Enabling Security Research through Efficient Partial Deployment Topology Configuration and Validation

1st Bashayer Alharbi
University of Colorado - Boulder
Boulder, USA
bashayer.alharbi@colorado.edu

2nd Karl Olson
University of Colorado - Boulder
Boulder, USA
karl.olson@colorado.edu

3rd Eric Keller
University of Colorado - Boulder
Boulder, USA
eric.keller@colorado.edu

***Abstract*—How to measure security value in partial deployments has long been a consideration for the Internet research community. Without clear security outcomes, adoption of security mechanisms may take years before users begin to see any benefit. This lack of clarity can serve to further delay adoption as incentives to implement are often outweighed by additional costs or complexity. While prior efforts have looked at theoretical approaches to estimate this critical mass of partial deployment within a topology, no effort has been able to effectively simulate and measure such an outcome. In this work, we provide an early effort to demonstrate how topology simulation can be used to effectively deploy and measure partial deployments of RPKI utilizing the SEED Internet Emulator. Our efforts show that this approach can be used to simulate large networks and provide an effective means to measure partial deployment value of security protocol deployments. Further, we demonstrate that adoption rates of greater than fifty percent begin to show exponential return on security outcomes for both adopters and non-adopters alike.**

***Index Terms*—Internet Simulation, RPKI, Partial Deployment**

I. Introduction

The Internet is central to every aspect of modern life, yet it's main protocol, the border gateway protocol (BGP), is ripe for attack. This can be seen in a variety of recent attacks and demonstrations of its vulnerabilities [12]–[14] where an attacker is able to hijack an IP prefix. What this means is they would be able to influence the traffic destined to that prefix - e.g., intercepting it for a man-in-the-middle attack, directing it to a blackhole for a denial of service attack, or forwarding to a server under the attacker's control to impersonate a particular address. This can reduce anonymity in systems like Tor [16], influence cryptocurrency in systems like Bitcoin [15], or disrupt services like Youtube [17].

In response, a number of security protocols have been introduced to thwart these attacks [10], [19], [25]. The resource public key infrastructure (RPKI) [10], has gained the most traction and provides a protection known as route origin authentication (ROA). With RPKI, the owner of an IP prefix address block can sign a certificate with a private key, and share that with a central authority who will store that certificate along with a public key in a database. Network operators can download that database to a local server, known as a validator, which will validate all entries in the downloaded data. Through an RPKI to Router (RTR) protocol with local routers, these prefixes can be cached in that local router such that when it receives a BGP update message, it can look in the database to see if the given prefix is valid (and, e.g., discard the bad announcement).

One challenge with RPKI, and all of the other proposed Internet security solutions, is that its effectiveness is highly dependent on how widely deployed it is. While there were some theoretical studies around effectiveness in partial deployments [20], and a request for comments process that allowed stakeholders to voice concerns, RPKI wasn't really evaluated with real routers until network operators started deploying it in production networks. Fortunately, some experimental frameworks have been introduced which leverage containerization to emulate larger scale network topologies. For example, Containernet [7] allows an experimenter to describe a network topology of hosts and switches in Python, and it will launch a container for each. Experimenters can configure the switches (e.g., using software-defined networking protocols), run software on the hosts, and generate traffic. The SEED Internet Emulator [2], took a similar concept but also allows running routing software (e.g., BIRD [21]), and automatically configuring the routers based on higher level information (e.g., this autonomous system is an internet exchange and peers with a specified set of autonomous systems). However, these experimental frameworks are still missing one important capability - being able to effectively evaluate partial deployment of Internet security protocols.

In this paper, we extend the SEED Internet emulator framework to enable partial deployment evaluation of RPKI (and other routing protocols with small extensions). In particular, we: (i) enable the automatic deployment and configuration of RPKI in an autonomous system (Section II), (ii) provide a programmatic API to enable experimenters to create a variety of scenarios of partial deployment, and introduce metrics to define effectiveness of a given scenario and provide means to measure those metrics programmatically (Section III), and (iii) use this extended testing framework to evaluate the effectiveness of RPKI, as an example, in a variety of partial deployment scenarios on a mini scale Internet (Section IV). Our results show that partial deployment simulations can be programmatically conducted and measured to help inform when adopters/non-adopters can begin to see large security value resulting from deployments. With RPKI in particular,

we find that an adoption rate over 50% is necessary before exponential security value is begins to present within a topology (where non-participants are likely to begin to see the benefits in addition to the adopters). Both the approach and results show value for future security research efforts and help provide clarity to partial deployment value.

II. Automated RPKI Emulation

Conducting large-scale network research is challenging at best as hardware costs can quickly become prohibitive, even for small-scale topologies. For this reason, hardware emulation through software has been used to replicate network devices efficiently within a local simulation environment. Efforts such as GNS-3 [1], NSA's Greybox [3], Boson's Netsim [5], and Eve-NG [4] are just a few common examples where device emulation can be used to replicate physical topologies. However, pure hardware emulation still requires a significant investment in compute resources to maintain large networks as each device emulated requires its own allocation of resources from the local host. Further, emulated devices often require their own individual configuration to establish the right connections, routing, and topology goals, making experiments that require significant topology configuration changes challenging at best.

To address these limitations, platforms which *simulate* networking devices do so by replicating the operation of networking protocols, which are then run in a lightweight container or host. Efforts such as Mininet [6], Containernet [7], and the recent SEED Internet Emulator (SIE) [2] follow this approach, relying on the Linux kernel for networking support. In the cases of Containernet and SIE, these devices are then deployed via lightweight Docker containers which are then networked together to provide a lightweight simulation environment. Configuration of individual devices within a network topology are further enabled by programming abstractions that can quickly define configurations through associated C or Python libraries.

A. Configuring SIE to use RPKI

In order to enable the large-scale change necessary to efficiently measure partial deployment scenarios, we relied on the SEED Internet Emulator. While originally designed to provide a networking environment for security education [2], the lightweight containerization of networking nodes and use of Python classes to represent internet elements allows for quick scripting of networking topologies at any scale. Further, the use of Docker containers allows for simulation of hundreds of nodes on a single host and can be further scaled to a cloud environment for even larger topologies, making it an ideal platform to perform large scale testing of experiments that require significant topology changes.

By default, SIE provides a number of routing protocols and common host templates (router, web server, PC host, DNS, etc.) to deploy. We were able to incorporate additional elements into this platform, such as introducing RPKI, through a number of small configuration changes. First, we modified

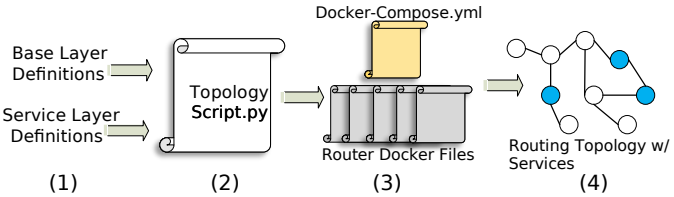


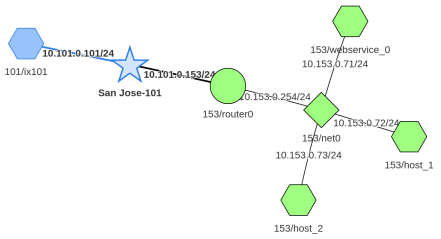
Fig. 1. **Topology Deployment Process** (1) A set of Python classes is used to define a base layer of devices and connections. An additional services layer is then defined to establish individual services necessary for each device (e.g. web server, DNS, etc.) (2) A topology script is run to generate Docker configuration files for each device along with a Docker-compose file, which will be used to deploy the overall topology. Within the topology script is a designator for a defined percentage of devices to operate RPKI randomly. (3) The Docker-compose file is run and deploys the experiment topology. (4) The topology is active and ready to inject a malicious advertisement and measure resulting outcomes.

the included `/seedemu/compiler/Docker.py` compiler to create an RPKI server template. When deployed, this template would create a server which installed the NLnet Labs Routinator software [8] and established the initial configuration which routers could use to validate RPKI data. Second, we modified the `/seedemu/layers/Ebgp.py` configuration template for the `birdc` BGP routing daemon to include an optional configuration for routers that we wanted to run RPKI on. If a router was selected to use RPKI, all associated configurations and setups would be done to connect to our RPKI server using this template. Last, we modified `/seedemu/utilities/Makers.py` to implement RPKI on routers as an additional configuration element for the SIE topology configuration script. This included a random selector used to define a target percentage for RPKI deployment.

B. Creating and Deploying an Environment

To create our test topologies, SIE relies on the concept of layering to establish and define a network and associated services. A simple workflow to establish an environment is shown in Figure 1. First a base layer defines the devices and connections between them. These may be routers, internet exchanges, hosts, or other elements of a network topology. As part of this base layer, routing and neighbor relationships are also defined through a simple Python script which identifies a device and then configures network peerings and linkages through a simple definition, as shown in Figure 2. Second, a services layer adds any additional services to a base device. These are again specified as part of the Python configuration script, but only after a host has been created on the base layer for these services to run on.

Once a topology configuration script is established, the SIE compiler builds the associated Docker configuration templates for each host with the specified services and configurations. A `docker-compose.yml` template is also built to orchestrate the deployment of each container into a single operational environment and handle the underlying networking configuration between each container. Changes to the environment can quickly be integrated by adjusting the toplogy script, recompiling to update the Docker container template(s), and




```
# Create and set up the stub AS (AS-153)
# and attach to IX101 with one web server,
# two hosts, and random RPKI selection.
Makers.makeStubAs(emu, base, 153, 101,
[web, None, None],rpki[5])
```

Fig. 2. **Topology Definition Example** Establishment of devices within a topology follow a simple configuration pattern. First, a base layer is defined where the object will apply. Second, a number is given to define the autonomous system, in this case `AS153`. We then define where this AS will connect to, in this case `IX101`. Any additional hosts/services that should be deployed off of the AS router can be specified within the final argument. In this example, three hosts are deployed with one being a webserver. Last, we added an RPKI identifier statement that would either be True or False in order to determine whether or not a device should be operating RPKI in our tests. Additional configuration templates exist for transit AS's, peering relationships, and real-world routers.

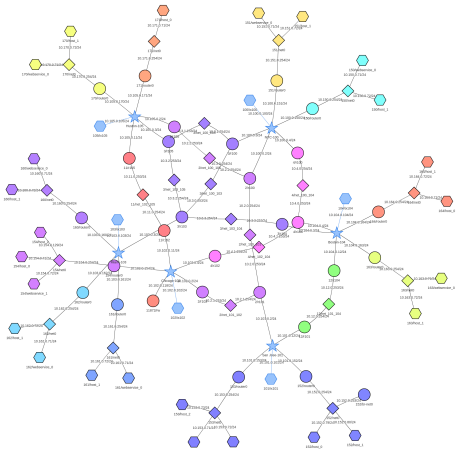


Fig. 3. **Deploying Larger Topologies** A sample larger topology view used to measure partial deployment of RPKI. Topology contains a mix of internal and external routing protocols, internet exchanges, host systems & services, along with a number of independent Autonomous Systems used to measure attack propagation. Color coding marks the various separate Autonomous networks for easy identification. Less than 40 LoC were used to define a topology of 69 devices hosting various services such as real world connectivity, an RPKI server, routing, peering relationships, web servers, and hosts.

finally updating the associated container(s). A further benefit of this approach is that we could save pre-configured topology scripts and quickly rebuild the containers for deployment in as little as a few minutes, allowing for efficient testing and modification of experiments, regardless of topology size. Demonstration of this scalability is shown in Figure 3.

III. Enabling Partial Deployment Experiments

Measuring the security benefit of a partially deployed security protocol required an efficient mechanism to both deploy

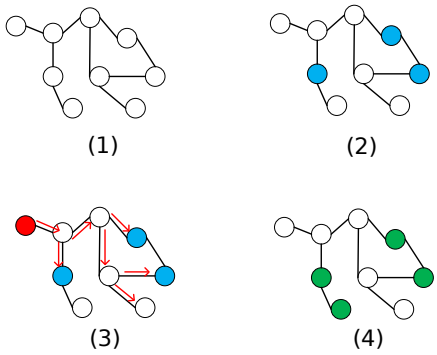


Fig. 4. **Measuring Partial Deployment of RPKI** (1) Topology of routers and services is defined and deployed in simulated environment. (2) Target percentage of RPKI deployment is selected and configured. (3) False route info injected into topology and allowed to propagate. (4) Routing tables checked for inclusion of bad routes and topology is graphed to measure overall effect.

and correctly configure the targeted solution on some subset of available hosts. For our security mechanism, we chose to implement RPKI, due to its favor by IANA as a preferred BGP security solution and its growing deployment in real-world topologies [9], [10]. RPKI works to limit the acceptance and propagation of bad BGP routes by validating that a route advertisement is associated with the true owner and that they have advertised the route. By detecting bad routes (malicious hijacks or misconfiguration), a router utilizing RPKI can prevent a bad route from entering their routing table and possibly re-directing traffic away from the intended source. By preventing this route acceptance, downstream routers can also be protected from a bad route as the update would never propagate past the validating router. For this reason, RPKI provides potential benefits to non-participating routers making it an ideal choice for measuring partial deployment security.

To implement the partial deployment of RPKI into our network topology we implemented a random sample selector into the SIE topology configuration script that could be tuned to reflect a given percentage of devices. Devices selected at random would be configured to operate RPKI while the rest would remain as standard BGP routers. By introducing this selector we could quickly adjust a topology to reflect any percentage of RPKI deployment while the remaining configuration elements of each device would be handled by the underlying structure of the SIE compiler. Saving the compiled Docker configuration templates allowed for precise replaying of any scenario, allowing us to further test for edge case outcomes or to validate experiment outcomes multiple times.

A. Measuring Partial Deployment Outcomes

In order to measure the effect of RPKI and resulting security afforded by a partial deployment, a series of scripts were generated to both randomly generate a prefix hijack from a "rogue" device in the topology and measure the resulting route propagation post attack, as shown in Figure 4. For post-attack outcomes we relied on two key measurements, as follows:

Control plane measurements. To measure the impact of prefix hijacking on the control plane, we developed a script to pull the routing table information from each router in the topology and search for the hijacked prefix. From this information, we could quickly identify the number of affected routers that had the hijacked routes in their routing tables and validate that ASes adopting RPKI were not affected. We followed this measurement by manually validating our topology results in order to ensure that route propagation followed a clear and available path through the topology and was not propagated by devices implementing RPKI.

Data plane measurements. To test the impact of prefix hijacking on the data plane, we conducted a traceroute from each router not implementing RPKI to the targeted prefix. If a router was not affected, the path would take us to our real-world AS and out to the actual real-world endpoint. If a router was affected, the routing would end within our topology at the router conducting the prefix hijack. A script was developed with the aim of assessing the reachability of each Stub AS to a valid IP address 8.8.8.8 through the utilization of the ping and traceroute commands using `docker exec`. The script subsequently calculated the number of ASes that were able to successfully reach the target IP address per each experimental iteration, providing an effective method for identifying the devices that were affected in the data-plane.

Repeat simulations were conducted to test for potential variance in outcomes. In particular we note two key outcomes of interest that we needed to be aware of: For re-simulated experiments, all outcomes should remain the same, e.g., for the same topology and attacking device, the same devices should be affected post attack. Second, for a given deployment % of RPKI a series of random topologies and attacking devices were selected in order to determine if a specific iteration had a greater or less effect on malicious route propagation. In cases where outcomes were different based on topology, we note these differences in our results. We averaged the results in order to consider the effect of edge cases for our test topology (we discuss these cases further in the Discussion section.)

B. Experiment Workflow

In order to conduct repeatable experiments, each iteration of a partial deployment measurement used the following procedure:

1) Define topology % for RPKI deployment
2) Run SEI compiler to build environment
3) Save a copy of the environment for re-use
4) Deploy environment and validate topology operation
5) Conduct attack
6) Extract topology routing tables
7) Model and measure outcomes
8) Repeat experiment to test for variances

By following this procedure, the only changes we would need to make for testing partial deployment security is defining a new percentage of devices that would host RPKI and redeploying the experiment with the new configuration. All other procedures would be handled through automated scripts to both conduct an attack and measure the resulting routing tables to determine attack propagation and defense within a given topology.

C. Preventing Accidental Exposure

Since our topology used real-world connectivity through a single AS, we needed to ensure that external route information was propagated into our simulation environment but that no actions or mis-configurations would have an external effect. By default, the SIE provides this isolation through a combination of VPN configurations (for external devices to reach into the simulation) and use of importing real-world route announcements into the sim from a real-world AS (for devices to reach outward). When combined, these two implementations provide for a simulation boundary that acts like it is connected to the real-world, but is in fact separated through management abstractions which prevent announcement of internal routes due to no actual peering being performed with a real-world router. This would allow us to test hijacking of real-world prefixes that both did and did not use RPKI, but within the protected environment of our simulation.

IV. Results

For our measurements, we considered RPKI deployment scenarios ranging from 10% - 90% deployment, conducted in 20% intervals. 0% and 100% measurements were conducted specifically to validate topology operation and outcomes associated with each (we discuss separately below). Within this deployment a random autonomous system not within the RPKI deployment selection was chosen to initiate a route-hijack for our target prefix of 8.8.8.0/25. We chose this prefix due to its known ownership and registration in the ARIN RPKI database and for our ability to utilize a longer, more specific prefix from the true subnet mask of 8.8.8.0/24. All experiments were conducted on a CloudLab [11] node using the modified SEED `mini-internet.py` topology, which incorporated our changes for establishing various deployment scales of RPKI. This topology simulated a total of 69 devices containing a mix of transit ASes, stub ASes, internet exchange points, and end-hosts.

After conducting our prefix hijack, we then pulled each router's routing table and checked for propagation of our chosen prefix. A total count of affected systems allowed for correlation between RPKI deployment percentage and resulting overall topology security (defined as either routers having direct avoidance through implementation of RPKI or through downstream protection of devices not running RPKI but protected by the upstream router.) Results correlating RPKI deployment to adoption of the malicious prefix hijack are shown in Figure 5. Of note are the significant security benefits found when a topology breaks the 50% deployment mark due to the higher likelihood of non-participants receiving protection from upstream RPKI routers. This is further demonstrated at the 70% deployment mark where nearly every router avoided adopting the malicious route.

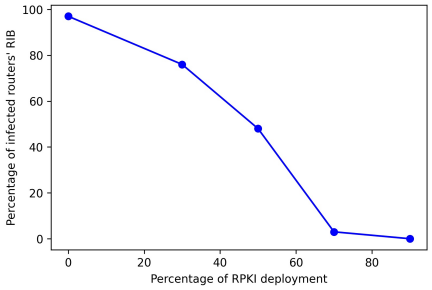


Fig. 5. **Control Plane Measurements** Measuring the impact of prefix hijacking on the control plane over different RPKI adoption rates.

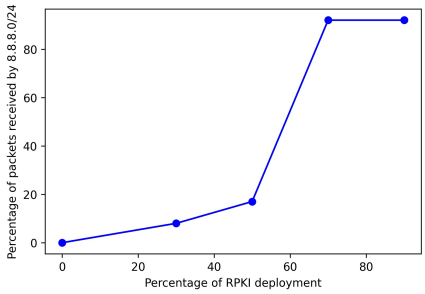


Fig. 6. **Data Plane Measurements** Percentage of data plane traffic able to route to true owner and not malicious hijacker. It was discovered that a combination of topology layout and random selection of routers not deploying RPKI would allow for hijacking of traffic despite broader RPKI deployment.

As confirmation for our control plane assessment, we further conducted traces from each router to the targeted "valid" host located at 8.8.8.8/24. If a router had RPKI enabled, we expected these routers to successfully reach the valid host. What we discovered was a significant mismatch between the control plane and data plane for topologies that implemented a low RPKI deployment percentage, as shown in Figure 6. The reason for this outcome we discovered was upstream routers that were not utilizing RPKI accepted the malicious advertisement and routed the corresponding traffic towards the malicious host. Updates sent by these routers would be rejected by the RPKI systems as they would see these updates as "INVALID" and maintain their original route information. We provide a detailed overview of this outcome next.

A. Data-plane Routing Mismatch

While a BGP router's control plane is used to select and maintain routes, the actual path a packet may take can differ from this control-plane information due to either misconfiguration, routing anomalies, or malicious action [23]. In one estimate, as much as 8% of global routes present a mismatch between the control and data plane [24]. In considering the potential for these differences, we further conducted a post-hijack assessment utilizing a series of traceroutes to confirm equivalent routing behavior between both the data and control plane. What we found were instances where a router may utilize RPKI and populate the correct information in their routing table, but transit across an AS affected by a hijack

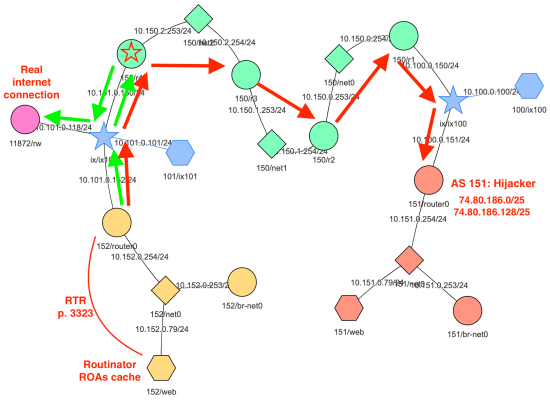


Fig. 7. **Data-Plane Routing Mismatch** (1) AS152 operates RPKI and has a valid route (green path) to 74.80.186.0/24 through ASN150 (4) to the real-world connection (2). (3) ASN151 acts as a hijacker and advertises a longer sub-prefix of 74.80.186/25. (4) ASN150 accepts the bad prefix to ASN151 due to no local implementation of RPKI. ASN152 (1) traffic ends up routing to attacker even though a valid RPKI route is in the local routing table. Note: This is a reduced size topology to highlight an example. Similar occurrences were found in our larger simulation topology.

that results in different routing behavior. In our experiments, this resulted in traffic being re-directed to an attacker even though the local AS was correctly populating route data to the true host. A demonstration of this outcome is shown in Figure 7.

This has a number of concerning implications. First, this puts into question the value of RPKI as a security mechanism for route origin validation if the underlying routing data-plane behavior does not match. This can provide false assurances to operators, making actual hijacks harder to identify through control plane assessments. Second, this shows the importance of early adoption by up-stream service providers to integrate RPKI due to their out-sized role in enabling broader security outcomes. Third, while RPKI is not a path validation solution, the need for path validation in conjunction with RPKI is still a valid need. While efforts such as BGPsec provide this capability, it requires significant adoption before broad value can be realized. This is due to the requirement of *all* routers needing to implement BGPsec along a given path, otherwise the standard reverts to removing the path-validation in absence of full participation [25].

V. Discussion

A significant challenge in the adoption of security protocols is defining the immediate value for early adopters. Lack of incentives or a clear operational value can hinder deployments, stagnating security goals until a critical mass is reached. However, defining where this critical mass may lie is rarely clear. These challenges are especially prevalent on the Internet as autonomous systems are independently managed by operators who may not readily adopt new network complexities. This is readily observed in the adoption of RPKI where after ten years of availability still remains at less than 30% adoption globally [9].

By demonstrating the value of a partial deployment scenario for RPKI over multiple adoption scenarios, we provide a programmatic approach that demonstrates a measurable outcome to security deployments. Further, our solution is protocol agnostic and can easily be reconfigured to test future security mechanisms with minimal configuration in the SIE environment.

One limit of our current approach is related to the small topology used at present. The largest simulation we conducted replicated 69 devices consisting of a mix of routers, hosts, and switches, used to provide a complete testing environment. Of these 69 devices, only 26 devices represented routers for 15 distinct autonomous systems, of which six were transit networks. The low sampling, combined with a single external network exit, provided for outcomes where potentially as few as four devices running RPKI could provide 100% coverage to the entire topology. We are careful to consider these results and have worked to provide measurements that average these edge cases over a number of experiments. However, our broader intent here is to demonstrate the workings of the approach used. In a future work we plan to significantly scale the simulated environment to accommodate thousands of autonomous systems utilizing a number of exit points that can serve as an operational test-bed for other researchers.

VI. Future Work

SIE has shown to be a very versatile platform for conducting Internet emulation and assessments. We plan to further extend the results of this early work in a number of ways with future efforts. In particular, we would like to see how far the platform could take us towards realizing a 1:1 scale network simulation platform. We note that a number of opportunities for further efficiencies exist within the current SIE environment that could further reduce container overhead requirements, allowing for reasonably large scales (a few thousand devices) of simulation being performed on a single host. This would enable a number of research efforts where access to large test-bed platforms is either too costly or complicated to scale efficiently.

While SIE has a separate compiler that is able to build a Google Teraform deployment environment, we have only minimally validated its operation. We would like to combine the effort to minimize container overhead with this cloud environment to see how far we could get to achieving a full internet topology simulation. Given the ability of the SIE emulator to import real-world routing table information, we believe this data could be used to automate the container deployment and topology creation.

VII. Conclusion

Communicating the value of security mechanisms can be hard to convey to operators, who often consider the added complexity and cost as detriments to adoption. This is especially true for early adopters who may not immediately realize the benefits until a critical mass of deployments are reached. In this work we show how measuring the adoption of RPKI across partial deployment scenarios can help inform the community of where this critical mass may lie, serving to answer the question: "When will I see the benefit?". We further demonstrated a programmatic approach to building and simulating network topologies efficiently so that other mechanisms can be validated and compared in support of security research for the Internet community.

References

[1] Graphical Network Simulator-3. https://www.gns3.com/.

[2] Du, W., Zeng, H., Won, K. (2022, November). SEED Emulator: An Internet Emulator for Research and Education. In Proceedings of the 21st ACM Workshop on Hot Topics in Networks (pp. 101-107).

[3] National Security Agency Graybox Program. https://www.nsa.gov/business/programs/greybox/.

[4] EVE-NG: the Emulated Virtual Environment For Network, Security and DevOps Professionals. https://www.eve-ng.net

[5] Boson NetSim Network Simulator. https://netsim.boson.com/.

[6] Mininet. http://mininet.org/.

[7] Containernet. https://containernet.github.io/.

[8] NLNetLabs Routinator. https://nlnetlabs.nl/projects/rpki/routinator/.

[9] NIST RPKI Deployment Monitor. https://www.nist.gov/services-resources/software/nist-rpki-deployment-monitor.

[10] Internet Assigned Numbers Authority Resource Public Key Infrastructure. https://www.iana.org/assignments/rpki/rpki.xhtml.

[11] CloudLab. https://www.cloudlab.us

[12] Truth Behind the Celer Network cBridge cross-chain bridge incident: BGP hijacking. https://medium.com/coinmonks/truth-behind-the-celer-network-cbridge-cross-chain-bridge-incident-bgp-hijacking-52556227e940

[13] Maria Apostolaki, Aviv Zohar, Laurent Vanbever. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. IEEE Symposium on Security and Privacy. May 2017.

[14] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, Prateek Mittal. RAPTOR: Routing Attacks on Privacy in Tor. USENIX Security. August 2015.

[15] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[16] Roger Dingledine, Nick Mathewson, Paul Syverson. Tor: The Second-Generation Onion Router. Usenix Security. 2004.

[17] YouTube Hijacking: A RIPE NCC RIS case study. https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study

[18] Matthew Lepinski, Kotikalapudi Sriram. RFC 8205: BGPsec Protocol Specification. https://www.rfc-editor.org/rfc/rfc8205.html

[19] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure Border Gateway Protocol (S-BGP) — Real World Performance and Deployment Issues. Proceedings of the Network and Distributed System Security Symposium (NDSS). February 2000.

[20] Robert Lychev, Sharon Goldberg, Michael Schapira. BGP security in partial deployment: is the juice worth the squeeze? ACM SIGCOMM. 2013.

[21] The BIRD Internet Routing Daemon. https://bird.network.cz/

[22] Feamster, N., Winick, J., & Rexford, J. (2004). A Model of BGP Routing for Network Engineering. ACM SIGMETRICS Performance Evaluation Review, 32(1), 331-342.

[23] Wong, E. L., Balasubramanian, P., Alvisi, L., Gouda, M. G., & Shmatikov, V. (2007, August). Truth in advertising: Lightweight verification of route integrity. In Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing (pp. 147-156).

[24] Mao, Z. M., Rexford, J., Wang, J., & Katz, R. H. (2003, August). Towards an accurate AS-level traceroute tool. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (pp. 365-378).

[25] Austein, R., Bellovin, S., Housley, R., Kent, S., Kumari, W., Montgomery, D., ... & Sriram, K. (2017). RFC 8205-BGPsec Protocol Specification.