Toward correctness by construction for network security configuration*

Daniele Bringhenti, Riccardo Sisto, and Fulvio Valenza†

Politecnico di Torino, Turin, Italy
{daniele.bringhenti, riccardo.sisto, fulvio.valenza}@polito.it

Abstract

Cyber attacks have been continuously becoming more frequent and powerful in modern computer networks, which interconnect a huge number of devices, including mobile ones. A main reason explaining the effectiveness of the attacks lies in the errors introduced by human administrators in the configuration of network security functions such as firewalls and VPN gateways. In literature, formal methods were used to check if a security configuration satisfies the policies describing the security properties to be enforced in the network. However, most existing proposals use a-posteriori formal verification, which still requires multiple tasks to be carried out by human administrators and may be time-consuming. In order to overcome these limitations, we have carried out research toward applying correctness-by-construction approaches for network security configuration. In this paper, we survey our most relevant contributions to this area, describing techniques based on constraint programming, like MaxSMT formulations, for a formal representation and resolution of the automatic security configuration problem, in such a way that the computed result is already proved to be correct and compliant with the requested policies.

Keywords: network security configuration, formal verification, correctness by construction

1 Introduction

In recent years, cyber attacks against computer networks have grown in power and complexity. As reported by various sources [13, 21], nowadays, modern network attacks have a shorter duration and rapidly mutate between multiple attack vectors within minutes. Cloudflare reported [13] that in 2022 96% of Distributed Denial of Service (DDoS) attacks remain below 500 Mbps, and with 52% of the total attacks lasting less than 10 minutes. Also, the news of the attack undergone by Proton in 2022 [21] showed evidence of burst DDoS attacks, with multiple attack vectors and rapid changes within minutes. The effectiveness of these attacks is even higher in mobile computer networks [15] and in mobile Internet, where many devices are connected simultaneously and can be exploited to amplify the power of an attack.

Unfortunately, many attacks are successful because they exploit human errors related to the configuration of the network security functions that should prevent them, such as firewalls and Virtual Private Networks (VPNs). According to the most recent Data Breach Investigations Report produced by [23], which yearly analyzes the characteristics and impact of cyber incidents and data breaches, in 2023 the third most common error type exploited by cyber attacks was indeed misconfiguration. Such an issue can be explained by multiple reasons. On the one hand, some companies may wrongly decide not to invest adequately in the cyber security training of their employees, because of lack of awareness about all the problems related to

*Proceedings of the 8th International Conference on Mobile Internet Security (MobiSec'24), Article No. 62, December 17-19, 2024, Sapporo, Japan.

†Corresponding author

unstopped attacks. On the other hand, fallibility is undeniably an intrinsic characteristic of human behavior, and also expert people may make unintentional misconfiguration mistakes, especially under conditions of stress or deadlines.

Formal verification is an effective solution to mitigate the consequences of network security misconfigurations. Specifically, formal methods can be employed to verify if a certain security configuration satisfies network security policies, where a policy is a statement describing a security property that must be enforced and fulfilled in the network. For example, network security policies may request to block malicious traffic flows, encrypt a communication to provide confidentiality, guarantee the integrity of some network packets, or check that access control rules are not violated. Unlike simple algorithms that may be developed to perform these checks, formal methods provide formal assurance of the result of their verification operations, thus boosting the confidence of humans in using them and consequently assisting them in avoiding avoidable configuration errors.

However, in literature, most of the proposed formal solutions provide a-posteriori formal correctness. The human administrator first creates the security configuration, then the formal technique is applied to verify if such configuration satisfies the requested security properties (e.g., confidentiality, authentication or integrity of the communications). Examples of methods that can be used for this purpose are model checking and theorem proving. Even though they successfully find misconfigurations, their problem is that they find them after they have already been included. Therefore, later, the administrator must fix them and re-run the verification method, hoping to have them fixed correctly, because if new issues are found, the whole sequence of actions must be repeated again.

In order to overcome these limitations of the state of the art, in the last years we have worked in the direction of a-priori formal verification, which could introduce correctness by construction in network security. The idea is to produce a security configuration automatically from the policies themselves, in a way that this configuration is already correct without requiring the application of other a-posteriori formal verification methods, thanks to the intrinsic correctness of the algorithms used for its automatic configuration. In this paper, we provide a short survey of our proposals in this context, describing how we formulated different network security configuration problems with constraint programming, i.e., a paradigm that allows modeling problems with constraints where some functions and variables are left open, so that their values can be established automatically by automated solver employing provably correct algorithms.

These proposals have been designed for generic computer networks, so they can also be applied to enforce security for mobile Internet. In fact, mobile networks are dynamic, with devices constantly joining and leaving the network, making security configurations highly prone to errors. Misconfigurations of network functions (such as firewalls and VPNs) are a primary target for attackers, who exploit these weaknesses to gain unauthorized access or disrupt services. Therefore, the correctness-by-construction techniques proposed in this paper avoid configuration errors by automating the setup to ensure compliance with security policies. Some of them can also be tuned for optimization purposes, e.g., to minimize the configuration rule sets, as mobile devices and networks often have limited resources, requiring that security configurations be optimized for efficiency.

The remainder of this paper is structured as follows. Section 2 describes the main characteristics and differences of a-posteriori formal verification and correctness by construction. Section 3 surveys our research about correctness by construction for network security. Section 4 concludes the manuscript and discusses future work.

2 Background

In the field of network security configuration, formal verification aims to prove that the configuration produced for functions such as firewalls and VPNs satisfied the policies describing how the network should be protected against cyber attacks. The application of formal methods also requires that all the elements of the verification problem (e.g., the network topology, the behavior of each network function, and the policies themselves) must be formally modeled in a way that adequately represents their counterpart real entities.

Two main paradigms can be pursued to provide formal correctness for network security configuration: a-posteriori formal verification and correctness by construction.

2.1 A-posteriori formal verification

A-posteriori formal verification techniques are used to check if an already created configuration complies with the requested policies. Traditional examples are model checking, theorem proving, and constraint programming.

In model checking, after a finite-state model is designed for the network and the policies, an algorithm exhaustively explores all possible states of this model so as to systematically check whether some formal security properties hold for that model in the different states. For example, model checking can be used to verify that a firewall rule list blocks traffic flows that are considered malicious according to some isolation policies, or that a distributed routing protocol adheres to security requirements such as message integrity and confidentiality. If vulnerabilities or violations are found, the model checker can provide counterexamples to help identify and address the root cause of the security flaw.

In theorem proving, formal mathematical proofs are developed to show that the model of the problem components satisfies certain the properties specified by the requested security policies. For example, theorem proving can be applied to protocols like Transport Layer Security (TLS) to ensure that data is encrypted and authenticated per the protocol's specifications. Unlike model checking, which automatically verifies properties, theorem proving often requires human intervention to construct and guide the proof, making it more suited for critical systems where rigorous security guarantees are necessary.

In constraint programming, a problem is formulated as a set of constraints, which are declaratively stated as the feasible solutions of the problem itself for a set of decision variables. These constraints do not correspond to the typical primitives of imperative programming languages. Actually, they do not represent a sequence of algorithmic operations to be executed, but instead, they must represent the properties of a solution to be found, i.e., the ones requested by the network security policies. Possible problem formulations based on constraint programming are the traditional SATisfiability (SAT) problem, where all variables are Boolean, or the Satisfiability Modulo Theories (SMT) problem, where the constraints may include variables and operations related to other theories than only the Boolean one, such as integer numbers, real numbers, lists, arrays, bit vectors, and strings. For example, the bit vector or integer theories are suitable for modeling firewall rules, as the IP addresses defined in their conditions may be represented as a sequence of bits or alternatively as four integer numbers representing the four bytes in dotted notation.

In literature, several research papers ([16, 17, 22, 24, 18, 25, 2, 19]) proposed formal approaches for network security by adopting a-posteriori formal verification techniques, such as the examples mentioned before or even more advances ones. In greater detail, HSA [16] presents a model according to which each data packet can be represented as a point in the header space,

and then it applies header space analysis. NoD [17] improves header space verification in Datalog, so as to verify reachability and isolation properties on complex paths (e.g., when the traffic between two hosts flows through a service function). SymNET [22] uses symbolic execution to analyze the behavior of networks, including stateful networks. NetSMC [24] proposes a new language for policy specification, so as to define formal models for many security properties. Anteater [18] converts the data plane information of network functions into Boolean expressions, used for the definition of a SAT problem. Gravel [25] verifies the behavior of middleboxes, by converting the source code representing it into SMT problems. Finally, Verigraph2.0 [2] and APT [19] formulate SMT problems to provide formal assurance about the reachability or isolation of traffic flows.

The main limitation of a-posteriori formal verification is that it applies only to an existing network security configuration. Therefore, after violations are found, they must be fixed, and the formal method must be re-executed, hoping the fix is correct. The overall process may, therefore, be time-consuming, and such timing may not be acceptable anymore to stop modern cyber attacks.

2.2 Correctness by construction

Correctness-by-construction methods provide formal assurance about the configuration compliance with security policies through a different philosophy. They can automatically produce a network security configuration that is already proved correct, i.e., with a guarantee about the satisfaction of the policies, thus avoiding the need to apply other a-posteriori formal verification techniques for further checks. The correctness of the automatically computed solution is provided as long as two main requirements are fulfilled. On the one hand, the solvers that are developed to compute the correct solution integrate multiple algorithm types, including search pruning methods and heuristic combinations of algorithmic proof methods called tactics, so it is necessary to prove that all the internal operations always produce a correct solution, if it exists. Concerning this requirement, nowadays, there exist many state-of-the-art solvers, such as Z3 by Microsoft Research, whose authors have proved the algorithm correctness [14]. On the other hand, the models of the real entities on which these solvers work must capture all the characteristics that may impact the correctness of the solution itself. This task should be done by the proposers of each new correctness-by-construction method, independently from the employed solver.

Constraint programming is suitable for these formal methods. Going back to the example of the SMT problem, for a-posteriori formal verification, each variable had to be bound to a specific fixed value, and each function or predicate had a pre-established interpretation. Instead, for correctness-by-construction methods, some of the variables of the SMT problem may be left open, i.e., their values are not decided a-priori, but are established at run time by the solver. Also, some functions and predicates may be left free, i.e., their interpretation is also decided by the solver. These assignments of variables and function interpretations represent the actual output of the formal method (i.e., the network security configuration in the case analyzed in this paper).

Applying constraint programming for correctness by construction brings manifold advantages in the field of network security configuration. First, humans are not required to work actively to create the configuration because their role is mainly to write the security policies and then to ensure that the applied method works as expected. Second, the configuration is not written before being verified, but it is directly produced in a way that guarantees compliance with the policies. Any further operation of error fixing and formal verification repetition

is thus unneeded. Therefore, the overall process tends to be less time-consuming. Third, correctness-by-construction methods can be enriched with optimization features. Specifically, there exists an optimization-enhanced version of SMT, called Maximum SMT (MaxSMT). A MaxSMT problem differs from an SMT problem because it allows the definition of two types of clauses: the hard constraints whose satisfaction is compulsory, and the soft constraints that have an associated weight. The selected solution is the one that satisfies all the hard constraints and maximizes the sum of the weights of the satisfied soft constraints. Hard constraints thus guarantee the solution correctness, while the soft ones their optimization. Clearly, correctness still has precedence over optimization.

The literature about correctness by construction for network security configuration is unfortunately lacking, because most of the efforts have been spent on a-posteriori formal verification so far.

3 Our Research

In the last years, we have tried to improve the state of the art about network security configuration by proposing novel correctness-by-construction approaches. In this section, we discuss the most relevant results achieved in our research work.

3.1 VEREFOO

VErified REFinement and Optimized Orchestration (VEREFOO) is an approach that we proposed for the automatic configuration of Network Security Functions (NSFs). In particular, it is the first approach in literature to combine automation, correctness by construction and optimization to simultaneously solve the allocation and configuration problems for NSFs in virtual computer networks.

The inputs on which VEREFOO works are a *Service Graph* (SG) and a set of *Network Security Policies* (NSPs). First, the SG represents the network topology, i.e., an interconnection of service functions and network nodes. The only purpose of the SG is to provide a networking service to the users, whose points of access are represented by the end points of the SG (e.g., clients, servers, subnetworks). The functions that may be included in the SG are network functions implementing various functionalities, such as web caching and load balancing, but there are no NSFs. This SG is automatically transformed into an *Allocation Graph* (AG). There, for each link of the original SG, a placeholder position, named *Allocation Place* (AP), is included. Each AP represents a possible position for the allocation of an NSF instance. Second, the NSPs describe the security requirements that must be enforced in the network, such as isolation of traffic flows, or confidentiality and integrity of packets. Their specification can be performed with a user-friendly policy language in order to ease the work of the user.

Upon receiving the AG and NSPs, VEREFOO formulates a MaxSMT problem modeling the NSF allocation and configuration problem. Hard constraints are employed to rigorously define the behavior of the network functions that make up the AG, the paths allowed for traffic flows across the network, and the required satisfaction of all NSPs. Instead, soft constraints are employed to represent two key optimization goals, related to minimizing the number of allocated NSFs and of configured rules. These constraints are then passed to an automated MaxSMT solver, which searches for an optimal correct solution.

If no solution to the problem can be found, a non-enforceability report is generated for the user, who can try to guess why it has not been possible to enforce the NSPs. Instead, if a solution can be found, it is composed of the allocation scheme of the NSFs in the input

SG and the configuration of each allocated NSF. The NSF allocation scheme specifies the APs where each NSF has to be allocated, and it minimizes their numbers so as to minimize resource consumption. Instead, the configuration of each allocated NSF specifies its behavior rules (e.g., the filtering rules for a firewall or the security associations for a VPN gateway). Also their number is minimized, thus minimizing the amount of memory needed to store them and maximizing the NSF performance. When such a solution is found, it is guaranteed to be correct by construction. In fact, VEREFOO complies with the two requirements for correctness by construction. On the one hand, the algorithms of the employed MaxSMT solver, Z3, have been proved correct by its creators [14]. On the one hand, the models defined for the SG and NSPs reflect all their characteristics, possibly impacting the NSF configuration (e.g., the forwarding and transformation behavior of the network functions).

The VEREFOO approach is designed to be a general method that can be applied to any NSF type [8]. As examples, it has been successfully applied for the configuration of packet filtering firewalls [4, 5, 1, 9], VPN gateways [3, 7], SDN switches [12], and smart home devices [11]. Therefore, it is also a suitable solution for the security configuration in mobile networks.

3.2 React-VEREFOO

A limitation of the vanilla version of VEREFOO is that it is not optimized for the case of NSF reconfiguration. For example, supposing that a distributed firewall configuration has been already computed by VEREFOO and deployed onto the network, if later new isolation NSPs are specified to block traffic flows considered malicious because part of a cyber attack, VEREFOO can be reused to compute a new firewall configuration. However, such computation does not take into account the previous firewall allocation scheme and rule sets, but VEREFOO recomputes them from scratch, as they never existed. VEREFOO can still provide a correct-by-construction solution that is compliant with all the NSPs, but the positions and rules of the firewalls may differ from the previous ones, so a huge effort and time to modify them in the actual deployment may be required.

Therefore, we developed a new version of VEREFOO, called React-VEREFOO [20], which tries to recompute a correct-by-construction reconfiguration for distributed firewalls while trying to keep the existing configuration as much as possible.

React-VEREFOO requires two inputs to work. The first input is an SG enriched by an already existing distributed firewall configuration, composed of the allocation scheme of its instances and their filtering rules. The second input is a pair of NSP sets: the *Initial NSP set*, including the old NSPs already satisfied by the existing firewall configuration, and the *Target NSP set*, including the new NSPs to be enforced in the updated network configuration.

Starting from these inputs, an automatic algorithm identifies the network areas that must be reconfigured based on the intersection of the two provided NSP sets. Throughout this intersection, it is possible to discern which NSPs have been added, kept, or deleted in the new set of NSPs with respect to the original one. The algorithm identifies, for all the "added" NSPs, i.e., those relevant to the reconfiguration scenario, which elements of the provided network should be modified because they conflict with the new set of security requirements. The configuration of these elements is therefore put under question. Finally, the approach formulates a new MaxSMT problem whose resolution allows the generation of the new allocation graph and configuration rules of the needed firewalls. This MaxSMT problem still has the same soft constraints employed in VEREFOO to express its two optimization objectives, i.e., the minimization of resource usage and the preference for reusing the original firewall configuration. However, the weights associated with these constraints differ in relation to the considered subject, whether it is

an empty AP, a reconfigured firewall, or a newly generated firewall rule with respect to an already configured one. This difference is such that using a reconfigured firewall, as any of its rules, would result in a higher sum of weight and a preferred solution. As a result, The final configuration will not only minimize the number of consumed resources in general but also produce the smallest number of changes with respect to the initial configuration.

React-VEREFOO avoids recomputing the full configuration from scratch, resulting in a significant reduction in computation time. Specifically, it speeds up the process by narrowing down the space of possible solutions that are analyzed, keeping certain parts of the configuration as fixed, and modifying soft constraints, which are provided to the solver, to have a faster convergence toward the optimal solution. At the same time, the problem still models all the traffic for the NSPs in the target set, so the union of "added" and "kept". This ensures that the computed solution is guaranteed to be correct with respect to the requested NSPs.

Thanks to this behavior, React-VEREFOO has also been successfully leveraged in a looping process for cyberattack mitigation [6] so that it can work on NSPs extracted from logs produced by intrusion detection systems to establish a new firewall configuration without requiring any other human intervention.

3.3 FATO

When a new firewall configuration is computed (e.g., with React-VEREFOO), it differs from the previous configuration for at least one of the two management aspects: the firewall allocation scheme might have been altered (e.g., a new firewall instance has been introduced, or an existing one has been removed), or the firewall rules might have been modified to be compliant with new NSPs. Therefore, the deployed security service must be updated accordingly by applying a series of operations of different types: deployment of a new virtual firewall, removal of an existing firewall, update of the filtering rules of a firewall, and deviation of a traffic flow. This problem is called in literature firewall reconfiguration transient, and it consists of determining a specific ordering of the reconfiguration operations so that the global configuration is changed from the initial state to the target one. In this transient, there may be multiple intermediate steps, and their number is equal to the number of changes that are applied to the firewall configuration, i.e., the number of newly deployed instances, the number of removed instances, and the number of modified filtering rule sets.

For the management of the firewall reconfiguration transients, we proposed another correctness-by-construction approach, called *FirewAll Transients Optimizer* (FATO)[10], which can automatically compute the scheduling of the reconfiguration operations so as to minimize the number of intermediate unsafe states where some NSPs are not yet satisfied.

In order to accomplish this objective, the FATO approach works as follows.

First, it requires the user to specify four inputs. The first and second inputs are the initial and target security SGs, which respectively include the description of the initial configuration of the distributed firewall (i.e., the start state of the reconfiguration transient) and target configuration of the distributed firewall (i.e., the final state of the reconfiguration transient). The second input is a set of target NSPs expressing the requirements that must be satisfied by the target configuration, defining which traffic flow must reach their destination, and which ones must instead be blocked. The fourth input is an optimization profile, which represents a compact specification about the relative priority of the NSPs. For instance, the user may request that the isolation NSPs have higher priority than the reachability NSPs (*security-max* profile) or vice versa (*service-max* profile). They may also specify other optimization objectives in addition to the basic one, e.g., to maximize the number of NSPs that are satisfied in each

intermediate state (*policy-max* profile).

After receiving these inputs, from the specification of the optimization profile, FATO defines a ranking for the input NSPs. Then, the initial and target security SGs with the firewall configurations, the target NSPs and their ranking are used by FATO to formulate a new MaxSMT problem. After solving this optimization problem, FATO identifies the optimal order of reconfiguration changes in such a way that the optimization fulfills the criteria derived from the ranking. This scheduling can be followed by a human who manages the virtual network, or a state-of-the-art orchestrator such as Open Source MANO or Docker Compose can exploit it to perform the required actions.

4 Conclusion and Future Work

This paper surveyed our latest research on introducing correctness by construction in network security configuration. Specifically, it presented VEREFOO as an approach to allocate and configure automatically NSFs such as firewalls and VPN gateways, React-VEREFOO as a special version designed to optimize firewall reconfiguration, and FATO as a methodology to manage firewall reconfiguration transients.

Future work is envisioned to further advance the state of the art about correctness-by-construction approaches. On the one hand, VEREFOO may be customized to be applied to other types of NSFs, such as stateful functions, web-application filters, and intrusion detection systems. On the other hand, if the proposed methodology mostly leverages MaxSMT as problem formulation, the usage of other formal methods may be investigated in order to compare their effectiveness with respect to MaxSMT for the enforcement of correctness by construction.

References

[1] Daniele Bringhenti, Simone Bussa, Riccardo Sisto, and Fulvio Valenza. A two-fold traffic flow model for network security management. *IEEE Trans. Netw. Serv. Manag.*, 21(4):3740–3758, 2024.

[2] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Serena Spinoso, Fulvio Valenza, and Jalolliddin Yusupov. Improving the formal verification of reachability policies in virtualized networks. *IEEE Trans. Netw. Serv. Manag.*, 18(1):713–728, 2021.

[3] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, and Fulvio Valenza. Short paper: Automatic configuration for an optimal channel protection in virtualized networks. In *CYSARM@CCS '20: Proceedings of the 2nd Workshop on Cyber-Security Arms Race, Virtual Event, USA, November, 2020*, pages 25–30. ACM, 2020.

[4] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. Automated optimal firewall orchestration and configuration in virtualized networks. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*, pages 1–7. IEEE, 2020.

[5] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. Automated firewall configuration in virtual networks. *IEEE Trans. Dependable Secur. Comput.*, 20(2):1559–1576, 2023.

[6] Daniele Bringhenti, Francesco Pizzato, Riccardo Sisto, and Fulvio Valenza. A looping process for cyberattack mitigation. In *IEEE International Conference on Cyber Security and Resilience, CSR 2024, London, UK, September 2-4, 2024*, pages 276–281. IEEE, 2024.

[7] Daniele Bringhenti, Riccardo Sisto, and Fulvio Valenza. Automating vpn configuration in computer networks. *IEEE Trans. Dependable Secur. Comput.* in press.

[8] Daniele Bringhenti, Riccardo Sisto, and Fulvio Valenza. A novel abstraction for security configuration in virtual networks. *Comput. Networks*, 228:109745, 2023.

[9] Daniele Bringhenti and Fulvio Valenza. Greenshield: Optimizing firewall configuration for sustainable networks. *IEEE Trans. Netw. Serv. Manag.* in press.

[10] Daniele Bringhenti and Fulvio Valenza. Optimizing distributed firewall reconfiguration transients. *Comput. Networks*, 215:109183, 2022.

[11] Daniele Bringhenti, Fulvio Valenza, and Cataldo Basile. Toward cybersecurity personalization in smart homes. *IEEE Secur. Priv.*, 20(1):45–53, 2022.

[12] Daniele Bringhenti, Jalolliddin Yusupov, Alejandro Molina Zarca, Fulvio Valenza, Riccardo Sisto, Jorge Bernal Bernabé, and Antonio F. Skarmeta. Automatic, verifiable and optimized policy-based security enforcement for sdn-aware iot networks. *Comput. Networks*, 213:109123, 2022.

[13] Cloudflare. DDoS attack trends for 2022 Q2, 2022. Available: https://blog.cloudflare.com/ddos-attack-trends-for-2022-q2/, Visited: 2024-10-10.

[14] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *Proc. of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, Budapest, Hungary, March 29-April 6, 2008*, volume 4963, pages 337–340. Springer, 2008.

[15] Yonas Engida Gebremariam, Daniel Gerbi Duguma, Hoonyong Park, Ye Neung Kim, Bonam Kim, and Ilsun You. 5g and beyond telco cloud: architecture and cybersecurity challenges. In *2021 World Automation Congress, WAC 2021, Taipei, Taiwan, August 1-5, 2021*, pages 1–6. IEEE, 2021.

[16] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 113–126, San Jose, CA, 2012. USENIX.

[17] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. Checking beliefs in dynamic networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 499–512, Oakland, CA, 2015. USENIX Association.

[18] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 290–301, New York, NY, USA, 2011. ACM.

[19] Aurojit Panda, Ori Lahav, Katerina Argyraki, Mooly Sagiv, and Scott Shenker. Verifying reachability in networks with mutable datapaths. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, pages 699–718, Berkeley, CA, USA, 2017. USENIX Association.

[20] Francesco Pizzato, Daniele Bringhenti, Riccardo Sisto, and Fulvio Valenza. Automatic and optimized firewall reconfiguration. In *NOMS 2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, May 6-10, 2024*, pages 1–9. IEEE, 2024.

[21] Proton. A brief update regarding ongoing DDoS incidents, 2022. Available: https://proton.me/blog/a-brief-update-regarding-ongoing-ddos-incidents, Visited: 2024-10-10.

[22] Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. Symnet: Scalable symbolic execution for modern networks. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 314–327, New York, NY, USA, 2016. ACM.

[23] Verizon. Data Breach Investigations Report, 2024. Available: https://www.verizon.com/business/resources/Tf72/reports/2024-dbir-data-breach-investigations-report.pdf, Visited: 2024-10-10.

[24] Yifei Yuan, Soo-Jin Moon, Sahil Uppal, Limin Jia, and Vyas Sekar. Netsmc: A custom symbolic model checker for stateful network verification. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 181–200, Santa Clara, CA, February 2020. USENIX Association.

[25] Kaiyuan Zhang, Danyang Zhuo, Aditya Akella, Arvind Krishnamurthy, and Xi Wang. Automated

verification of customizable middlebox properties with gravel. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 221–239, Santa Clara, CA, February 2020. USENIX Association.