestRule` finds the rule for the highest number of addresses.

Input: $pattern$ is a bit pattern with determined and undetermined bits

```
1: addresses = getAddressesWith(pattern)

2: for each undetermined bit in pattern do
3:     calculateSupportForRule(addresses, undet. bit = 0)
4:     calculateSupportForRule(addresses, undet. bit = 1)
5: end for each

6: return rule with highest support
```