An Authenticated Policy-Compliant Routing

Tatarao Dagani, JNTU University, Kakinada.

Tatarao Dagani
Godavari Institute of Engineering & technology
N.H 5 Chaitanya Nagar
Velugu Banda (V)
Rajanagaram (M)
Rajahmundry, East Godavari (D)
Andhra Pradesh, India Pin :533294

***Abstract*—In today's Internet, inter-domain route control remains elusive; nevertheless, such control could improve the performance, reliability, and utility of the network for end users and ISPs alike. While researchers have proposed a number of source routing techniques to combat this limitation, there has thus far been no way for independent ASes to ensure that such traffic does not circumvent local traffic policies, nor to accurately determine the correct party to charge for forwarding the traffic.**

We present Platypus, an authenticated source routing system built around the concept of network capabilities, which allow for accountable, fine-grained path selection by cryptographically attesting to policy compliance at each hop along a source route. Capabilities can be composed to construct routes through multiple ASes and can be delegated to third parties. Platypus caters to the needs of both end users and ISPs: users gain the ability to pool their resources and select routes other than the default, while ISPs maintain control over where, when, and whose packets traverse their networks. We describe the design and implementation of an extensive Platypus policy framework that can be used to address several issues in wide-area routing at both the edge and the core, and evaluate its performance and security. Our results show that incremental deployment of Platypus can achieve immediate gains.

***Index Terms*—Authentication, capabilities, overlay networks, source routing.**

I. INTRODUCTION

The main objective of this paper is used to avoid the default and take the alternative path, at the time to check the traffic of each path. Network operators and academic researchers alike recognize that today's wide-area Internet routing does not realize the full potential of the existing network infrastructure in terms of performance, reliability or flexibility, while a number of techniques for intelligent, source-controlled path selection have been proposed to improve end-to-end performance, reliability, and flexibility. We present the design and evaluation of Platypus, a source routing system that, like many source-routing protocols before it, can be used to implement efficient overlay forwarding, select among multiple ingress/egress routers, provide virtual AS multi-homing, and address many other common routing deficiencies. The key advantage of Platypus is its ability to ensure policy compliance during packet forwarding. Platypus enables packets to be *stamped* at the source as being policy compliant, reducing policy enforcement to stamp verification. Hence, Platypus allows for management of routing policy independent of route export and path selection. Our approach to reducing this complexity is to separate the issues of connectivity discovery and path selection. Removing policy constraints from route discovery presents an opportunity for end users and edge networks. The key challenge becomes determining whether a particular source route is appropriate. ASes have no incentive to forward arbitrary traffic; currently they only wish to forward traffic for their customers or peers. We argue, however, that this is simply a poor approximation of the real goal: ASes want to forward traffic only if they are compensated for it. Henceforth, we will consider traffic *policy compliant* at a particular point in the network if the AS can identify the appropriate party to bill, and that party has been authorized by the AS to use the portion of the network in question. It is well known that multiple paths often exist between any two points in today's Internet. The central tenet of any source routing scheme is that no single route will be best for all parties. Instead, sources should be empowered to select their own routes according to whatever criteria they determine. Protocols for efficient wide-area route discovery and selection, however, are beyond the scope of this paper.

II. OVERVIEW

It is well known that multiple paths often exist between any two points in today's Internet. The central tenet of any source routing scheme is that no single route will be best for all parties. Instead, sources should be empowered to select their own routes according to whatever criteria they determine. Protocols for efficient wide-area route discovery and selection, however, are beyond the scope of this paper. We assume that the network is configured (using BGP, for example) with a set of default routes and that certain motivated parties become aware of alternative paths, either through active probing, or route discovery services . Platypus builds on this basic infrastructure, allowing entities to select paths other than the default. Packets may specify a set of waypoints to be traversed on the way to a destination, but are *not* required to specify each router along the path. A source-routed packet is forwarded using default paths between the specified waypoints; an end-to-end path is therefore a concatenation of default paths provided by the existing routing system.

Platypus is designed to be deployed selectively by ASes at choice locations in their networks. To support incremental deployment, Platypus waypoints are specified using routable IP addresses. When source routing a packet, the routing entity, which may be an end host or a device inside the network, encapsulates the payload and replaces the original destination IP address of the packet with the address of the first waypoint. The original destination IP address is stored in the packet for replacement at the last waypoint. When a Platypus packet arrives at a waypoint, the router updates the Platypus headers and forwards the packet on to the next waypoint.

III. NETWORK CAPABILITIES

Network capability is made up of two fields: a waypoint and a resource principal identifier.
The waypoint specifies a topological network location through which the packet should be routed and the resource principal specifies the entity willing to be charged for the routing request. Using intra-AS routing mechanisms, an AS can route packets for a given waypoint to different Platypus routers, thus giving it more control over the effects of source-routed traffic on an ISP's traffic engineering. For now, we will consider waypoints to correspond to a specific router within an AS.

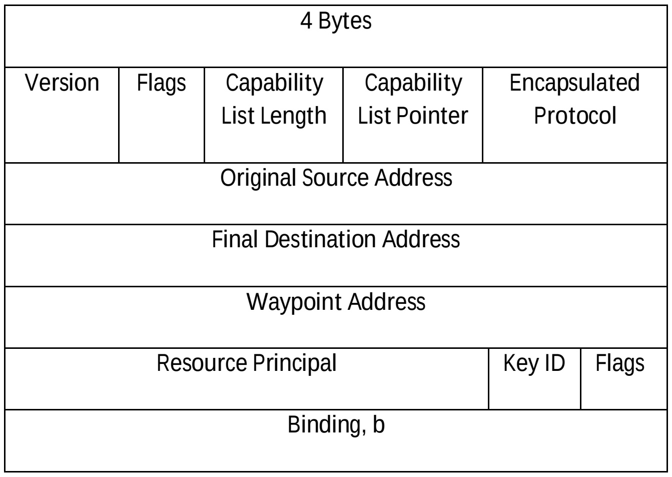

Fig. 1. Platypus header format with a single capability and binding attached.

In Platypus, packets are stamped with a source-routing request by inserting a Platypus header immediately after the IP header of each packet and including some number of capabilities, encapsulating the existing payload. Fig. 2 shows the Platypus header format with one capability attached. The header contains fields for the protocol version (currently 0), a set of bit flags a length field (specified in terms of 32-bit words), a pointer to the current capability (also in terms of 32-bit words), and an encapsulated protocol field to facilitate de-encapsulation. Capabilities are appended immediately after the Platypus header. The Platypus header and capabilities may be added by in-network stampers while the packet is in transit.

Since anyone can use a capability to forward packets through the specified waypoint and bill the indicated resource principal, Platypus must ensure that eavesdroppers watching packets in the network cannot use capabilities they observe in flight for their own packets.

A. *MAC-Based Authentication*

Platypus prevents forgery of capabilities or their bindings with the *cascade construction* of Bellare *et al.* [7], which is provably secure given an underlying MAC that is a pseudorandom function (PRF), as most modern MACs are believed to be. We define a secret temporal key, s=MACk(c),generated from the capability, c using a message authentication code (MAC) such as HMAC [24]. The MAC is keyed with k, the key of the specified waypoint. This value is securely transferred to the resource principal. In order to use a capability, an individual packet must be stamped with the capability and a binding b=MACs(MASk(P)) where MASk(P) is the invariant contents of the packet (not including Platypus headers) with the end-to-end source and destination addresses substituted and the packet length field omitted. Both and are included in the packet, as shown in Fig. 2. In this way, the binding is dependent upon both the secret key and the packet's contents, and thus cannot be reused for other packets. Similarly, any changes to the capability would render bindings computed with the secret temporal key invalid.

```
R: Revocation set, ID: Current key ID
PRO C E S S(P : Packet)
c ← ∗(P.phdr.ptr)
if |c.id−ID| > 1 or c ∈ R then
    ICMPERROR(P )
s ← MACk(c.way||c.rp||GETTIME(c.id))
b' ← MACs(MA S K(P ))
if c.b = b' then
    AC C O U N T(c.rp, P )
if P.phdr.src = 0 then
    P.phdr.src ← P.src
P.phdr.ptr ← P.phdr.ptr + |c|
if P.phdr.ptr ≥ P.phdr.len then
    P.dst ← P.phdr.dst
else
    c ← ∗(P.phdr.ptr)
    P.dst ← c.way
    FORWARD(P )
else
    ICMPERROR(P )
```

Fig. 2. Pseudocode for Platypus forwarding. P is a packet, P.src is the packet's source IP address, and P.phdr is the Platypus header in which src(dst) is the source (destination) address, ptr is the pointer to the current capability and len is the length of the capability list. c is a capability, c.way is its waypoint field, c.rp is its resource principal field, c.id is its key ID, and c.b is the binding accompanying c. | denotes concatenation.

Fig. 2 presents pseudocode for Platypus packet verification and forwarding. To verify a packet's binding (and, therefore, capability), a Platypus router only needs the local waypoint key, k, since b'=MACMACk(c)(MASK(P))=MACs(MASK(P)).if b=/b' , either the capability or the binding (or both) has been forged and the packet should be discarded. An advantage of this construction is that the router needs to maintain only a constant amount of state irrespective of the number of resource principals. In addition, rejected packets elicit ICMP responses to the sender to quell further invalid transmissions (subject to standard ICMP rate limiting).

B. Security

Security in Platypus is provided by the fact that not all parties have the information needed to bind known capabilities to new packets or create new, usable capabilities. Binding a capability to a packet requires only the temporal secret key, s, which is generated based upon and the current time. Knowledge of one capability's temporal secret key, however, does not allow a party to generate temporal secrets for others. Resource principals wishing to transfer their full rights for a particular waypoint to a trusted third party can pass both the capability and corresponding temporal secret key. While the capability can be passed in the clear, the temporal secret key must be communicated privately, ensuring that only the chosen third parties are able to receive it. These third parties can then use to generate bindings to stamp their own packets. Others, including those sniffing packets on the network, can see capabilities and their bindings, but lack the secrets required to generate valid bindings. Periodic key expiration ensures that third parties cannot use temporal secrets indefinitely. In addition, any temporal secret key may be revoked by the resource principal through communication with the key server.

IV. CAPABILITY MANAGEMENT

Platypus gains significant flexibility from the ability to transfer capabilities. Entities can collect capabilities from multiple resource principals and construct source routes to which no single entity would otherwise have rights. We describe capability management in several steps: First, we detail how capabilities are generated both in general and in special cases. Second, we describe how resource principals obtain temporal secrets for their own capabilities and capabilities delegated to them by others. Third, we present a policy framework for applying capabilities to IP packets.

A. Capability Generation

While capabilities are generally minted by an ISP, there are two important cases when individuals may wish to create new capabilities based on those provided to them by their ISPs.

1) Reply Capabilities: Protocols such as TCP have been shown to work best when forward and reverse path characteristics are similar [6]. In order to use Platypus source routes, however, both ends of a flow must have their own capabilities and perform their own routing. Platypus allows for resource principals to include a *reply capability* and its corresponding temporal secret as part of a packet stream for the recipient to use in response.

2) General Delegation: In general, a resource principal may want to specify a particular IP address prefix to which a third party may send packets using the principal's capability. Furthermore, the third party should be able to sub-delegate (specify a subnet of the previously delegated prefix) the capability without needing to contact the resource principal or key server. Platypus therefore allows the minting of *delegated capabilities*, which are derived from normal (or previously delegated) capabilities, but limited in their scope.

B. Capability Distribution

There are three main aspects to wide-area capability distribution: bootstrapping, lookup, and revocation.

1) Bootstrapping: To bootstrap the capability distribution process, we expect that each AS provides an interface (likely a Web server) through which resource principals establish their accounts.

2) Ordinary Capability Lookup: To look up the current temporal secret associated with a capability, a resource principal generates a request by encoding the capability and a special request opcode as a string and prepends it to the key-lookup subdomain (specified during the bootstrap process) in a DNS `TXT` lookup request, which is routed by DNS to an appropriate key server.

3) Delegated Capability Lookup: Lookup of delegated capabilities is fundamentally different from ordinary capability Platypus gains significant flexibility from the ability to transfer capabilities. Entities can collect capabilities from multiple resource principals and construct source routes to which no single entity would otherwise have rights.

C. Policy

So far we have discussed the mechanisms for stamping and delegation, deferring questions such as (a) how a stamper decides to stamp a particular packet and with which capabilities, (b) how a resource principal decides to delegate a capability to a peer, and (c) how its peer decides to accept a delegated capability. We now present a per-AS policy framework designed to address these questions.

V. IMPLEMENTATION

We have built prototype software components for UNIX that provide Platypus stamping, key distribution of delegated capabilities, policy specification, and forwarding services. Fig 3 depicts key components in our prototype. Each is described in turn below.

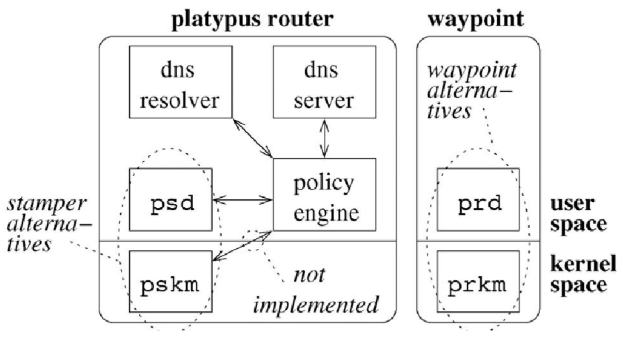

Fig.3. Overview of implemented components.

A. Forwarding and Stamping

We have implemented Platypus forwarding and stamping functionality as user-space daemons, (`prd` and `psd`), which runs in Linux and on Planetlab, and as Linux kernel modules, (`prkm` and `pskm`). While `prd` implements our full policy framework, user-level packet capture and forwarding requires multiple user/kernel context switches, resulting in poor forwarding performance. Thus, we use `prkm` to better the potential forwarding performance of an in-kernel implementation. `Prkm` processes Platypus packets entirely inside the kernel. Upon a packet arrival, in the kernel soft-IRQ context, `prkm` verifies the packet; if the binding is valid, the packet is updated and forwarded. By binding interrupt handling for different network interfaces to different CPUs on a machine, `prkm` can provide good scaling across multiple processors.

B. Distribution of Delegated Capabilities

We implemented DNS-based distribution of delegated capabilities (Section IV-B.3) using the Poslib DNS library. We leverage deployed DNS infrastructure by deferring DNS lookup work to existing DNS resolvers and servers, only performing transformations on DNS messages. For example, when a Platypus DNS server receives a `TXT` query, it transforms the query into the corresponding `A` query and lets a local conventional DNS server handle the query. On receiving the `A` response from the DNS server, the Platypus DNS server transforms the response into a `TXT` response, includes delegated capabilities as needed, and replies to the query.

C. Policy

The policy engine is implemented as a user-level process that communicates with other components in the Platypus router using an XDR-based [41] *policy protocol*. The policy protocol allows the engine to give instructions to the stamper and DNS server (Steps 0 and 5 in Fig. 4) and receive delegated capabilities from a DNS resolver (Step 4). The engine's instructions currently resemble the rule set of a firewall. (In reality these would be derived from high-level objectives created by an AS administrator.) The policy specification supports *prefix-based* matching of traffic, allowing traffic to be sent to and received from specific remote ASes through specified waypoints, and *load-sharing*, allowing some proportion of traffic to be forwarded through specified waypoints.

D. Protocol Interactions

We have attempted to design around possible negative interactions between Platypus and existing protocols. In particular, proper ICMP delivery is complicated by source routing. Since ICMP responses can occur for many reasons, the appropriate recipient of such messages can be ambiguous. For example, should an ICMP time expired message be sent to the last Platypus waypoint in the source route, the stamper, or the original source? The cause of such expiration may be due to in-network stamping or other problems such as routing loops. Further complicating the matter, non-Platypus routers may generate ICMP responses for source-routed packets and send them to the last waypoint in the source route. In both of the two primary cases—end-host stamping and in-network stamping—the end-host perceives its Platypus-enabled connectivity to be the same as ordinary network connectivity, thus we send all ICMP packets back to the original source address. The first 64 bits of the Platypus header contain the original source address, enabling RFC-compliant routers to include the original source address in ICMP error response packets; Platypus routers forward such ICMP packets along to the source, subject to standard ICMP rate limiting.

VI. DISCUSSION

During the design of Platypus, we have considered issues of performance, security, accounting, the effect of source-routed traffic upon the network, and alternative means of capability delegation. In this section we discuss these considerations.

A. Distributed Accounting

In Platypus, however, a customer may authorize third parties to inject packets into its ISP as part of a source route. Any accounting scheme that only charges customers for packets that traverse their access link clearly will not properly account for the customer's additional use. A straightforward approach would maintain counters for each resource principal at all Platypus routers within an AS, and bill for the total consumption.

B. Replay Attacks

For rate-based accounting, we can constrain resource principals to a fixed, aggregate bandwidth. However, while packet bindings cannot be forged (modulo standard cryptographic hardness assumptions), they may be replayed by an adversary, who may wish to waste a resource principal's limited bandwidth for a given capability. Since capabilities expire periodically, a natural countermeasure to replay attacks is to track packets that traverse a router within some time window and only count each distinct packet once. A Bloom filter allows for tracking of packets in such a way, but may fill up over time, resulting in false positives. This issue can be addressed by maintaining a small circular array of Bloom filters which are cleared as they fill up [2], [38]. While an adversary may be able to log all packets and replay them after the corresponding Bloom filter is emptied, if the filters are emptied only at key expiration intervals, stored packets cannot be replayed.

C. Scalability

A Platypus router does not need to keep track of permissions for end hosts, potentially providing for greater scalability. In particular, by using capabilities, Platypus is able to implement capability delegation without involving Platypus routers or key servers. The down side ,of course, of capabilities is communication overhead (28 bytes per-packet in our prototype).

D. Traffic Engineering

Conventional wisdom holds that widespread source routing deployment would complicate traffic-engineering efforts. While there admittedly is cause for concern, we have reasons for optimism. Recent simulations by Qiu *et al.* show that while sourcerouted traffic can have deleterious interactions with intra-AS traffic engineering mechanisms in extreme cases, certain techniques may be able to mitigate these effects [34]. In their studies, however, source-routed traffic was capable of completely specifying intra-AS paths. Our design for Platypus is meant to allow ISPs to specify any globally routable IP address within their IP space as a Platypus waypoint and dynamically adjust the actual (set of) internal router(s) to which the IP corresponds in response to traffic load. By dilating waypoints in this way, an ISP can meet its traffic engineering goals while delivering improved service to end hosts; we discuss this in greater detail in an earlier version of this work [35]. In addition, an ISP has the option of pricing capabilities in a way that attracts traffic to lightly loaded links or that compensates for the use of links that have little spare capacity.

E. Alternatives for Capability Distribution

The use of DNS for distribution of delegated capabilities is suited to a usage model in which a server is interested in delegating a capability to a large number of clients. While this design choice entails modifications to DNS servers and resolvers, we have found that the required changes can be made in a modular fashion, i.e., without making DNS implementations more complex. By using interposition as described in Section V-B, we maintain the separation of concerns between domain administration and capability management.

Alternatively, the server can opt to use *in-band distribution*, which is designed for transmitting delegated capabilities from receivers to senders of particular flows and does not distinguish between client and server roles. Consider a Platypus-aware traffic receiver R with IP address dst —we show how R transmits a delegated capability to a Platypus-aware traffic sender S with IP address src . The mechanism is based on inserting “Platypus signaling packets” within the flow. A Platypus signaling packet is an IP packet that has the same source and destination addresses as the flow but uses a Platypus transport protocol. Thus, the signaling packet follows the same forwarding path as the flow. periodically inserts a *delegation listen packet*, which contains a randomly generated key ks, into the flow, advertising that it is capable of receiving delegated capabilities also stores the time at which it generated ks. In response, R inserts a *delegation packet* containing the delegated capability c,t,dst and Hr = MACks(c). Upon receiving the delegation packet, S verifies and Hr checks that the corresponding key ks is recent. This process ensures that only parties on a recent default forwarding path from the S to R can have created the delegated capability, and thus prevents unauthorised diversion of packets.

VII. CONCLUSIONS AND FUTURE WORK

We argue that capabilities are uniquely well-suited for use in wide-area Internet routing. The Internet serves an extremely large number of users with an even larger number of motivations, all attempting to simultaneously share widely distributed resources. Most importantly, there exists no single arbiter (for example, a system administrator or user logged in at the console) who can make informed access decisions. Moreover, we believe that much of the complexity of Internet routing policy stems from inflexibility of existing routing protocols. We aim to study how one might implement inter-AS traffic engineering policies through capability pricing strategies. For example, an AS with multiple peering routers that wishes to encourage load balancing may be able to do so through variable pricing of capabilities for the corresponding Platypus waypoints. While properly modeling the self-interested behavior of external entities may be difficult, we are hopeful that this challenge is simplified by the direct mapping between Platypus waypoints and path selection (as compared, for example, to the intricate interactions of various BGP parameters).

REFERENCES

[1] S. Agarwal, C.-N. Chuah, and R. H. Katz, “OPCA: Robust interdomain policy routing and traffic control,” in *Proc. IEEE OPENARCH*, Apr. 2003, pp. 55–64.

[2] M. K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D. G. Andersen, M. Burrows, T. Mann, and C. A. Thekkath, “Block-level security for network-attached disks,” in *Proc. USENIX FAST*, Apr. 2003.

[3] D. G. Andersen, “Mayday: Distributed filtering for Internet services,” in *Proc. USITS*, Mar. 2003.

[4] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. T. Morris, “Resilient overlay networks,” in *Proc. ACM SOSP*, Oct. 2001.

[5] R. Atkinson, “Security architecture for the Internet protocol,” in *IETF, RFC 1825*, Aug. 1995.

[6] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, “The effects of asymmetry on TCP performance,” in *Proc. ACM Mobicom*, Sep. 1997.

[7] M. Bellare, R. Canetti, and H. Krawczyk, “Pseudorandom functions revisited: the cascade construction and its concrete security,” in *Proc. IEEE FOCS*, 1996, pp. 514–523.

[8] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway,“UMAC: Fast and secure message authentication,” in *Advances in Cryptology (CRYPTO’99)*, 1999, vol. LNCS 1666.

[9] J. Black and P. Rogaway, “A block-cipher mode of operation for parallelizable message authentication,” in *Advances in Cryptology (EUROCRYPT’ 02)*, 2002, vol. LNCS 2332.

[10] M. Caesar and J. Rexford, “BGP policies in ISP networks,” *IEEE Network*, vol. 19, no. 6, pp. 5–11, Nov. 2005.

[11] CAIDA Skitter Project. [Online]. Available: http://www.caida.org/ tools/measurement/skitter/

[12] M. Casado, T. Garfinkel, A. Akella, D. Boneh, N. McKeown, and S. Shenker, “SANE: A protection architecture for enterprise networks,” in *Proc. ACM/USENIX NSDI*, May 2006.

[13] I. Castiñeyra, N. Chiappa, and M. Steenstrup, “The Nimrod routing architecture,” in *IETF, RFC 1992*, Aug. 1996.

[14] D. D. Clark, “Policy routing in Internet protocols,” in *IETF, RFC 1102*, May 1989. RAGHAVAN *et al.*: SECURE AND POLICY-COMPLIANT SOURCE ROUTING 777

[15] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden, “Tussle in cyberspace: Defining tomorrow’s Internet,” in *Proc. ACM SIGCOMM*, Aug. 2002.