Towards a Framework for Evaluating BGP Security

Olaf Maennel[†] Iain Phillips[†] Debbie Perouli[§]
Randy Bush[⋆] Rob Austein[‡] Askar Jaboldinov[†]

[†]Loughborough University, UK [⋆]Internet Initiative Japan
[‡]Dragon Research Labs, USA [§]Purdue University, USA

Abstract

Security and performance evaluation of Internet protocols can be greatly aided by emulation in realistic deployment scenarios. We describe our implementation of such methods which uses high-level abstractions to bring simplicity into a virtualized test-lab.

We argue that current test-labs have not adequately captured those challenges, partly because their design is too static. To achieve more flexibility and to allow the experimenter to easily deploy many alternative scenarios we need abstractions that allow auto-configuration and auto-deployment of real router and server code in a multi-AS infrastructure. We need to be able to generate scenarios for multi-party players in a fully isolated emulated test-lab and deploy the network using virtualized routers, switches, and servers.

In this paper, our abstractions are specifically designed to evaluate the BGP security framework currently being documented by the IETF SIDR working group. We capture the relevant aspects of the SIDR security proposals, and allow experimenters to evaluate the technology in topologies of real router and server code. We believe such methods are also useful for teaching newcomers and operators, as it allows them to gain experience in a sand-box before deployment. It allows security experts to set up controlled experiments at various levels of complexity, and concentrate on discovering weaknesses, instead of having to spend time on tedious configuration tasks. Finally, it allows router vendors and implementers to test their code and to perform scalability evaluation.

1 Introduction

The Border Gateway Protocol (BGP) [1] was not designed with any concern for security and is therefore quite vulnerable to attacks such as mis-originations of prefixes (known as "route hijacks"), protocol attacks, and accidental misconfiguration [2–4]. Efforts have been made to secure BGP [5, 6], with the IETF Secure Inter-Domain Routing (SIDR) working group proposing several new standards [7]. Initial implementations of these standards are available [8] and are being deployed. However, before a proposed security framework can be widely deployed, it would be good if an evaluation of the architecture, implementation, and operational implications could be performed. If there are problems in the proposed protocols, the IETF and vendors would want to correct them before widespread deployment. In this paper we are interested in the question:

> *How can we determine if these BGP security protocols are ready for large-scale, Internet-wide, deployment?*

The complexity of both the proposed BGP security solutions, as well as the expected deployment scale, suggest that simple protocol compliance tests are not sufficient: operational deployment and scalability tests are critical. Furthermore, system soundness during migration and in day-to-day operations needs to be demonstrated.

This problem has to be addressed using multiple methods, including formal verification, simulation, emulation, small-scale and real-world test-labs. The value of large real-world test-beds has been widely recognized [9, 10], but none of them sufficiently captures the realism required with respect to a comprehensive BGP security evaluation. In fact, the underlying fundamental problem with test-lab proposals is that each security evaluation poses different demands on the characteristics of the test-lab. With respect to the evaluation of the RPKI [11] it turned out to be easier to start from scratch [12] than to use any of the existing test-labs. Test-labs need to be flexible and adjust quickly to the demands of the evaluation task.

For this reason, we argue that it is important to focus on *high-level abstractions that allow auto-generation of experimental topologies*. We believe that virtualization combined with abstractions will help with complex protocol and system evaluations.

We would stress that we do not mean that existing approaches are heading in the wrong direction. We just believe they can be augmented with additional test-lab testing techniques that provide more flexibility using auto-configuration. With respect to the SIDR proposals in particular, we also consider the problem of complex BGP policy interactions [13, 14], and the ability to experiment with migration issues [15]. We want to empower security experts with the ability to create more realistic attack scenarios, e.g., [16].

1.1 Challenges

The challenge is to provide a testing methodology that allows security experts, router vendors, and Internet Service Providers (ISPs) to explore the system and protocols at a realistic scale. Such a toolkit needs to run real vendor code and infrastructure software that is used in the Network Operation Centers (NOCs).

To allow the repeated running of many different experiments in a fast and flexible manner, a fully functional and fully isolated emulated network should be automatically deployed and explored. This leads to the need for auto-generation of large parts of the BGP security components as well as the network infrastructure. We need abstractions to model multi-ISP topologies and reasonable defaults to generate the detailed device (router and server) configurations. These abstractions provide the ability to create networks of various levels of complexity, with minimum specification effort.

Unfortunately, today it is not trivial to create and configure a test-lab with several thousand nodes involving interactions of multiple Autonomous Systems (ASes)—even if that test-lab is run as a set of virtual machines (VM) on a compute cluster. The problem is that many separate but coordinated device configurations need to be created—a time consuming, repetitive and error prone task.

1.2 Approach

Our work extends abstractions in AutoNetkit [17], a configuration generation tool for complex network emulations. One of the key aspects is providing reasonable defaults whenever the user is not specific. For example, IP addresses are automatically allocated. The corresponding configuration entries will be created for each VM according to the specified platform.

We aim at recreating semi-realistic behavior, which is relevant for the security evaluation of the proposed SIDR architectures. For our methodology we auto-generate what we anticipate *good players* would do. We do not aim to create attacks, or what *evil guys* might be doing. We leave this to security experts who use our tool to facilitate a lab setup. However, we believe there is value in bootstrapping the initial deployment process with semi-realistic configurations.

We call such a static, generated setup a *scenario*. An virtual topology is deployed and the related protocols, like those involved in routing and rpki, propagate information in the network. In contrast to a scenario, an *event* is manually triggered. For instance, an event could be the revocation of a certificate. An experimenter, then, can design the (periodic) automated measurements which collect the desired data into log files.

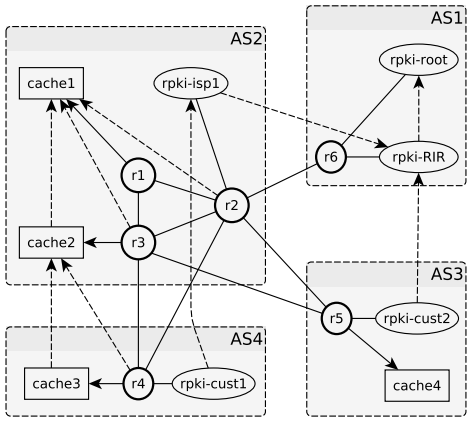

Figure 1: Example input based on our abstractions.

2 Methodology

Our goal is to enable easy set-up; therefore our toolkit has only four phases: (1) download the tool and accompanying semi-configured Linux images, e.g., the RPKI software [8]; (2) specify the abstractions (Sec. 2.2); (3) compile to configurations and deploy VMs (Sec. 2.3); (4) run experiments.

2.1 An example

Fig. 1 shows a complete configuration example, that specifies, configures and deploys a multi-AS topology. Only one configuration file, one that details access to our compute-server, is omitted. Node properties in the graphml input are used to specify properties such as in which AS the node is or if it is an RPKI server (i.e. runs rpkid, the CA daemon speaking the Up/Down Protocol with other RPKI servers), etc. In this example, *rN* (where N = 1..6) denotes a router; *rpki-X* an RPKI server; and *cacheM* (where M = 1..4) a cache server.

Each node of the topology becomes either a Junosphere virtual router or a server running on a Linux virtual machine. Every machine has a pre-installed image and receives its configuration (databases and scripts) at boot time. An 89 router network (not shown) compiled in less than a minute and deployed on Junosphere in roughly 17 minutes using 136.8GB memory. If a user wishes to host two servers, e.g. a cache and a rpki, in the same virtual machine, then the user will draw a single node. The functionality of the node will be evident not only from the node properties but also from the links adjacent to it.

2.2 Abstractions

The proposal to secure BGP consists of *certification authorities* (CAs) which create and publish X.509 certificates, ROAs, etc, and *relying parties* which fetch, validate, and use the products of the CAs [11]. Publication

is composed of the RPKI certification hierarchy, a publication repository and a backend within each ISP which signs end-entity (EE) certificates and creates *Route Origin Authorizations* (ROAs). The verification side consists of an external cache-server that crawls the publication repositories via a tool called *rcynic* [8], cryptographically validates the data, and communicates to a router via the rtr-protocol [18]. The routers perform either origin validation [19] or BGPSEC [20].

In our abstractions we strive towards simplicity, but we also need to capture the relevant structures of the SIDR framework: (1) a test-lab topology which we express in a graphml file employing node properties; (2) publication side, including the RPKI CA (Certification Authority) a hierarchy of publication servers producing ROAs; (3) a validation side of relying party cache servers, including the rtr-protocol; and (4) BGP configurations on routers.

2.2.1 Topology specification

For the test-lab topology specification we aim at roughly the same level of abstraction that one would use when drawing on a whiteboard. The graphml format allows the user to describe node properties. None of those properties are mandatory, for example *asn* specifies the AS number, but if omitted all nodes would be in one AS (AS1). A node in the graph can either represent a router or a Linux VM, the default being a router.

Most of the discussion in the following sections concerns link properties. If the value of the property **physical** is true (default value) on a link, then there is a physical connection between the nodes connected. Otherwise, the link specifies one or more *logical* relationships between the nodes. In Fig. 1, physical links are drawn as solid lines, while all the others are represented as dotted lines.

2.2.2 Publication side

RPKI certification hierarchy: The RPKI X.509 certificate hierarchy defines how the RPKI CA servers exchange data via the Up/Down, AKA Provisioning, Protocol [11]. To set this up, cryptographic identities and keys are exchanged to establish the relationship between parent and child. This relationship hierarchy follows the address space allocation hierarchy, e.g., IANA gives address space to ARIN, ARIN to ISPs, ISPs to customers. The compilation phase performs this rather complex setup exchange. The hierarchy is specified through the link property **certified_by**. This property can be configured on a directed edge between two rpki nodes. For example, in Fig. 1, both *AS2* and *AS3* get address space from the same RIR (Regional Internet Registry) and they are certified by it. *AS3* is also a customer of *AS2* and receives a certificate from the later as well.

ROAs: The RPKI exchanges and publishes certificates which attest to the ownership of address space. Those certificates do not specify who will be allowed to announce a particular address block. It is up to the certified owner to specify by what Autonomous System (AS) this block may be announced. This is done using a ROA, which associates a prefix with an authorized originating AS number. To publish ROAs, EE-certificates are generated, and these sign ROAs. As resources are automatically allocated, by default we generate and publish valid ROAs for all prefixes in the model. However, a node might use the node property *roa* to specify deviations from this rule. If the node property *roa* is set to *false* that means that no ROAs are generated or published, if it contains a number it generates the specified, potentially *invalid*, ROA.

Publication hierarchy: The third, independent, RPKI entity publishes ROAs. It is the choice of the network operators to specify whether to self-publish (run their own *pubd* or *rootd*) or delegate. The default option in our system is to self-publish. This means that the certification engine and the publication site are collocated in the same *rpki* node. The edge property **publishes_in** is used to denote that an AS wishes another AS to publish its ROAs. For example, if the link *(rpki-cust2, rpki-RIR)* in Fig. 1 has the **publishes_in** property set, then *AS3* does not maintain its own publication point, but delegates it to *AS1*. If this property is not set in the *(rpki-isp1, rpki-RIR)* link, which is the default case, then *AS2* self-publishes.

2.2.3 Verification side

The previous section described how the RPKI publishes the cryptographically verifiable information. The task of obtaining those ROAs for verification is done by relying party tools, e.g., by *rcynic* [8].

Cache server/rcynic: Rcynic collects the relevant ROA information from repositories. Suppose that a certificate *B* is issued and signed by the parent's certificate *A*. This parent certificate contains a Subject Information Access (SIA), which is a URI (Uniform Resource Identifier) pointing to the publication point of certificate *B*. All certificates are linked in this way. The Authority Information Access (AIA) is the reverse and links certificate *B* with the parent's certificate *A*. The tool, *rcynic*, starts with a preconfigured Trust Anchor Locator (TAL) and then recursively descends through the repositories using the SIAs. By default, a *cache* node runs *rcynic* and the publication repository search starts from the *rpki-root*.

If the user wishes, though, it is possible to configure a cache to download and verify the data from a custom location, e.g. another cache server. This is denoted by setting the **pulls_from** property value to true on a cache-to-cache directed link. In Fig. 1, *cache3* receives infor-

mation from *cache2* and then validates it. The protocol used in this case is not *rcynic* but *rsync*. Similarly for the *(cache2, cache1)* link. Note that there is no link departing from *cache1*, which means that it executes the default action with *rcynic*.

rtr-protocol: The rtr-protocol [18] denotes a server-router relationship in which the router pulls verified information from one or more cache servers. The link property **pulls_from** is used again to denote this relationship. If **pulls_from** is set on a server-to-server link, then is captures an *rsync* exchange as noted in the previous paragraph. However, if it is set on a router-to-server link, then it marks an rtr-protocol exchange. For example, router *r5* is pulling information from *cache4*.

2.2.4 BGP policies

Origin validation [19] can be supported by using a statement such as *(* **if** *rpki=invalid* **reject** *)*. This creates a route-map/policy-option statement to drop BGP announcements that have been marked as invalid by the router's origin validation matching rules.

2.3 Implementation

For the initial RPKI key exchange we use a modified *yamltest* script [8]. Our tool can deploy to Junosphere, GNS3 and RPKI Linux VMs running on large scale infrastructures such as StarBED [21]. All Linux VMs are configured using two disk images, (1) a modified installation with all required software and (2) an image with configuration files, which is created during the compilation phase. We focus on integrating real router vendor code as this allows us to experiment with real router code and we are therefore able to evaluate BGPSEC [20] implementations as vendors make them available.

We note that a virtualization hypervisor will distort time-dependent characteristics. Experiments requiring strict timing (of the order of seconds or less) or high-throughput cannot be tested in a realistic way. However, for the scaling and security measurements currently contemplated by vendors and operators, this should not be an issue. We believe that our approach is valid when the required accuracy of time measurements is in the order of minutes. Comparing different implementations of the same protocol or monitoring the propagation of cache updates in large-scale deployments are the kinds of experiments we are targeting.

3 Conclusion

In this paper we propose an abstract representation that allows us to experiment with the SIDR proposals to secure BGP. This abstraction is necessary to bring configuration simplicity to controlled test-labs and thence to evaluate complex protocol and system interactions. In future work, we will explore the strengths and weaknesses of such auto-generated, flexible test-labs with security experts.

Acknowledgments

We like to thank Joel Obstfeld, our shepherd Ron Ostrenga, and the anonymous reviewers for their feedback. This work is partially funded by a gift from Cisco grant #2011-89493.

References

[1] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," 2006, RFC 4271.

[2] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford, "A Survey of BGP Security Issues and Solutions," *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, January 2010.

[3] G. Huston, M. Rossi, and G. Armitage, "Securing BGP - A Literature Survey," *Communications Surveys Tutorials, IEEE*, vol. 13, no. 2, pp. 199 –222, 2011.

[4] D. Wetherall, R. Mahajan, and T. Anderson, "Understanding BGP misconfigurations," in *Proc. ACM SIGCOMM*, 2002.

[5] S. Kent, C. Lynn, and K. Seo, "Secure Border Gateway Protocol (S-BGP)," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 582 –592, 2000.

[6] R. White, "Securing BGP through secure origin BGP (soBGP)," *Business communication review*, vol. 33, no. 5, pp. 47–53, 2003.

[7] IETF Working Group, "Secure Inter-Domain Routing (sidr)." [Online]. Available: http://datatracker.ietf.org/wg/sidr/

[8] Rob Austein, Dragon Research Lab. [Online]. Available: https://trac.rpki.net/

[9] J. Mirkovic, T. Benzel, T. Faber, R. Braden, J. Wroclawski, and S. Schwab, "The DETER project: Advancing the science of cyber security experimentation and test," in *IEEE Technologies for Homeland Security*, nov. 2010, pp. 1 –7.

[10] C. Elliott and A. Falk, "An update on the GENI project," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 3, June 2009.

[11] M. Lepinski and S. Kent, "An Infrastructure to Support Secure Internet Routing," 2012, RFC 6480.

[12] R. Bush, R. Austein, S. Bellovin, and M. Elkins, "The RPKI & Origin Validation," *NANOG 52*, 2011, slide 54.

[13] T. Griffin and G. Huston, "BGP Wedgies," 2005, rFC 4264.

[14] D. McPerson, V. Gill, D. Walton, and A. Retana, "Border Gateway Protocol (BGP) Persistent Route Oscillation Condition," 2002, RFC 3345.

[15] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure, "Lossless Migrations of Link-State IGPs," *IEEE/ACM Transactions on Networking*, 2012, (To appear).

[16] A. Kapela and A. Pilosov, "Stealing the Internet - A Routed, Wide-area, Man in the Middle Attack," 2008. [Online]. Available: http://defcon.org/html/defcon-16/dc-16-speakers.html#Kapela

[17] H. Nguyen, M. Roughan, S. Knight, N. Falkner, R. Bush, and O. Maennel, "How to build complex, large-scale emulated networks," in *TridentCom*, Berlin, Germany, May 2010.

[18] R. Bush and R. Austein, "The RPKI/Router Protocol," 2012, draft-ietf-sidr-rpki-rtr-26.

[19] G. Huston and G. Michaelson, "Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs)," 2012, RFC 6483.

[20] M. Lepinski and S. Turner, "An Overview of BGPSEC," 2011, draft-ietf-sidr-bgpsec-overview-01.

[21] "StarBED Project," http://www.starbed.org/.