# # SYN attack scenarios

We define a legitimate host as a non-spoofed host.

We define the following scenarios for SYN attacks:

## 1. Attacks that establish the 3-way handshake

1.1 SYN attack from a single legitimate source

1.2 SYN attack from a number of legitimate sources in network /16

1.3 SYN attack from a number of legitimate sources in network /20

1.4 SYN attack from a number of legitimate sources in network /24

## 2. Attacks that fail to establish the 3-way handshake

2.1 SYN attack from a single source - reflection attack?

2.2 SYN attack from a number of spoofed sources in network /16

2.3 SYN attack from a number of spoofed sources in network /20

2.4 SYN attack from a number of spoofed sources in network /24

We need to identify what is the potential damage to applying TCP SYNPROXY over these potential scenarios:

1.x The originating hosts will successfully establish the TCP connection with the SYNPROXY, the possible damage to our system is a potential (D)DoS by exhausting the resources of the server.

SOLUTION: Whatever leaks from the TCP SYNPROXY can be then rate limited before it is sent/fed to the Circular Pool. The rate limitation can happen in the Filter.OUTPUT chain of the SYNPROXY node itself or/and on the Circular Pool in the Mangle.PREROUTING chain of the Realm Gateway node.

2.x We receive a high load of traffic from the same "source". Because the source fails to establish the TCP connection with the SYNPROXY we begin to collect evidence of spoofing. For every spoofed SYN there is the correspondent SYN,ACK response, the SYNPROXY behaves as a reflector for an attack. Failing to send a SYN,ACK response effectively means denying that particular connection attempt.

2.1 We need to understand what is the real damage of sending SYN,ACK response to the source. A standard SYN,ACK response contains 16(ether)+20(ip)+40(tcp)=76 bytes per packet (448 bits). At a rate of 1M packets per second, this means sending 427 Mbps of data traffic only in SYN,ACK segments. If we decide to accept 10K new connections per second we could potentially be sending 4.27 Mbps of junk traffic.

SOLUTION: In order for the SYNPROXY to minimize the potential reflection of SYN,ACKs we could include 2 global counters for SYN,ACKs sent and ACK received. The difference between these counters constitute the number of unresponded SYN,ACK responses. However, we need to be careful with the way we process these counters, as the both the SYN,ACKs and ACKs can still be on the fly or else indicate spoofed connection attempts. Using the global counters results very cheap in terms of memory allocation and computing processing. The benefit is that we are able to filter out spoofed connection attempts to certain rates, but if a serious SYN attack is triggered, we may be blocking SYN,ACK responses for legitimate connections, effectively resulting in DoS for all source addresses.

2.2 In the previous case we learned that we can potentially generate high rates of undesirable SYN,ACK packets to a targeted host and also block all new incoming connections if the SYNPROXY starts dropping SYN requests. As a result our network reputation could be negatively impacted due to the unwanted traffic and our hosts are also unable to receive new connections.

SOLUTION: Store a limited amount of state information with certain network slicing, e.g. /16. That means we can have a table of 65K entries to store data related to that /16 network. The information and algorithms required are discussed later on this document. With this approach we can decide whether to process the send a SYN,ACK with the coarse granularity of /16 network accuracy. If an attack is not highly widespread, we can still process new connection attempts from untainted sources.

2.3 and 2.4 Applying the solution devised in 2.2 we are still blocking a /16 network in case of attack.

SOLUTION: We can gain in network accuracy by using a larger network address in our state information. The implementation can have 2 tables. The first table is 65K in length and statically indexed to each of the /16 networks available. The second table can be defined as a hash table with a capacity Cx. If the number of unanswered SYN,ACKs grow over a threshold T0 in Table1, we can create a new entry in Table2 with a higher resolution for the network address. The entry in Table1 must indicate a further lookup is required in Table2. It is desirable the same algorithm can be applied for any table entry. The algorithm for creating entries dynamically in Table2 is not discussed here.

## Caveats and considerations:

\* We need to control the amount of state information that will be required on each table so it does not skyrocket. The entries in Table1 that require other entries in Table2 for greater accuracy must point to them as to prevent memory leaks.

\* We need to devise a per-packet behavior to consider the current state of the counters for a given entry and quickly reacts to burst attacks and high loads.

\* We need to devise a time-based algorithm that learns from past events and keeps historic data for load calculations and thresholds.

\* We need to devise a solution that integrates both per-packet and time-based triggers to protect against burst-attacks and takes into consideration past events.

\* The SYNPROXY in the Filter.INPUT table seems to generate correctly the RST,ACK response to the remote client. This is not the case when the SYNPROXY is configured in an intermediary gateway and the functionality is called from the filter.FORWARD chain. In this case, the TCP handshake is successfully established with the remote client, however if the server generates a RST,ACK upon receiving the initial SYN, the response is not properly modified by the SYNPROXY and makes the remote client hang and retransmit a previous TCP-segment (with also wrong fields!).

## TODO

\* Find the exact public address space for table dimensioning

\* Define in a clear way the main objectives of using SYNPROXY and the potential tradeoffs

\* Define the metrics and success criteria from the SYNPROXY point of view

\* Define the metrics and success criteria from an attacker point of view