AuthedIP: Authenticated IP Layer

Daniel Chin. nq285. April 2022.
Project report for Networks and Mobile Systems with professor Anirudh Sivaraman.

1. Introduction

Ethane was proposed by Casado et al. in 2007 to provide a programmable control plane for an enterprise network. Ethane routers, when encountering new flows, ask the centralized Controller for routes to install. The Controller can thus implement policies for the enterprise network on a software level. Although Ethane offers significant controllability improvement for the enterprise network, its original proposition had several security limitations. Here I address two of them.

The first problem is with Ethane's binding between (a) the user, (b) the MAC address, and (c) the IP address. When a user appears in the network, they first authenticate with the Controller with their credentials. This endorses their IP and MAC addresses. However, multiple users may share the same addresses, e.g. with multi-user OSes and multi-VM end hosts. Ethane is blind to multi-user address sharing. The binding between the user and the addresses is therefore weak. The fundamental cause is that IP has no per-packet *user* identity.

The second problem is with Ethane's binding between (a) the packet, (b) its first Ethane router ingress port, and (c) its source addresses. Via MAC spoofing, a packet can easily lie about its source addresses, so it is impossible for Ethane to verify which end host sent the packet. The only secure information is which Ethan router ingress port the packet first went through. That is insufficient to hold packets accountable, unless the enterprise has deployed Ethane routers to all edges and each end host is connected directly to a different port on an Ethane router. Therefore, the correspondence between the packet and its source is weak. An attacker, sharing the same ingress port with a valid user, can steal that user's MAC address and send packets in their name. From Ethane's perspective, a stream of packets with the same source address arrives at one ingress port after the user authenticated successfully with the Controller. No anomaly is noticeable. More generally, a user's authentication only proves "such a user is behind such an ingress port" and nothing else. The user endorses the ingress port, and not their packets. In other words, the core problem is that Ethane has no *per-packet* user verification.

Observe that the above two problems share the same root: the lack of per-packet user verification. In this project, I design and study Authenticated IP ("AuthedIP") where every packet contains user endorsement. In an enterprise network using AuthedIP, non-users cannot send anything, and registered users cannot send anything anonymous. Anomalies can thus be traced back to registered users. In the face of Denial of Service (DoS) attacks, traditional methods usually involve designing blocking policies to pick out malicious packets. AuthedIP, on the other hand, puts the user identity in every packet, so it is much easier to tell DoS attacks from benign high-rate traffic.

2. Threat Model

Network-layer security is concerned with many types of threats, but in this project, I only focus on the Denial of Service (DoS) attack originating from within the network. DoS aims to deplete network resources and overwhelm the victim server via sending high-rate traffic to the victim. One can defend against DoS on the application layer, but that already surrenders CPU resources, and cannot solve link congestion near the victim. On the contrary, network-layer security aims to drop the DoS traffic near its source, therefore isolating the damage.

In the context of enterprise networks, DoS attacks can be further categorized according to whether the attacker is a registered user. A non-user may connect to the enterprise network and perform a DoS. A corrupted end host may perform a DoS against its user's will. A misaligned employee may willingly perform a DoS. Those cases should all be addressed.

I assume a subset of the enterprise network is physically secure. Denote this subset by "the Inside". The Inside must be a connected graph, i.e., it cannot have islands. On the Inside, no link is hijacked, no router is corrupted, and no port is exposed. Everything else is "the Outside". Any data coming from the Outside may be fabricated by an attacker. Any packet on the Outside can be sniffed by an attacker on the same Ethernet/WIFI.

3. AuthedIP Protocol

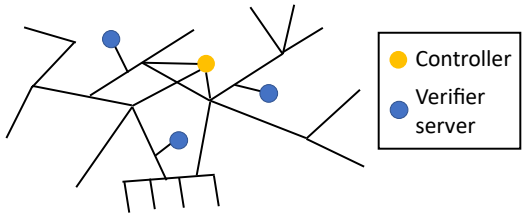

Figure 1. Example network.

The overall design of AuthedIP is:

- Each user's RSA public key is registered with the Controller.
- Each IP packet is signed by a user.
- Routers forward packets, but duplicate some of them to the nearest verifier server.
- Verifier servers check the packets' signature. If verification fails, alert the responsible router.
- When sufficiently alerted, routers enter the *verify-then-forward mode* to reject unverified packets.

The following subsections describe the complete AuthedIP protocol.

3.1. User registration

One employee may possess multiple users, each with different access privileges. Each user has an RSA key pair, the private key and the public key. The public key is 512 bits. If I was to put a public key in every packet, that will bring a 512-bit overhead per packet, which is already worse than all fields and options in the IP header combined. Therefore, each public key is associated with its 64-bit "public key ID". The public key ID is simply the 64-bit suffix of the hash of the public key. The public key ID goes into every packet, which implies an acceptable 8-byte overhead per packet. The Controller stores a hash table that maps public key IDs to public keys. New users register with the Controller by adding an entry to the hash table. A 64-bit public key ID allows $2^{64} \approx 10^{19}$ different users in the network. To avoid ID collision, use rejection sampling.

3.2. Packet

In addition to an IP packet, an AuthedIP packet has three extra fields:

- timestamp (16 bytes),
- public key ID (8 bytes),
- signature (64 bytes).

Because the IP options are limited to 40 bytes, I put the three extra fields at the beginning of the IP payload, similar to related works (Argyraki et al. 2005, Naous et al. 2011). The packet's "real payload" is thus offset by 88 bytes.

The timestamp is an integer in seconds since Epoch. The public key ID identifies the public key that verifies the packet. The signature is an RSA signature of the SHA-256 hash of the packet's *identity*. The packet's identity is the 24-byte concatenation of its timestamp, its source address, and its destination address.

This way, a packet with a correct signature shows the private key owner (i.e., the registered user)'s endorsement for the source address, the destination address, and the timestamp. This is enough to deter DoS, as will be shown in section 4.

3.3. Router

Not all routers in the network need to be AuthedIP routers. However, every border router (that separates the Inside and the Outside) has to be an AuthedIP router. An AuthedIP router knows which of its egress links point to the Inside and which links point to the Outside. This is pre-configured per-router. Also pre-configure the Controller's IP address into each AuthedIP router. An AuthedIP router maintains a suspicion score, sus, for each of its ingress port. sus is the only persistent state, and is initialized to be zero for every port.

When a packet wants to go from the Outside to the Inside, the router first looks at sus associated with the ingress port. If $sus < 4$, the ingress port is in the *forward-then-verify mode*. The router first forwards the packet normally, then randomly decides whether to check the packet. The probability of checking the packet is $P_c = 10\% \times \left(1 + \frac{sus}{10}\right)$, which starts at a 10% baseline when $sus = 0$. If the router decides to check the packet, the router makes a duplicate of the packet and wraps it in a new IP packet. The wrapper packet is sent from the router to the nearest verifier server (section 3.4), and the wrapper packet contains two extra fields:

- a 5-byte field that identifies the ingress port;
- a 1-byte flag, marked as $false$, meaning the verifier server should *not* forward this packet.

If $sus \geq 4$, the ingress port is in the *verify-then-forward mode*. Conceptually, the packet is redirected to go through a verifier as a middlebox. Concretely, the router wraps and sends the packet to the nearest verifier server (instead of its original destination). The wrapper packet has the two fields described above, but the 1-byte flag is set to $true$, indicating that the verifier server should forward the packet to its original destination if it passes verification.

When the router is alerted by a verifier server that an unverified packet from ingress port x has been detected, $sus[x]$ is incremented by 1. Each port's sus decays over time, at a rate of $0.05/sec$. With these two rules, a DoS will increase sus quickly, while sporadic verification failures due to link distortion will decay spontaneously.

3.4. Verifier Server

Verifier servers are simply some special end hosts attached to the Inside of an AuthedIP network. Their purpose is to verify packet signatures. A verifier keeps an up-to-date copy of the hash table of public keys via TCP sessions with

the Controller. For every packet, the verifier server checks three things:

- The timestamp is fresh. Specifically, less than 10 seconds old.
- The public key ID is in the hash table, and resolves to public key x.
- The RSA signature of the packet can be verified with x.

If any check fails, the packet is considered unverified. Because the wrapper packet contains the 5-byte field that identifies the border router and the ingress port, the verifier server can alert the responsible router to raise sus for the specific ingress port. At the same time, the event is logged. Bursts of verification failures may notify the network operator.

If all checks pass, the packet is considered verified. The 1-byte flag in the wrapper packet tells the verifier server whether to drop the packet or forward it to its original destination.

3.5. Controller

The Controller is a special end host attached to the Inside of an AuthedIP network. Every AuthedIP router and verifier server knows its IP address. Its job is to maintain a hash table that maps public key IDs into public keys. The hash table is then shared to all verifier servers.

3.6. End Host

When sending an IP packet, the end host must prepend the 88-byte AuthedIP fields to the IP payload and perform the signing. When receiving an IP packet, the end host must know that the "real payload" is offset by 88 bytes. The receiver has no reason to verify the signature.

3.7. Optionally: Active Subscription

This section describes an optional but important feature of AuthedIP, where verifier servers actively subscribe to border routers for packet duplication.

First consider the alternate design where border routers take the agency to duplicate packets. Two disadvantages arise:

- When duplicated packets overwhelm the network, the congestion will make it hard to instruct the border routers to back off.
- Each border router will need to learn about its nearest verifier server somehow. It is less centralized and less manageable.

In the improved design, a border router only duplicates packets to a verifier server that subscribes to it. The subscription times out automatically, so packet loss makes things stop. Only the verifier servers need to know about the border routers in its "governing area", and no router needs to be configured.

The subscription also specifies a duplication rate limit. The router never duplicates more packets than the rate limit. This way, the verifier server can dynamically match its subscription flow rate to its computing power.

The router must make sure that P_c of all forwarded packets are duplicated for verification. Therefore, when the duplication rate limit is bounding, the router must drop any incoming packet! The packet loss will look like congestion to the sender, throttling its sending. That is the desired behavior, because here the access to the Inside is bounded by the verifier server's computing power, which is analogous to a congestion bottleneck.

4. How to DoS AuthedIP

First, consider using **unverified** packets to attack.

Suppose an attacker sends a stream of unverified packets. It takes time for the first packet to be duplicated, arrive at a verifier server, get rejected, and lead to an alert. It also takes time for sus to accumulate. During this time window, unverified packets can freely flow into the network. However, such a time window is short. In my simulation (section 6.4) such a time window is shorter than one second. The victim will hardly notice such an attack.

The sus levels decay over time. An attacker can send unverified packets sporadically, waiting for sus to reach zero before sending another one. This attack disguises as random link distortions. Therefore, it must not send unverified packets at a significantly higher rate than random link distortions. That means, this attack's damage is at most as bad as random link distortions, up to a constant multiplier.

Then, consider using **verified** packets to attack.

An attacker can try to brute force the signature, but as long as we use up-to-date cryptography standards, brute forcing should take a very long time. Timestamps expire in 10 seconds, so the attacker must brute force signatures for timestamps for a planned, future attack. Every successful cracking of signature grants the attacker 10 seconds of DoS time. So, good luck.

A malware on a valid user's machine may try to steal the private key, but key files are protected in the OS. The malware may try to trick the user's programs into signing some DoS packets. Programming-wise, signing with a private key has much more salient semantics than shipping

packets, which hopefully makes it harder to inadvertently sign packets.

An attacker can replay recent packets sniffed from the network. For a sustained DoS attack, this requires a valid user in the same Outside subnet to be talking with the victim at least once every 10 seconds. That is possible. To make things worse, the signed (!) DoS packets all share the same border router with the valid conversation. It is not even clear that an attack is happening at all. AuthedIP fails to defend against such an attack. However, note that this attack requires specific circumstances.

5. After an Attack

After an attack with **unverified** packets, we know which border router the attack originated from. Either investigate that subnet on the Outside, or install more AuthedIP routers with physical security to push the frontier of the Inside. That will pinpoint the attacker's host in the future.

After an attack with **verified** packets, we know which registered user signed the packets. Either investigate that user, or check their machine for malware.

6. Simulation Results

I implement the specification of AuthedIP protocol in Python. The simulation is IP-over-socket on a localhost, and each router, end host, and verifier is simulated as a thread. For simplicity, the network topology is fixed, the routes are pre-installed, and active subscription (section 3.7) is disabled. The source code is released at github.com/Daniel-Chin/AuthedIP.

6.1. Delivery Test

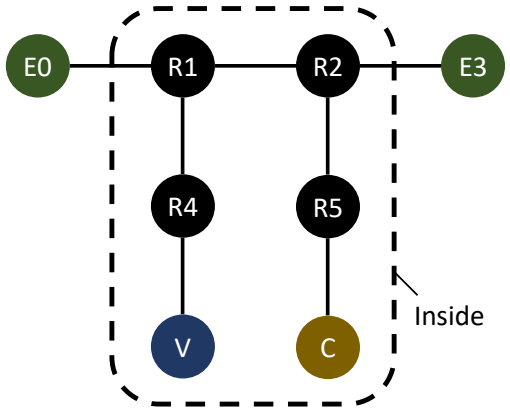

Figure 2. Simulated Network. E0 and E3 are end hosts. R1, R2, R4, R5 are routers. V is a verifier server. C is the Controller.

`sim_1_test_authedip.py` simulates a simple AuthedIP network as shown in Figure 2. E0 and E3 exchange UDP-like messages and print them to the terminal. This test mainly demonstrates the correctness of the code and the completeness of the AuthedIP protocol.

6.2. Comparing Packet Latency and Traffic Load

`sim_2_benchmark.py` compares AuthedIP against IP in terms of packet latency and network traffic load. The network topology is still the same as the previous test (Figure 2). Packet latency refers to the end-to-end latency viewed from the transport layer, so it includes packet signing, network flight (which includes router computations), and receiver parsing.

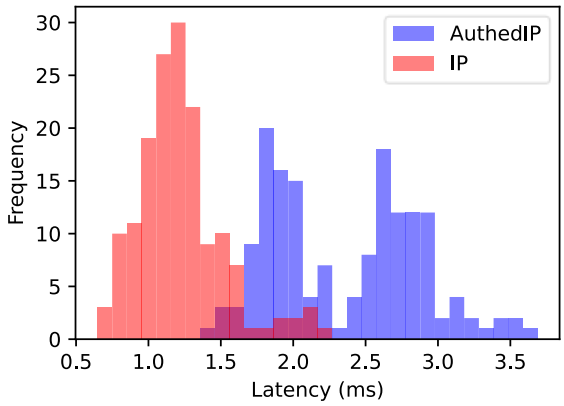

Figure 3. Latency Histogram.

Figure 3 shows the latency statistics. The simulation lasts for five seconds and delivers 164 packets of 1024 bytes. There is no unverified packets and $sus = 0$ for every port. In the Python implementation, IP has ~1.5 milliseconds of latency, while AuthedIP has ~3 milliseconds of latency. It is reasonable to conclude that deploying AuthedIP in an enterprise network will introduce no more than 3 milliseconds of latency when $sus = 0$.

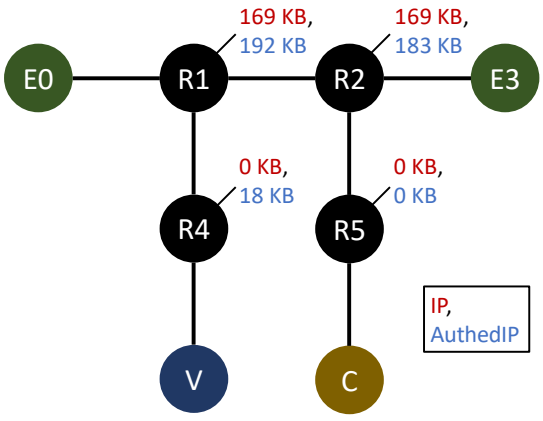

Figure 4. Traffic Load Over 5 Seconds.

The 5-second simulation also measures the traffic load at each router, which is shown in Figure 4. Packet duplication

consumes 18 KB / 5 s of throughput from R1 and R4. The 88-byte per-packet overhead consumes 14 KB / 5 s of throughput from R1 and R2. The aggregate traffic load over all routers, for AuthedIP, is 17% higher than IP.

6.3. Verification Power Puts a Bound on Throughput

`sim_3_verify_throughput.py` measures the computing cost of packet verification. A property of AuthedIP is that the verification computing power poses an upper bound on border router throughput. According to the simulation results, my CPU[1] takes on average 0.0369 ms to verify a packet. Assuming $P_c = 10\%$, we essentially amplify the throughput by $\frac{1}{10\%} = 10$ times, yielding 27107 packets per second. Assuming 60000 bytes per packet, one verifier server grants 15.1 GB/s of flux to enter the Inside. This efficiency is achieved with a single-thread Python implementation on my laptop.

6.4. Defending DoS

`sim_4_dos.py` compares IP to AuthedIP when the network is under a DoS attack. The network topology is still the same as the previous test (Figure 2).

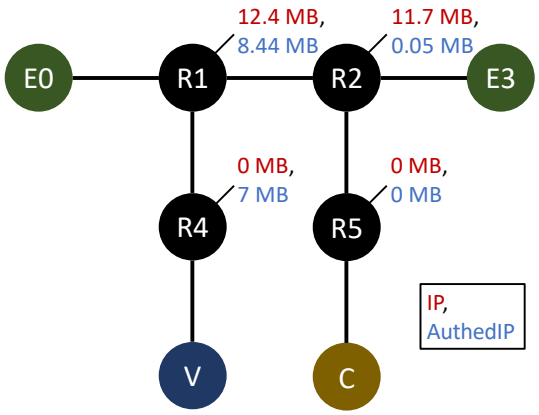

Figure 5. Traffic Load Under DoS.

Figure 5 shows the traffic load at each router. Within less than one second after DoS starts, R1 enters the verify-then-forward mode. At this point, R2 and E3 have only received 0.05 MB of DoS traffic. Afterwards, all DoS traffic is redirected to V and gets dropped. Surprisingly, even R1 experiences lower traffic load in AuthedIP than in IP (12.4 MB < 8.44 MB). That happens because V is CPU-bound, which makes R1's output queue towards V full, which makes R1 forward less packets than in IP. Under this DoS, the aggregate traffic load over all routers, for AuthedIP, is 36% lower than IP. Importantly, R2 and E3 are almost unaffected.

7. Limitations

Currently, because the timestamp increments predictably, it is possible to plan an attack in the future and slowly brute force a collection of signatures for that attack. One potential solution is to let the Controller announce a random *salt* every day. The salt is then `xor`-ed with the timestamp before being signed.

Currently, the conversations between routers, verifier servers, and the Controller are insecure. An attacker can send fake "duplicated packets" to verifier servers, or send fake alerts to border routers, essentially turning AuthedIP against itself. To solve this, I can see two directions:

- Encrypt the conversations. Because I still don't want to pre-configure too many things in the routers, the verifier servers should subscribe to routers *via the Controller* --- routers can be pre-configured to know the Controller's public key. The Controller knows all verifier servers' public key, which is fine to configure manually.
- Depend on physical security. Make sure all routes between Inside nodes stay on the Inside. Then, when packets got from the Outside to the Inside, make sure the source address does not collide with one of the Inside nodes. This way, Inside nodes can trust any packet from the Inside that claims to be from an Inside node.

As an additional limitation, AuthedIP does not defend against the replay of recent packets. That has been discussed in section 4.

8. Miscellaneous

- You want robust connectivity on the Inside, and many islands on the Outside. Having many islands makes it easier to isolate attack sources.

9. Conclusion

In this project I design and study AuthedIP, an augmentation to the Internet Protocol for the enterprise network. In an AuthedIP network, each packet is signed by a user. Non-users can be rejected within seconds. Anomalies can be traced back to its endorsing user.

[1] My processor is an 8-core 11th Gen Intel® Core™ i7-1185G7 @ 3.00GHz.

10. References

Argyraki, K. J., & Cheriton, D. R. (2005, April). Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In USENIX annual technical conference, general track (Vol. 38).

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., & Shenker, S. (2007). Ethane: Taking control of the enterprise. ACM SIGCOMM computer communication review, 37(4), 1-12.

Naous, J., Walfish, M., Nicolosi, A., Mazieres, D., Miller, M., & Seehra, A. (2011, December). Verifying and enforcing network paths with ICING. In Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies (pp. 1-12).