An Iterative and Toolchain-Based Approach to Automate Scanning and Mapping Computer Networks

Stefan Marksteiner
DIGITAL - Institute for Information and Communication Technologies
JOANNEUM RESEARCH
Graz, Austria
stefan.marksteiner@joanneum.at

Harald Lernbeiß
DIGITAL - Institute for Information and Communication Technologies
JOANNEUM RESEARCH
Graz, Austria
harald.lernbeiss@joanneum.at

Bernhard Jandl-Scherf
DIGITAL - Institute for Information and Communication Technologies
JOANNEUM RESEARCH
Graz, Austria
bernhard.jandl-scherf@joanneum.at

ABSTRACT

As today's organizational computer networks are ever evolving and becoming more and more complex, finding potential vulnerabilities and conducting security audits has become a crucial element in securing these networks. The first step in auditing a network is reconnaissance by mapping it to get a comprehensive overview over its structure. The growing complexity, however, makes this task increasingly effortful, even more as mapping (instead of plain scanning), presently, still involves a lot of manual work. Therefore, the concept proposed in this paper automates the scanning and mapping of unknown and non-cooperative computer networks in order to find security weaknesses or verify access controls. It further helps to conduct audits by allowing comparing documented with actual networks and finding unauthorized network devices, as well as evaluating access control methods by conducting delta scans. It uses a novel approach of augmenting data from iteratively chained existing scanning tools with context, using genuine analytics modules to allow assessing a network's topology instead of just generating a list of scanned devices. It further contains a visualization model that provides a clear, lucid topology map and a special graph for comparative analysis. The goal is to provide maximum insight with a minimum of a priori knowledge.

CCS Concepts

•**Security and privacy** → **Network security;** *Systems security;* •**Networks** → **Network monitoring;** *Logical nodes;*

Keywords

Network Scanning; Security; Network Mapping; Networks

SafeConfig'16, October 24 2016, Vienna, Austria

ISBN 978-1-4503-4566-8/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2994475.2994479

1. INTRODUCTION AND MOTIVATION

Scanning and mapping a computer network is the crucial part of network reconnaissance [10]. As such, it is usually one of the first steps performed by an attacker[21]. Therefore, it is also useful for auditors or people in charge of network security to use the same techniques to examine their network in order to identify potential security weaknesses [22]. This overview differentiates in this context also network scanning and mapping. While the former generates solely a list of network devices, the latter generates a map, providing additional information about the network's structure, building the fundament to measuring security in the form of attack graphs [13]. To support this mapping task, this paper from an ongoing research project presents a widely automated approach for mapping non-cooperative, previously unknown computer networks and an expressive visualization of the obtained data. Within this paper, *unknown* means that there is only minimal a priori knowledge of the network, in particular the target network range to be scanned (or even only parts of it, like a single machine's address, from that range). *Non-cooperative* in this context means that the scanner does not have any administrative access (such as credentials).

Similar to testing applications with unknown internals, these conditions can also be subsumed as *black box scanning* [27, 1]. The benefit of black box scanning non-cooperative networks in security auditing and ethical hacking is threefold: firstly, it allows comparing the network documentation with the actual reality in a company's network. Secondly, undocumented, unauthorized and possibly malicious devices may be discovered [23]. And thirdly, as mentioned above, it provides an attacker's view [2]. All three of the above points (but especially the last one) benefit from an approach with only minimal a priori information (black box), as it prevents the security staff from taking paved roads in their analysis and, thus, from gaining the same results they already obtained beforehand by more white box oriented security evaluation forms. Also, network topology is among the needed information for creating an attack graph [24]. There is a broad variety of network and port scanners available (for instance nmap[1] or amap[2]), as well as specific vulnerability

[1] www.nmap.org
[2] www.thc.org/thc-amap

scanners (like Nessus[3] or OpenVAS[4]) or tools for further information gathering (as for example snmpwalk[5]). Unfortunately, the process of performing a thorough network reconnaissance (that results in a topology map) is, presently, rather manual and time-consuming.

Therefore, this paper addresses the problem of how aforementioned network scanning tools can be most effectively combined and their scanning results analyzed and graphically represented. In order to fulfill this objective, the following questions need to be clarified:

- How can the scanning process be automated and controlled?
- How can the results be automatically analyzed?
- How can the results be best graphically represented in a scalable and lucid form?

The third question has the additional constraint that large networks should be presented in an aggregated manner, but still leave the basic topology visible at a glance , which existing solutions (see Section 2) have proven not to meet.

Also, as one of the most relevant aspects in analyzing computer networks is discovering and visualizing the network's structure, the second question contains the major problem of augmenting the obtained data with structural information. Modern computer networks consist of nodes (hosts and routers) and edges (links) that are structured in a hierarchy, which means that their structure can be represented as a tree (that is, as a directed acyclic graph with a single root) as defined by graph theory [11]. Therefore, the pivotal element of determining a node's place within that structure is finding the edge to its parent node or, in computer networking terms, its default gateway. This leads to the following subquestion that defines the step from list-like network scanning to actual network mapping by adding hierarchical context to the scans:

- How can a node's default gateway be determined?

This problem statement applies, in principle, to all computer networks. However, the presented solution is shaped to fit to modern day's organizational (in other words: classical) information and communication technology (ICT) networks. Specific requirements for other types of networks (for instance within industrial control systems or the Internet of things) are out of scope of this work. A main concept used to automatize the scanning process is chaining these popular network scanning tools (see Section 3). Tool chaining is a well-known concept in information technology and also (although less commonly) used in approaches to network scanning in order to yield better results than there could be gained from a single scanning tool (see Section 2). The presented solution further develops this technique by enhancing the versatility of the scanning process introducing *scanning policies*. These policies contain customizable toolchains improved with an iterative scanning model, allowing each pass of the said toolchain to benefit from the results obtained by its predecessor (see Section 3.4). Although such a concept is used in white box network mapping (often called *network discovery* in this context), the approach is quite novel to black box network mapping (see Section 2). Scanning policies further integrate external scanning tools and genuine analytics to automatically interpret the scanning results, yielding a mostly automatically generated topology model (see Section 3). Furthermore, as the analysis and subsequent graphical representation still are less well elaborated parts in mapping non-cooperative computer networks in general, this approach also includes a visualization concept for the data obtained through scanning and the contextual enhancements generated by the analytics modules (see Section 4). The visualization concept includes automatic grouping of the data (grouped by operating systems or network segments), while leaving network infrastructure nodes, and therefore the overall topology, visible. It also features a comparative view, allowing documenting network evolvement over time and testing of access rules. Apart from an approach to solving the questions stated above, this work also provides a proof of concept in the form of a software prototype of the presented concept (see Section 5) and scan results of real-world networks (Section 6). Section 7, eventually, gives a conclusion and outlook to further research.

2. RELATED WORK

There is a known approach to toolchaining in black box scanning that has some similarities in terms of using adapter plugins for scanners and splitting normalization and aggregation [9]. That work focuses on building an integrated scanning tool and merging the obtained information in a common structure, while the presented research, in contrast, does not merge the information but displays each scan result distinctly in order to distinguish which module found which information. The main difference, however, is that this paper's approach focuses on network mapping (including scanner analytics and visualization) instead of the scanning part and uses an iterative model. The visualization concept in this paper is the result of an evaluation (see Section 4) of a variety of commonly known algorithms [28, 15, 25, 14, 17, 7, 12], as commonly used graphical scanner interfaces (such as *Zenmap*[6]) have proven to be insufficient for network mapping. Using an iterative model, on the other hand, is common in commercial and open source solutions for credential-based white box network mapping[7], but rarely used in black-box scanning. In fact, only one approach came to the authors' attention [3], which again does merely scan but not map unknown networks.

3. TOOLCHAINS

This section describes a concept for tool chaining in the context of computer network scanning and mapping. In this approach, the building blocks of such toolchains are, in principle, scanner modules and analyzer modules. The two differ mostly in two points: their purpose and their input method. Firstly, while scanner modules represent existing network scanning tools, the purpose of analyzer modules is to provide context for the scanned data. Secondly, scanner modules get external input from their respective scanning tools, while analyzer modules merely operate on the present data without external input. As seen in Figure 1, these two elements are the building blocks of a *scanning policy* which,

[3]www.tenable.com/products/nessus-vulnerability-scanner
[4]www.openvas.org
[5]www.net-snmp.org/docs/man/snmpwalk.html
[6]www.nmap.org/zenmap
[7]often called network discovery in this context, particularly in solutions for network management

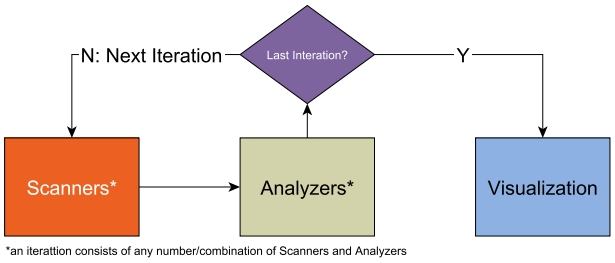

Figure 1: Schematic example toolchain with iterations

within this approach, represents a toolchain and the number of iterations the chain is passed through (see Section 3.4). Using policies, toolchains are not static and can be adjusted according to specific needs, as different toolchains might yield different results (even the same tools used in a different order). The same applies to starting options for both scanner and analyzer modules. This flexibility allows choosing the most effective policy for each case (for instance, different chains for small and big networks).

3.1 Scanner Modules

A scanner module controls an external network scanner. Its tasks are therefore handling the scanning tool's input (tool call) and output (result collection), converting the output into a common format (normalization) and storing it in a database (storage). This process is the same for every module; only the details differ (except for the storage, see Figure 2). Calling a tool could happen either through command line calls, an API or by incorporating the tool's source code directly into the module's. Either way, a module is responsible for enabling all sensible (in the context of a toolchain) options its tool provides, for defining additional ones (mostly optimizations for better interlocking of tools within a toolchain) and for using some options per default (for example output formatting options). Consequentially, the next step is collecting the results delivered by the scanner software, either through a return value (if called by code or API) or (if otherwise) by reading a file or console output. These results then must be normalized to correspond to each other. As it provides a broad range of already present attributes, this solution uses the *nmap xml output* as a basis [19]. Attributes not present in this format are incorporated as needed by other modules. For storage reasons stated in Section 5, the normalized data is subsequently transferred into the *JavaScript Object Notation (JSON)* [5]. The combination of obtained data consists of two components: normalization (see above) and aggregation (linking the different modules' results together). In this approach, the aggregation is a matter of the visualization engine. The data model creates a separate object per tool for each found node in order to make it clearly distinguishable which scanning tool yielded which results. This is the reason why scanner modules only handle the normalization and not the aggregation. In general, newer results take precedence over older ones (except for manual edits). The storage, eventually, basically consists of writing the obtained results into a database.

3.2 Analyzer Modules

In contrast to scanner modules, analyzer modules do not

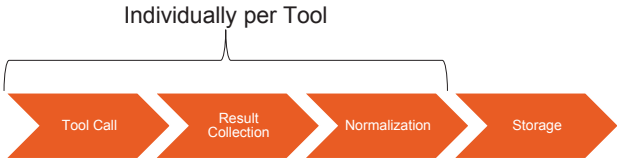

Figure 2: Execution trail of a scanner module

use external data, but only information provided by scanner modules. Therefore, a toolchain containing only analyzer modules will not produce any results. The purpose of analyzer modules is interconnecting information and providing context. These modules generate database entries containing the analysis results, which again can be used by subsequent modules, in later iterations or by the visualization engine. The main reason not to incorporate the analytical logic within scan modules is that data from several scanners might be used as input for a specific analysis. For instance, network trace information might come from an arbitrary *traceroute* implementation, as well as an *nmap* (or other trace-capable) scanner module. The following subsection gives an example for an analyzer module.

3.3 Estimating Default Gateways

Estimating default gateways is the necessary stepping stone to overcome the gap between network scanning and network mapping. As the required data to achieve this goal might be present within the acquired data but has to be interpreted correctly, this task poses an archetype for an analyzer module in the sense of the presented concept. To fulfill this task, this work proposes three methods to determine a host's default gateway $DGW(H)$:

- *Estimation by trace*: Determining predecessor information in a network trace[4];
- *Estimation by singleton router*: Assessing if only one router is present in a scanned network;
- *Estimation by usual suspects*: Watching for addresses frequently used for gateways.

The first method naturally requires a list of hops (h) obtained through traceroute from a scanner in the toolchain. If the host is in the same subnet (hopcount $n = 1$), the host will be attached to the scanning host's own default gateway $DGW(X)$, which is determined using system functions. Otherwise, the default router of the host is set to the last hop before the host (h_{i-1}, see Formula 1).

$$i : \{1, .., n\}; DGW(H) = \begin{cases} DGW(X) & \dots \quad n = 1 \\ h_{(i-1)} & \dots \quad n > 1 \end{cases} \quad (1)$$

As the path towards the target network holds no information about its topology and is therefore of less interest, an algorithm identifies the last common gateway of the network. This *network entry point* (from the scanner's perspective) of all hosts $NEP(H)$ serves as a central connection node that links different parts of a graph together (for an example, see Section 6). Said algorithm traverses all of the trace paths $h_i(H_j)$ in parallel from their beginning and compares if they are all equal at the respective hop position. The last common router is the router at one position (h_{i-1}) before they begin to diverge, which means that at least one path differs

at a hop position ($h_i(H_j) \neq h_i(H_{j+1})$. Therefore, if not all paths contain the same address value at hop position x, the last common gateway is x-1. If hosts from the scanner's network are present in the target (with $n = 1$), the result is the scanning machine's default gateway (see Formula 2).

$$\begin{aligned} i : \{1, .., n\}, j : \{1, .., m\}; \\ NEP(H) = h_{i-1} \leftrightarrow h_i(H_j) \neq h_i(H_{j+1}) \end{aligned} \quad (2)$$

The second method yields a positive result if exactly one device of some *router* type is present in the dataset. This requires operating system (OS) detection and classification capabilities by a scanner in the toolchain. For instance, nmap provides this feature in its *nmap-os-db*. The analyzer module examines this data for router devices. The third method is similar to the second one in terms that it also needs the same information to be present and searches for *router*-type devices. In contrast to the latter, it uses the first three *dotted decimal* groups of IPv4 target addresses and adds commonly used router addresses (e.g. *.1* and *.254*) as final part. These composites form the addresses that are evaluated to be a router. Further, for multiple targets, duplicates are eliminated. If such a *usual gateway address* is in fact a router according to OS detection, there is a distinct probability of being a default gateway for the scanned network.

The first method is more reliable (as a traceroute predecessor is more likely a default gateway router than an arbitrary one found on the same network), but the success of these methods obviously depends on the present information and, thus, ultimately on the target network's configuration. As the latter is unknown beforehand, it cannot be known a priori which method is more successful. As none of the estimation methods provides assured detection, the opportunity for manual assignment and correction (through user interaction, superseding automated scan results) is included in this concept.

3.4 Iterative Scanning

The basic idea described in this section is to let the results of one iteration determine the input for the next one [3]. To do so, every module within the toolchain implements its own method for gaining *seed* data to be used in the next iteration. This method can differ greatly between modules. For instance, one tool might use network trace data, while another scanning module might use data gained via *SNMP* or *ARP* (Section 6 shows a practical example). The default gateway estimation module (see Section 3.3), for instance, adds identified intermediate hops that are not yet scanned to the seed database. This usually yields an expanded scan area, for intermediate hops frequently feature IP addresses from outside the target scope. Subsequently, all newly found hosts will serve as targets for the next scan iteration, thus being processed by all tools within the chain (see Figure 1). Together with the respective toolchain, the number of iterations forms a scanning policy.

4. NETWORK VISUALIZATION

Due to the potential of visualizations [6] and the speed of visual perception [18], it seems appropriate to present a computer network in graphical form. Various solutions for automatically generating a graphical view of a network hierarchy through a computer program have been proposed (see Section 2). In order to find the appropriate algorithm

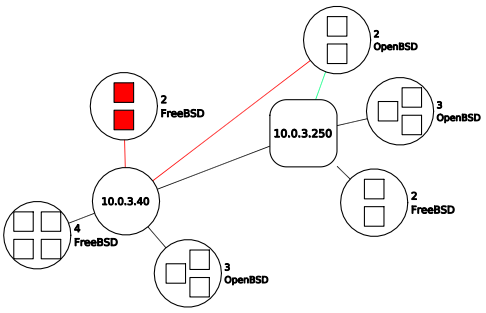

Figure 3: Compare view showing gateway change and node removal

for computing the graph's layout the following requirements have to be considered:

- Suitability for typical organizational network sizes;
- Economic space usage, even for unbalanced hierarchies;
- Edges shall contain as few bends as possible;
- Change tolerance regarding the overall layout.

Using the Tulip graph analysis program, layout algorithms for hierarchies were evaluated on these requirements with data collected from a scan of a part of the Joanneum Research computer network, as well as with generated and manually edited data. Most promising candidates were:

- The Tree Radial algorithm [14];
- The Balloon algorithm [17, 7];
- The Bubble Tree algorithm [12].

The first algorithm's adequacy depends on the parameters for node spacing and layer spacing, while the second works well for rooted trees if not enforcing equal angles, but has deficiencies in usage of available space. The Bubble Tree algorithm shows the best overall results, for the graph is laid out with good usage of screen space, edges contain at most one bend and the overall layout proved to be quite insusceptible to minor changes. All of these layouts, however, do not set focus on a network's structure as they value the structure-defining intermediate nodes (routers) the same as the usually vastly outnumbering leaf nodes (hosts). To emphasize the former, similar leaf nodes are aggregated and depicted only by one representative symbol (bubble) with a single edge. This aggregation could fund on similar characteristics (like the same OS) or on a threshold of the size of leaf node groups (see Figure 6 for a practical example). Further, leaf nodes could be hidden completely and only displayed on demand. For auditing and evaluating a network, not only a snapshot but also displaying deltas of a network is useful in order to document its evolvement or evaluate the effectiveness of network access control. To do so, an adapted version of the network graph view is used, highlighting the changes (red for removals and green for additions, see Figure 3 for a concrete example).

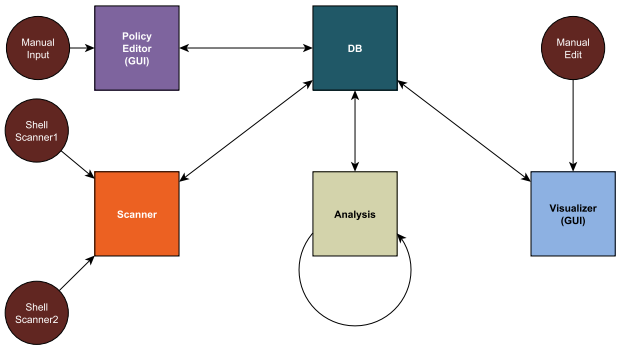

Figure 4: Diagram of the module interaction

5. SOFTWARE PROTOTYPE

This section describes a software prototype implementing the concepts illustrated above in C++, resulting in a tool for Kali Linux[8]. This software, called *Tactical Network Mapper (TNM)* uses MongoDB[9] and the Tulip graph library[10] for storage and visualization purposes, respectively. The main libraries of the software contain only a framework, while the plug-ins provide the functionality, resembling the structure of the Eclipse[11] software development environment. All the data exchange between the components (scanner and analyzer modules, and user interface including policy editor and the visualization engine) happens via the database (see also Figure 4). This couples these components very loosely, allowing a very flexible plug-in structure. As a result, the built solution allows integrating rather heterogeneous modules, making it an adequate testbed for both the current and future concepts. As the scanning process is iterative and a user might also manually revise the results and make changes, not only the most recent information might be relevant. A user could want to view changes between iterations and possibly roll back to a previous state, as well as to manually correct data (e.g. in case of wrongly assigned default gateways). These requirements point towards versioning as used in Revision Control Systems [26]. There are two main approaches to versioning: *snapshot* and *changeset (delta)* [16]. The former method stores a complete copy of the data set for each version. The latter saves storage space, but requires the construction of data sets in order to get a certain version. The changeset group can be further classified into *forward delta* and *reverse delta*. With reverse delta, always the latest version serves as reference for differences [29]. Despite earlier versions might be needed for comparing, it is reasonable to assume that most users will be interested in the most recent version of the data. Therefore, the presented solution tries balancing the benefits of both approaches by using *snapshots* for single objects, but *reverse delta* storage for the data set as a whole. This can be easily implemented by storing full snapshots of changed objects (and only those), with the most recent data set as reference. Through using MongoDB, which uses a binary-encoded *JSON* format (*BSON*) for storing the data [20], no additional conversion of the normalized data is needed.

[8]www.kali.org
[9]www.mongodb.com
[10]tulip.labri.fr/TulipDrupal/
[11]www.eclipse.org/ide/

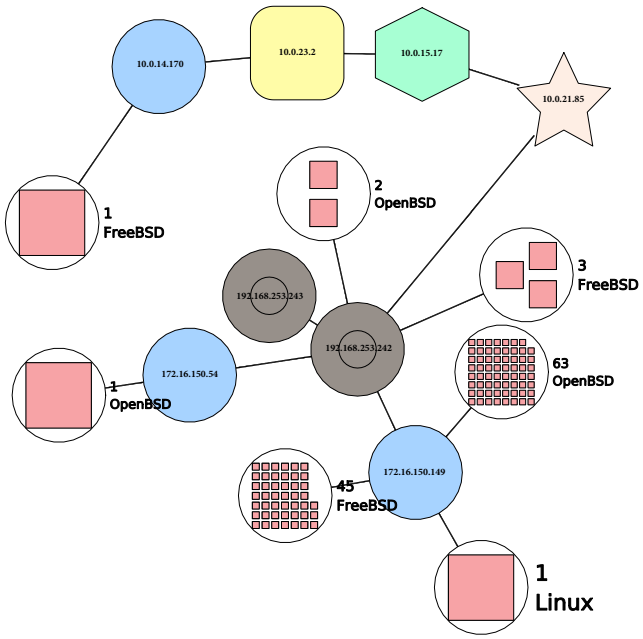

Figure 5: Graph of an anonymized external actual scan

6. RESULTS

As the concept (in form of the TNM implementation) is put to test, the results expectably differ according to different target settings. Obviously, the obtained results are more detailed when scanning from inside a network. This is a practical scenario for internal audits. Despite some natural restrictions (access rules enforced by firewalls), TNM was able to perform on external networks too. Figure 5 shows an anonymized example of a scan executed on an external, unknown (that is with only the target range known) network using an *nmap* module with traceroute and OS detection and the default gateway analyzer. The scanning machine was attached to a router not shown in the graph (according to the algorithm described in Section 3.3). Through the iterative model, TNM also discovered parts of the network that where originally not part of the target specification, which causes the different network portions of the IP addresses seen in Figure 5.

Another example for the use of iterations can be seen in Figure 6, which shows three iterations of an internal scan starting with a single machine. The bigger graph in the middle shows the third iteration, while the adjunct graphs in the bottom right and top left corners show the first and second iterations, respectively. This scan used a module that implements an snmpwalk scanner. The toolchain contains a TCP *nmap* scan containing traceroute and OS detection, the default gateway analyzer, a second *nmap*, scanning for the SNMP port (*UPD/161*) and the snmpwalk module that is executed on found snmp hosts. As many products are shipped with a *public* community, this could be used to discover devices reading out a device' ARP table, which was assumed in this example [8] (simulated by using a known write-only community). Clearly visible, each of the three iterations discovers significantly more devices than the last one (2, 7 and 856 nodes, respectively), eventually resulting

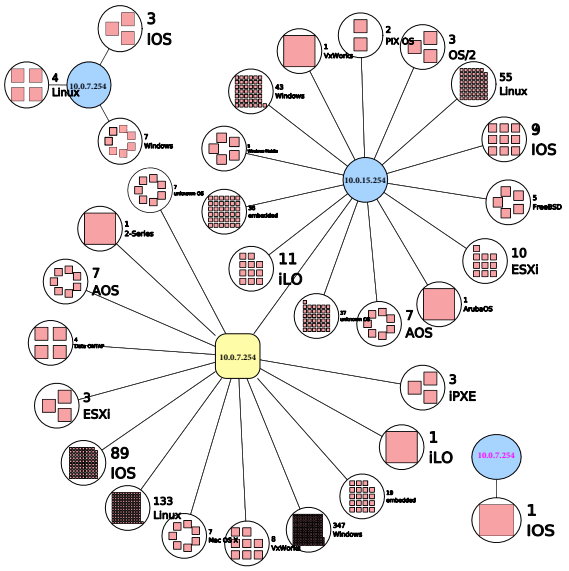

Figure 6: Comparison of three iterations of an anonymized actual scan result graph

Table 1: Runtimes per module and iteration in seconds compared to found nodes

Iteration	**1**	**2**	**3**
NmapScanner(-A)	7	27	1114
DefaultGatewayAnalyzer	<1	<1	<1
NmapScanner2 (UDP:161)	<1	2	8
SnmpwalkScanner	<1	43	797
Modules	7	43	1920
Total	7	44	1996
Number of found nodes	2	15	856
Nodes per second	0.29	0.34	0.43

in a tree of depth three. An additional scan on a public */19* network conducted with TNM (same policy as above) yielded 673 reachable hosts, 16 open SNMP devices, but no communities could be found.

Naturally, the number of discovered items has an impact on an iteration's runtime. The first iteration of said test took seven seconds to finish, the second already took forty-four, while the last iteration took over half an hour (1996 seconds). Table 1 shows an example table for test results (discrepancies between module and total runtimes are mostly database-induced delays). It can also be seen that, the ratio of time consumption and results (found nodes per second) improves significantly at each iteration. An iteration's scope allows a rough estimation of the upper boundary of its runtime, assuming no unforeseen events (for example network congestion) occur. More precise estimations on the runtime of an iteration are not possible a priori, as it depends on the result (the same applies for the overall process).

7. CONCLUSION

The presented results demonstrate the successful automated topology mapping of given computer networks (both internal and external[12]), without information beyond the target range. Especially, no credentials are needed (although utilized, for example in SNMP scanning). They further incorporate additional information about these nodes (operating system, open ports, et cetera) into the produced interactive map. This combination means a significant advancement compared to drawing topology maps and maintaining the result lists of unknown networks manually, using scanning tools and their result lists. It is also a novel approach, as no other known concept combines iterative toolchaining with analytics to conduct black box network mapping and tailored state-of-the-art visualization. By doing so, the presented concept could not only be used in practice to perform security audits and discover differences between documentation and reality as well as map undocumented networks, but also as a documentation tool by itself. The test results show that the concept is capable of not only mapping a network, but also discovering areas not found using less sophisticated methods. Furthermore, they demonstrate that using more iterations yields a more effective scanning performance (more found nodes per second of runtime).

Despite promising results, there is room for improvement. Apart from additional scanning modules, one possible upgrade is a module that allows the software to reconfigure the network address of the scanning machine by using data from passive scanners (or *network sniffers*). This way, the solution is capable of mapping a network it is directly attached to completely without prior knowledge (eliminating the need for a scan target). Useful for this improvement (but also for other cases) is a (yet work in progress) module, that tries to guess network boundaries (partially by using publicly available data like *whois databases*). Another work in progress module is an extension that fetches data from vulnerability databases, like the *National Vulnerability Database (NVD)*[13], and automatically compares the contained vulnerable services and operating systems with data from the scan results to facilitate vulnerability analysis. Another module could use graph theoretical methods to gain knowledge about the network's structure. Further, the concept could be tested in and adjusted to other environments like industrial control and Internet of things networks.

Finally, the resulting TNM software itself could also serve as a research tool. Possible studies carried out using TNM include the evaluation of the performance of different combinations of scanning tools under different conditions (for example with different network sizes). A prerequisite for such studies is the definition of sensible *key performance indicators (KPIs)* for network scanning and using respective toolchains under different conditions. Also, surveys on networks suggest themselves, like typical structures depending on network sizes or the rate of open snmp communities in average structures.

8. ACKNOWLEDGEMENTS

This research has been conducted on behalf of the Austrian Federal Ministry of Defense and Sports (BMLVS), Joint Command Support Centre (FüUZ), section ICT technology, department ICT security and funded by the Department for Science, Research and Development (WFE). We want to thank all partners, especially Lambert Scharwitzl, Christina Buttinger, Michael Pfister and Clemens Edlinger for their support.

[12]Although there are be limits by filter mechanisms and NAT

[13]nvd.nist.gov

9. REFERENCES

[1] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the art: Automated black-box web application vulnerability testing. In *2010 IEEE Symposium on Security and Privacy*, pages 332–345, May 2010.

[2] E. Bou-Harb, M. Debbabi, and C. Assi. Cyber scanning: a comprehensive survey. *Communications Surveys & Tutorials, IEEE*, 16(3):1496–1519, 2014.

[3] B. Boyter, R. Engelbach, and R. Taylor. System and method for network security scanning, Nov. 13 2003. US Patent App. 10/249,666.

[4] S. Branigan, H. Burch, B. Cheswick, and F. Wojcik. What can you do with traceroute? *IEEE Internet Computing*, 5(5):96–, Sep 2001.

[5] B. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, Internet Engineering Task Force, 2014.

[6] R. A. Burkhard. Learning from architects: the difference between knowledge visualization and information visualization. In *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*, pages 519–524, July 2004.

[7] J. Carriere and R. Kazman. Research report. interacting with huge hierarchies: beyond cone trees. In *Information Visualization, 1995. Proceedings.*, pages 74–81, Oct 1995.

[8] P. Chatzimisios. Security issues and vulnerabilities of the snmp protocol. In *1st International Conference on Electrical and Electronics Engineering*, pages 74–77, 2004.

[9] F. Cheng, S. Roschke, and C. Meinel. An integrated network scanning tool for attack graph construction. In *Advances in Grid and Pervasive Computing*, pages 138–147. Springer, 2011.

[10] S. Convery and B. Trudel. Cisco safe: A security blueprint for enterprise networks. Technical report, Cisco Systems, 2000.

[11] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632, Internet Engineering Task Force, 2006.

[12] S. Grivet, D. Auber, P. J. Domenger, and G. Melancon. *Computer Vision and Graphics: International Conference, ICCVG 2004, Warsaw, Poland, September 2004, Proceedings*, chapter BUBBLE TREE DRAWING ALGORITHM, pages 633–641. Springer Netherlands, Dordrecht, 2006.

[13] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 121–130, Dec 2006.

[14] T. J. Jankun-Kelly and K.-L. Ma. Moiregraphs: radial focus+context visualization and interaction for graphs with visual nodes. In *Information Visualization, 2003. INFOVIS 2003. IEEE Symposium on*, pages 59–66, Oct 2003.

[15] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on*, pages 284–291, Oct 1991.

[16] A. Koc and A. U. Tansel. A survey of version control systems. In *The 2nd International Conference on Engineering and Meta-Engineering: ICEME 2011*, Orlando, 2011. International Institute of Informatics and Systemics.

[17] C.-C. Lin and H.-C. Yen. *Graph Drawing: 13th International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005. Revised Papers*, chapter On Balloon Drawings of Rooted Trees, pages 285–296. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[18] H. Liu, Y. Agam, J. R. Madsen, and G. Kreiman. Timing, timing, timing: Fast decoding of object information from intracranial field potentials in human visual cortex. *Neuron*, 62(2):281 – 290, 2009.

[19] G. Lyon. *Nmap Network Scanning: Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure.Com, LLC, 2008.

[20] P. Membrey, E. Plugge, and D. Hawkins. *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Apress, 2011.

[21] J. Mirkovic and P. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[22] C. Muelder, K.-L. Ma, and T. Bartoletti. Interactive visualization for network and port scan detection. In *Recent advances in intrusion detection*, pages 265–283. Springer, 2005.

[23] A. Orebaugh and B. Pinkard. *Nmap in the enterprise: your guide to network scanning*. Syngress, 2011.

[24] C. Phillips and L. P. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

[25] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, March 1981.

[26] N. B. Ruparelia. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35(1):5–9, 2010.

[27] D. A. Shelly. Using a web server test bed to analyze the limitations of web application vulnerability scanners. Master's thesis, Virginia Polytechnic Institute and State University, 2010.

[28] M. Ward, G. G. Grinstein, and D. Keim. *Interactive data visualization : foundations, techniques, and applications*. CRC Press, Boca Raton, 2015.

[29] R. K. Wong and N. Lam. Managing and querying multi-version xml data with update logging. In *Proceedings of the 2002 ACM symposium on Document engineering*, pages 74–81. ACM, 2002.