

How to Build Incoming Tables?

- We have to protect path information
 - So that even rerouting cannot hurt the protocol
- We have to accommodate legacy routers
- We have to deal with route changes that happen at legacy routers

1

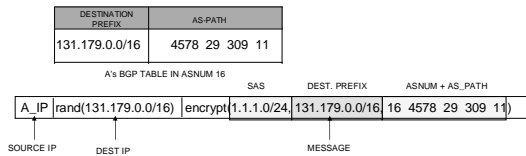
ASPIRE

- Work in progress at LASR lab, UCLA
- We have to protect path information
 - But noone knows complete path!
 - We will work at AS level – BGP routers know AS path
- We have to accommodate legacy routers
 - We will choose dIP to be from the destination AS, packets will fly there over legacy routers
- We have to deal with route changes that happen at legacy routers
 - Let's think about this later ...

2

ASPIRE Overview

- Each ASPIRE router is assigned a *source address space (SAS)*
- Routers exchange messages, use BGP routing table information



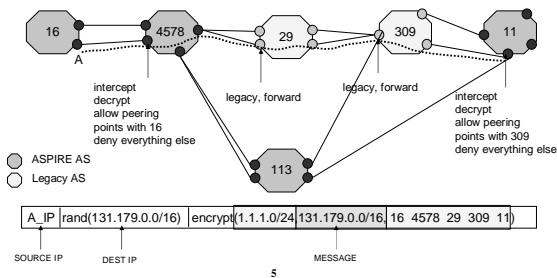
3

ASPIRE Overview

- Messages sent at bootstrap
- Messages sent when AS_PATH changes
 - Add message along the new path
 - Explicit teardown along the old path
- Incoming table entry info
 - SAS
 - Destination prefix
 - Incoming interface

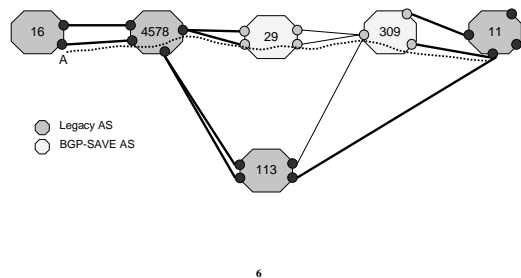
4

Message Processing



5

Message Processing



6

ASPIRE

- Route changes are still a problem
 - We can pretty much infer router-level changes – changes in peering relationship are easy to detect
 - Problem when a legacy AS X decides to change the AS level path
 - Change will eventually propagate to endpoints but it will take time – legitimate packets will be dropped during this time

StackPi

- Use packet-marking to derive incoming tables
- Routers will mark passing packets *deterministically*
- Those marks will be used to build packet tables
- Marks could also be used to devise filters for DDoS defense

"StackPi: A New Defense Mechanism against IP Spoofing and DDoS Attacks",
A. Perrig, D. Song, A. Yaar, CMU technical report

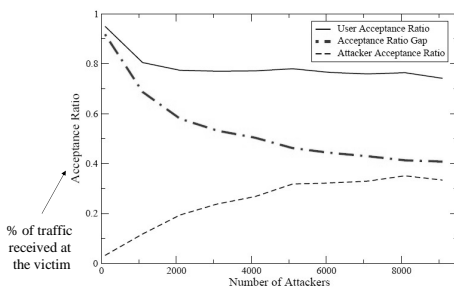
StackPi

- IP identifier field is used for marking
- Each router performs MD5 hash on its incoming link (IP_previous_hop|my_IP) and takes last n bits
- Each router shifts IP identifier contents n times to the left and concatenates his mark
- For $n=2$ we have 4 distinct marks and we can fit 8 marks in the field
 - Routers closest to the victim should not mark
- If there is a few legacy routers, we should mark for them – encode (IP_previous_hop|next_hop_IP)

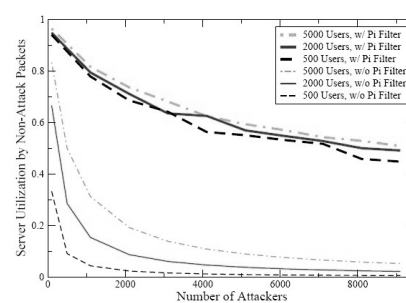
StackPi

- How can this help us build incoming tables?
 - Use machine learning to associate source IP with the mark
 - How big will tables be?
- How can this help us defend against DDoS?
 - Use marks to characterize packets belonging to a flood
 - Filter only if fraction of attack in marked traffic is significant
 - Or use combination of packet mark and source IP
 - What will happen with legitimate traffic?
 - What will happen in case of a very distributed attack?

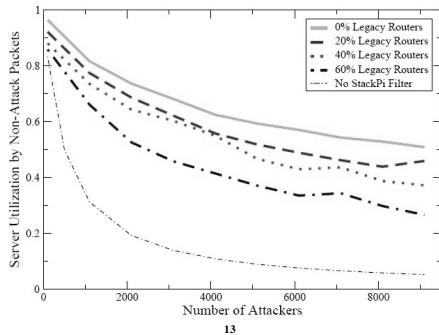
StackPi



StackPi



StackPi



13

StackPi

- What about route changes?
 - Maybe we can use machine learning to infer which addresses are from the same subnet
 - When all of them change path assume route change has happened

14

Hop-Count Filtering

- Learn from observing incoming traffic what is the proper TTL value for a given source IP address
 - Use this information to build incoming tables
 - Use information from established TCP connections only to defeat subversion by attackers
 - Tables actually contain hop count inferred from TTL
- We can also use traceroute to learn the hop count
- Filter starts to discard packets only upon DDoS attack

"Hop-Count Filtering: An Effective Defense Against Spoofed Traffic"
C. Jin, H. Wang, K.G. Shin, University of Michigan technical report

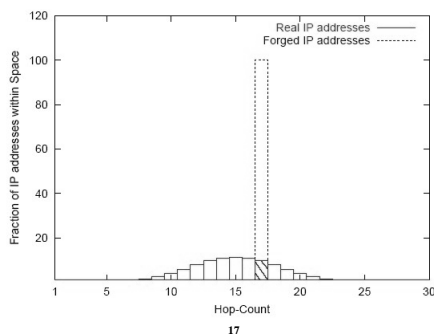
15

Hop-Count Filtering

- Hop count distributions and they follow normal distribution – there is enough variability
- For the attacker to pass his packets through
 - He has to guess proper TTL to put in packets
 - He cannot use random spoofing anymore (too many TTLs to guess). It is thus easier to characterize malicious packets.

16

Non-Spoofing Single Attacker



17

Hop-Count Filtering

- Hop count filter is usually inactive
 - It samples traffic at random intervals and computes the probability that the traffic is spoofed – based on mismatches with incoming table
 - If this probability is above threshold HCF starts filtering
 - At all times tables are updated looking at random established TCP connection

18