A tale of Nefariuos usage: IPv6 Covert Channels

Andrea Ercolino

andrea@ercolino.me

The views and opinions expressed in this presentation are solely my own and do not reflect those of my employer.
This presentation is a personal initiative and is not affiliated with, endorsed by, or conducted on behalf of my employer.

Our tale began

Once upon a time, in 1995, the tale of IPv6 began...

Why IPv6

IPv6 is the favorite protocol of King's emissaries to send messages of peace and harmony.

Its fixed header length makes it easy to handle, and thanks to its increased address space it can reach the most remote castles of the kingdom.

The evil wizard

But an evil wizard found a way to curse IPv6 packets and uses it to hide messages of war in the notes delivered by the King's emissary.

Our role in the story

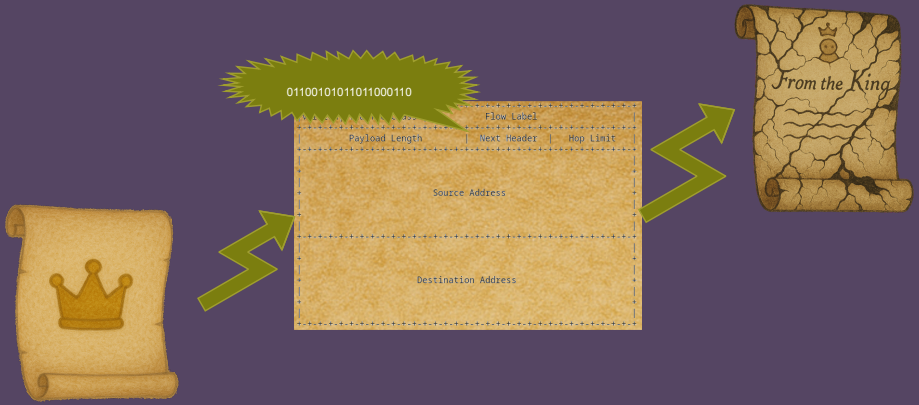
Today, we are role-playing as the evil wizard.

What is a covert channel...

A covert channel can be defined as any communication channel that can be exploited by a process to transfer information in a manner that violates the security policy of the system.

... and why we would want to use it

What an IPv6 Covert Channel Looks like



Implementation Goals

- Six covert channels implemented
- Three Communication Strategies... more on this later
- Extensible Interface
- The legit communication channel is not affected
- The exfiltrated data integrity is checked

Covert Channels Implemented & Notable Challenges

- Flow Label. Working using a size not multiple of 8.
- Traffic Class. Small amount of bandwidth, value of the field is commonly zeroed.
- Authentication Header. The computing time necessary to build an acceptable header.
- Routing Extension Header. Commonly discarded when exploited to carry a covert channel.
- Destination Options Header. The computing time necessary to build an acceptable header.
- Fragment Extension Header. Commonly discarded.

Covert channels are implemented through a combination of **iptables** rules, **Netfilter Queue** and **Scapy**.

Communication strategies

- **Naive**. Simple, works if covert sender and covert receiver are synchronized
- **Marking**. Sender marks packets with a hash of the concatenation of a shared secret and the number of the current covert data chunk
- **Reliable Marking**. If the transport layer used is TCP, the retransmission of covert packets can be achieved via a buffer and a check on the TCP sequence number of the intercepted packet.

How to not affect the legit communication

- Data are injected only into fields that have little to no effect on the overt communication.
- When using a fake extension header, it is initialized in order for the packet to be correctly routed.
- When exfiltrated, the packet is restored in its original form.

Example. How to not affect the legit communication

```
pad_len = (4 - (len(value_bytes) % 4)) % 4
if pad_len == 0:
    pad = 0
else:
    pad = b"\x00" * pad_len

authentication_header = AH(
    nh=pkt.nh,
    # As per RFC 4302 The payload lenght is calculated in 32-bit words - 2.
    payloadlen=((MAGIC_AUTH_HEADER_BASE_LEN_BYTE + len(value_bytes)) // 4 - 2),
    spi=1,
    seq=self._sent_received_chunks,
    icv=value_bytes,
    padding=pad,
    reserved=0,
)
new_pkt /= authentication_header
new_pkt.plen = pkt.plen + len(authentication_header)
new_pkt /= pkt[1:]
return new_pkt
```

Figure: Exploiting Authentication Header

```
    clean_pkt = IPv6(
        src=pkt.src,
        dst=pkt.dst,
        nh=pkt[1].nh,
        fl=pkt.fl,
        plen=len(pkt[2:]),
        hlim=pkt.hlim,
        version=pkt.version,
    )
    clean_pkt /= pkt[2:]
else:
```

Figure: Restoring the packet

How to check the integrity of the exfiltrated data

- Data is split in a byte array
- Before injection an hash of the data array is computed
- The first n covert packets carry the hash
- The receiver node computed the hash of the covert data received and compares it to the hash received

Example. How to check the integrity of the exfiltrated data

```python
with open(file_path, "rb") as f:
    buffer = 0
    bits_in_buffer = 0
    mask = (1 << chunk_size_bits) - 1  # mask to extract chunk

    while True:
        byte = f.read(1)
        if not byte:
            break

        # Add the byte to the buffer
        buffer = (buffer << 8) | byte[0]
        bits_in_buffer += 8

        # Extract chunks while enough bits are available
        while bits_in_buffer >= chunk_size_bits:
            bits_in_buffer -= chunk_size_bits
            chunk = (buffer >> bits_in_buffer) & mask
            yield chunk

    # Optional: handle remaining bits (zero-padded final chunk)
    if bits_in_buffer > 0:
        chunk = (buffer << (chunk_size_bits - bits_in_buffer)) & mask
        yield chunk
```

Figure: Split data in chunks of n bits

```python
for chunk in chunks:
    buffer = (buffer << chunk_size_bits) | chunk
    bits_in_buffer += chunk_size_bits

    while bits_in_buffer >= 8:
        bits_in_buffer -= 8
        byte = (buffer >> bits_in_buffer) & 0xFF
        hash_func.update(bytes([byte]))

# Handle any remaining bits (pad with zeros on the right)
if bits_in_buffer > 0:
    byte = (buffer << (8 - bits_in_buffer)) & 0xFF
    hash_func.update(bytes([byte]))

hash_values = hash_func.digest()
```

Figure: Calculating hash of data chunks

Example. How to check the integrity of the exfiltrated data

```
self._chunks_file = list(
    split_file_into_chunks_bytes(self._exfiltrationpath, self._chunk_size)
)
self._chunks_hash = list(
    create_hash_of_file_bytes(
        split_file_into_chunks(self._exfiltrationpath, self._chunk_size),
        self._chunk_size,
    )
)

self._chunks = self._chunks_hash + self._chunks_file
```

Figure: Building chunks to inject

```
value = int.from_bytes(pkt[1].icv, byteorder="big")
if self._sent_received_chunks == 0:
    self._number_chunks = value
elif self._sent_received_chunks < self._number_chunks:
    if (self._sent_received_chunks - 1) < self._number_of_hash_chunks:
        self._chunks_hash.append(value)
    else:
        self._chunks_file.append(value)
self._sent_received_chunks += 1
```

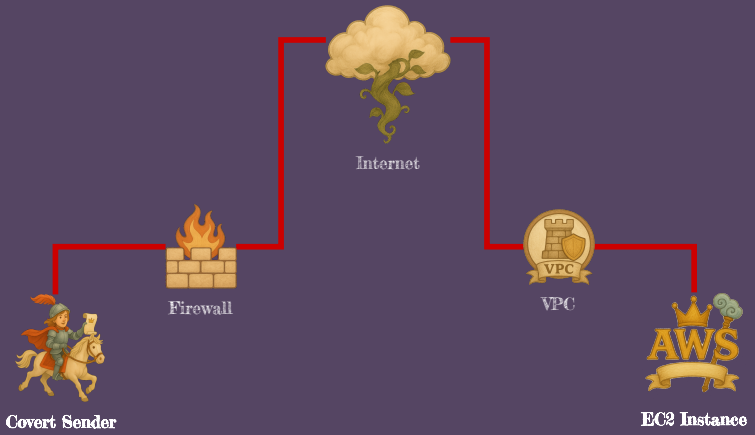
Figure: Retrieval of injected chunks

Usability in the wild

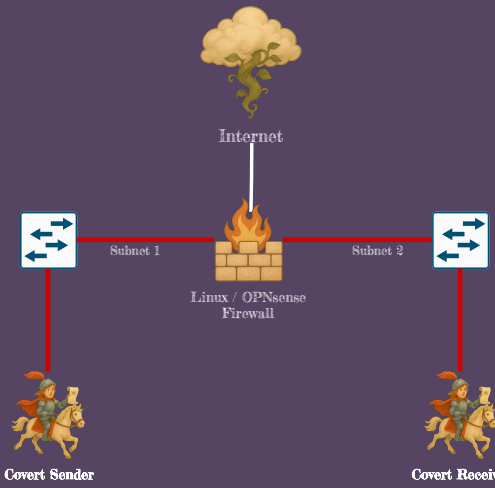
Scenarios in which an IPv6 Covert Channel was exploitable:

- IPv6 nodes in different LANs
- IPv6 nodes in the AWS cloud
- Hybrid setting between on-premises node and cloud native node

Usability in the wild



Usability in the wild



Usability in the wild. Details 1

	AWS	**Azure**	**Vultr**
Packet Normalization between VPCs	No	Yes	Yes
Packet Normalization in egress/ingress from VPC	Yes	Yes	Yes
Drop Crafted Extension Headers	No	Yes	No
Fragmentation Allowed	No	No	No
Exfiltration Involving On Premises Device	Partial	No	Possible

Usability in the wild. Details 2

	Suricata	**Snort**	**Zeek**
Flow Label	NA	NA	NA
Authentication	STREAM pkt seen on wrong thread	NA	NA
Destination Options	NA	NA	NA
Routing	SURICATA RH Type 0	IPv6 routing type 0 extension header	routing0 hdr
Fragment	reserved field in Frag Header not zero, FRAG IPv6 Fragmentation overlap	(stream ip) short fragment, possible DOS attempt	excessively large fragment

IPv6 Covert Channels in action

Thanks for the attention! Questions?

Three things cannot be long hidden: the sun, the moon, and the truth.
-Buddha

References I

Blumbergs, B., Pihelgas, M., Kont, M., Maennel, O., and Vaarandi, R. (2016). Creating and detecting ipv6 transition mechanism-based information exfiltration covert channels. pages 85–100.

Caviglione, L. (2021). Trends and challenges in network covert channels countermeasures. *Applied Sciences*.

Caviglione, L., Schaffhauser, A., Zuppelli, M., and Mazurczyk, W. (2022). Ipv6cc: Ipv6 covert channels for testing networks against stegomalware and data exfiltration. *SoftwareX*, 17:100975.

Iglesias, F., Meghdouri, F., Annessi, R., Zseby, T., and Li, W. (2022). Ccgen: Injecting covert channels into network traffic. *Sec. and Commun. Netw.*, 2022.

J. Abley, P. S. (2007). Deprecation of type 0 routing headers in ipv6. RFC 1, RFC Editor.

Kent, S. (2005a). Ip authentication header. RFC 1, RFC Editor.

Kent, S. (2005b). Ip encapsulating security payload (esp). RFC 1, RFC Editor.

References II

Li, F., An, C., Yang, J., Wu, J., and Zhang, H. (2014). A study of traffic from the perspective of a large pure ipv6 isp. *Computer Communications*, 37:40–52.

Lucena, N. B., Lewandowski, G., and Chapin, S. J. (2006). Covert channels in ipv6. pages 147–166.

Makhdoom, I., Abolhasan, M., and Lipman, J. (2022). A comprehensive survey of covert communication techniques, limitations and future challenges. *Computers & Security*, 120:102784.

Mazurczyk, W., Powójski, K., and Caviglione, L. (2019). Ipv6 covert channels in the wild. In *Proceedings of the Third Central European Cybersecurity Conference*, CECC 2019, New York, NY, USA. Association for Computing Machinery.

Nikkhah, M. (2016). Maintaining the progress of ipv6 adoption. *Computer Networks*, 102:50–69.

Nikkhah, M. and Guérin, R. (2016). Migrating the internet to ipv6: An exploration of the when and why. *IEEE/ACM Transactions on Networking*, 24(4):2291–2304.

References III

Ondov, A. and Helebrandt, P. (2022). Covert channel detection methods. In *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 491–496.

S. Deering, R. H. (2017). Internet protocol, version 6 (ipv6) specification. RFC 1, RFC Editor.

Simmons, G. J. (1984). *The Prisoners' Problem and the Subliminal Channel*, pages 51–67. Springer US, Boston, MA.

Wang, J., Zhang, L., Li, Z., Guo, Y., Cheng, L., and Du, W. (2022). Cc-guard: An ipv6 covert channel detection method based on field matching. pages 1416–1421.