FragAttacks: Fragmentation & Aggregation Attacks against Wi-Fi

Mathy Vanhoef

Draft version 1, 8 March 2021
### Table of contents

#### Design flaws:

1. [Design flaw: aggregation attack](#) (CVE-2020-24588)
2. [Design flaw: mixed key attack](#) (CVE-2020-24587)
3. [Design flaw: fragment cache attack](#) (CVE-2020-24586)
Table of contents

Implementation flaws allowing trivial plaintext injection:
4. Accepted plaintext frames (CVE-2020-26140 / 26143)
6. Plaintext broadcast fragments (CVE-2020-26145)
7. Cloacked aggregated frames (CVE-2020-26144)

Other implementation flaws:
8. Pre-auth EAPOL forwarding (CVE-2020-26139)
9. Non-consecutive packet numbers (CVE-2020-26146)
10. Mixed plain/encrypted fragments (CVE-2020-26147)
11. No fragmentation support (CVE-2020-26142)
Aggregation Attack

CVE-2020-24588
Background

Sending small frames causes high overhead:

header  packet1  ACK  header  packet2  ACK  header
Background

Sending small frames causes high overhead:

```
header packet1 ACK header packet2 ACK header
```

This can be avoided by **aggregating frames**:

```
header’ packet1 packet2 ... ACK
```

**Problem:** how to recognize aggregated frames?
Recognizing aggregated frames

<table>
<thead>
<tr>
<th>header</th>
<th>payload</th>
</tr>
</thead>
</table>

Recognizing aggregated frames

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recognizing aggregated frames

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>rfc1042</td>
<td></td>
</tr>
</tbody>
</table>
Recognizing aggregated frames

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>rfc1042</td>
<td>IP header</td>
</tr>
</tbody>
</table>
Recognizing aggregated frames

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>rfc1042</td>
<td>IP header</td>
</tr>
<tr>
<td>True</td>
<td>metadata</td>
<td>length</td>
</tr>
</tbody>
</table>

RFC 1042: IP header, TCP header, data
Recognizing aggregated frames

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>rfc1042</td>
<td>IP header</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCP header</td>
</tr>
<tr>
<td>True</td>
<td>metadata</td>
<td>packet1</td>
</tr>
<tr>
<td></td>
<td>length</td>
<td>metadata</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>packet2</td>
</tr>
</tbody>
</table>
Recognizing aggregated frames

Authenticated

header | aggregated? | payload

Authenticated & encrypted

False | rfc1042 | IP header | TCP header | data

True | metadata | length | packet1 | metadata | length | packet2
Recognizing aggregated frames

**Not authenticated**: adversary can flip value

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>rfc1042</td>
<td>IP header</td>
</tr>
<tr>
<td>True</td>
<td>metadata</td>
<td>length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What happens if we flip this flag?
## Spoofing aggregated frames

<table>
<thead>
<tr>
<th>False</th>
<th>rfc1042</th>
<th>IPv4 hdr</th>
<th>TCP hdr</th>
<th>data</th>
</tr>
</thead>
</table>
### Spoofing aggregated frames

<table>
<thead>
<tr>
<th>False</th>
<th>rfc1042</th>
<th>IPv4 hdr</th>
<th>TCP hdr</th>
<th>data</th>
</tr>
</thead>
</table>

Adversary **turns normal frame into aggregated** one

True
Spoofing aggregated frames

<table>
<thead>
<tr>
<th>False</th>
<th>rfc1042</th>
<th>IPv4 hdr</th>
<th>TCP hdr</th>
<th>data</th>
</tr>
</thead>
</table>

Adversary turns normal frame into aggregated one

True | metadata

› Metadata = rfc1042 & IPv4 hdr → 1\textsuperscript{st} sub-packet is ignored
Spoofing aggregated frames

Adversary turns normal frame into aggregated one

- Metadata = rfc1042 & IPv4 hdr → 1\textsuperscript{st} sub-packet is ignored
- Length = IPv4 ID field: determines start of 2\textsuperscript{nd} sub-packet
Spoofing aggregated frames

<table>
<thead>
<tr>
<th>False</th>
<th>rfc1042</th>
<th>IPv4 hdr</th>
<th>TCP hdr</th>
<th>data</th>
</tr>
</thead>
</table>

Adversary turns normal frame into aggregated one

<table>
<thead>
<tr>
<th>True</th>
<th>metadata</th>
<th>length</th>
<th>ignored</th>
</tr>
</thead>
</table>

- Metadata = rfc1042 & IPv4 hdr $\rightarrow$ 1\textsuperscript{st} sub-packet is ignored
- Length = IPv4 ID field: determines start of 2\textsuperscript{nd} sub-packet
## Spoofing aggregated frames

<table>
<thead>
<tr>
<th>False</th>
<th>rfc1042</th>
<th>IPv4 hdr</th>
<th>TCP hdr</th>
<th>data</th>
</tr>
</thead>
</table>

Adversary **turns normal frame into aggregated one**

<table>
<thead>
<tr>
<th>True</th>
<th>metadata</th>
<th>length</th>
<th>ignored</th>
<th>metadata</th>
<th>length</th>
<th>packet</th>
</tr>
</thead>
</table>

- Metadata = rfc1042 & IPv4 hdr → 1\textsuperscript{st} sub-packet is ignored
- Length = IPv4 ID field: determines start of 2\textsuperscript{nd} sub-packet
- Control IP ID & part of TCP data → **inject arbitrary packets**
How to control IPv4 ID and TCP data?

Send IPv4 packets to the victim

› Client: if there’s a firewall, trick client to visit our website
  » Allows adversary to send IP/TCP packets, not any others

› AP: can attack AP if a client uses predictable IP IDs

All major operating systems affected

› 802.11n mandates support for aggregated frames

› Excluding Net/OpenBSD and certain IoT devices
Recap: can inject arbitrary packets

1. Inject special IPv4 frame to the victim
2. Adversary will intercept the corresponding Wi-Fi frame, set the aggregated flag, and forward it to the victim

Example attacks:
› Inject IPv6 Router Advertisement with our DNS server
› Victim will use our DNS server → intercept IP-based traffic
› Against IPv4-only clients we performed a port scan
Mixed Key Attack
CVE-2020-24587
Background

Large frames have a high chance of being corrupted:

<table>
<thead>
<tr>
<th>header</th>
<th>packet</th>
<th>ACK</th>
</tr>
</thead>
</table>
Background

Large frames have a high chance of being corrupted:

Avoid by **fragmenting** & only retransmitting lost fragments:

Problem: **how to (securely) reassemble** the fragments?
### Reassembling plaintext fragments

<table>
<thead>
<tr>
<th>header</th>
<th>fragment1</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>fragment2</td>
</tr>
<tr>
<td>header</td>
<td>fragment3</td>
</tr>
</tbody>
</table>
Reassembling plaintext fragments

- All Wi-Fi frames have an incremental sequence number $s$
- Fragments of a frame have the same sequence number $s$
Reassembling plaintext fragments

- All Wi-Fi frames have an incremental sequence number $s$
- Fragments of a frame have the same sequence number $s$
- All fragments also have a fragment number ...
Reassembling plaintext fragments

<table>
<thead>
<tr>
<th>header</th>
<th>$s$</th>
<th>0</th>
<th>More</th>
<th>fragment1</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>$s$</td>
<td>1</td>
<td>More</td>
<td>fragment2</td>
</tr>
<tr>
<td>header</td>
<td>$s$</td>
<td>2</td>
<td>Last</td>
<td>fragment3</td>
</tr>
</tbody>
</table>

- All Wi-Fi frames have an incremental sequence number $s$
- Fragments of a frame have the same sequence number $s$
- All fragments also have a fragment number ... 
  ... and a flag to identify the last fragment
Reassembling encrypted fragments

<table>
<thead>
<tr>
<th>header</th>
<th>$s$</th>
<th>$n$</th>
<th>0</th>
<th>More</th>
<th>fragment1</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>$s$</td>
<td>$n+1$</td>
<td>1</td>
<td>More</td>
<td>fragment2</td>
</tr>
<tr>
<td>header</td>
<td>$s$</td>
<td>$n+2$</td>
<td>2</td>
<td>Last</td>
<td>fragment3</td>
</tr>
</tbody>
</table>

- All encrypted frames have a packet number to detect replays
  - Cannot reuse sequence number (not unique & too small)
Reassembling encrypted fragments

- All encrypted frames have a packet number to detect replays
  - Cannot reuse sequence number (not unique & too small)
- Everything except sequence number is authenticated
- Drop if packet & fragment numbers are not consecutive

<table>
<thead>
<tr>
<th>Header</th>
<th>s</th>
<th>n</th>
<th>0</th>
<th>More</th>
<th>Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>fragment1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>fragment2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>fragment3</td>
</tr>
</tbody>
</table>

Authenticated | Authenticated
Problem: key renewal

- Session key can be periodically renewed ...
- ... or updated when roaming between APs

- Receiver is allowed to reassemble fragments encrypted under different keys (i.e. mixed keys)
- Attacker can trick victim to combine fragments (from two different frames) encrypted under mixed keys
- Can be abused to exfiltrate data under specific conditions
Mixed Key Attack

Visit attacker website

Frag1(s, n)

Frag2(s, n+1)

Decrypted & store fragment

Frag1(s, n)
Mixed Key Attack

Visit attacker website

Frag1(s, n)

Frag2(s, n+1)

Decrypt & store fragment

<table>
<thead>
<tr>
<th>Header (Frag1)</th>
<th>Payload (Frag2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2 to 3.5.1.1</td>
<td>GET /image.png HTTP/1.1</td>
</tr>
</tbody>
</table>
Mixed Key Attack

Visit attacker website

Frag1(s, n)

Frag2(s, n+1)

Decrypt & store fragment

Renew session key

Frag1(s, n)
Mixed Key Attack

Visit attacker website

Frag1(s, n)

Frag2(s, n+1)

Frag1(s, n)

Decrypt & store fragment

Renew session key

Packet numbers will restart from zero
Mixed Key Attack

Visit attacker website
Frag1(s, n)
Frag2(s, n+1)
Renew session key
Login to website

Decrypt & store fragment
Frag1(s, n)
Mixed Key Attack

Visit attacker website

Frag1(s, n)
Frag2(s, n+1)

Frag1(s', n)
Frag2(s', n+1)

Renew session key

Decrypt & store fragment

Login to website

Frag1(s', n)
Frag2(s', n+1)

Frag2(s, n+1)
Mixed Key Attack

Renew session key

Visit attacker website

Decrypt & store fragment

Login to website

Header (Frag1)

Payload (Frag2)

192.168.1.2 to 39.15.69.7

POST /login.php HTTP/1.1
user=admin&pass=SeCr3t
Mixed Key Attack

Visit attacker website

Frag1(s, n)
Frag2(s, n+1)

Renew session key

Decrypt & store fragment

Login to website

Frag1(s', n)
Frag2(s', n+1)

Reassemble & forward
Mixed Key Attack

Visit attacker website

Frag1(s, n) → Frag1(s, n)

192.168.1.2 to 39.15.69.7

POST /login.php HTTP/1.1
user=admin&pass=SeCr3t

Reassemble & forward
## Data exfiltration

<table>
<thead>
<tr>
<th>Header (Frag1)</th>
<th>Payload (Frag2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2 to 3.5.1.1</td>
<td>GET /image.png HTTP/1.1</td>
</tr>
</tbody>
</table>
# Data exfiltration

<table>
<thead>
<tr>
<th>Header (Frag1)</th>
<th>Payload (Frag2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2 to 3.5.1.1</td>
<td>GET /image.png HTTP/1.1</td>
</tr>
<tr>
<td>192.168.1.2 to 39.15.69.7</td>
<td>POST /login.php HTTP/1.1 user=admin&amp;pass=SeCr3t</td>
</tr>
</tbody>
</table>
Data exfiltration

<table>
<thead>
<tr>
<th>Header (Frag1)</th>
<th>Payload (Frag2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2 to 3.5.1.1</td>
<td>GET /image.png HTTP/1.1</td>
</tr>
</tbody>
</table>
| 192.168.1.2 to 39.15.69.7 | POST /login.php HTTP/1.1  
user=admin&pass=SeCr3t |

Adversary mixes different fragments

| 192.168.1.2 to 3.5.1.1 | POST /login.php HTTP/1.1  
user=admin&pass=SeCr3t |
Data exfiltration

<table>
<thead>
<tr>
<th>Header (Frag1)</th>
<th>Payload (Frag2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2 to 3.5.1.1</td>
<td>GET /image.png HTTP/1.1</td>
</tr>
<tr>
<td>192.168.1.2 to 39.15.69.7</td>
<td>POST /login.php HTTP/1.1 user=admin&amp;pass=SeCr3t</td>
</tr>
</tbody>
</table>

Adversary mixes different fragments

→ Login info is sent to attacker’s server
Experiments

All major operating systems are vulnerable
› Specifics may depend on driver being used
› All four tested home routers affected (Asus, Linksys, D-Link)
› One out of three professional APs affected (LANCOM)

Practically all clients are vulnerable
› Data exfiltration not possible when exploiting a client
› Possible attacks depend on target (non-trivial, see paper)
Practicality

**Non-trivial** exploitation requirements:

1. In our example AP must be vulnerable
2. Client must send fragmented frames
3. Session key is periodically refreshed
4. Client sends IP packets to attacker’s server

When combined with *implementation bugs*, this attack becomes **more practical**
Fragment Cache Attack
CVE-2020-24586
Background: fragment cache

Incomplete fragments are stored in memory = fragment cache

Frag1(s, n) → Decrypt & store fragment in cache
Background: fragment cache

Incomplete fragments are stored in memory = fragment cache

Frag1(s, n)

Decrypt & store fragment in cache

Frag2(s, n+1)

Reassemble frame & clear fragments from cache
Background: fragment cache

Incomplete fragments are stored in memory = **fragment cache**

Frag1(s, n) → Decrypt & store fragment in cache → Frag2(s, n+1) → Reassemble frame & clear fragments from cache

Problem: **fragment cache isn’t cleared** when (re)connecting to a different network
Background: fragment cache

Incomplete fragments are stored in memory = fragment cache

Frag1(s, n) → Decrypt & store fragment in cache

Frag2(s, n+1) → Reassemble frame & clear fragments from cache

→ Can be abused to exfiltrate (or inject) data
Cache Attack

Spoof victim MAC address

Connect with own credentials
Cache Attack

Spoof victim MAC address

Connect with own credentials

Target is an enterprise (hotspot) network

› Users don’t trust each other

› Adversary has valid credentials
Cache Attack

Spoof victim MAC address

Connect with own credentials

Frag1(s, n)
Cache Attack

- Spoof victim MAC address
- Connect with own credentials
- Frag1(s, n)
- Store fragment under victim MAC address
Cache Attack

- Spoof victim MAC address
- Connect with own credentials
- Frag1(s, n)
- Store fragment under victim MAC address

<table>
<thead>
<tr>
<th>Header (Frag1)</th>
<th>Payload (Frag2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2 to 3.5.1.1</td>
<td></td>
</tr>
</tbody>
</table>
Cache Attack

Spoof victim MAC address

Connect with own credentials

Frag1(s, n)

Store fragment under victim MAC address
Cache Attack

Spoof victim MAC address

Connect with own credentials

Frag1(s, n)

Store fragment under victim MAC address

Disconnect

Vulnerability: AP won’t clear fragment from its memory
Cache Attack

- Spoof victim MAC address
- Connect with own credentials
- Frag1(s, n)
- Store fragment under victim MAC address
- Disconnect
- Connect with real credentials
Cache Attack

Store fragment under victim MAC address

Disconnect

Connect with real credentials
Cache Attack

Login to website

Connect with real credentials

Disconnect

Store fragment under victim MAC address
Cache Attack

Store fragment under victim MAC address

Connect with real credentials

Disconnect

Login to website

Frag1(s', n)

Frag2(s', n+1)

Frag2(s, n+1)
<table>
<thead>
<tr>
<th>Header (Frag1)</th>
<th>Payload (Frag2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>POST /login.php HTTP/1.1 user=admin&amp;pass=SeCr3t</td>
</tr>
</tbody>
</table>

Login to website

<table>
<thead>
<tr>
<th>Frag1(s’, n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frag2(s’, n+1)</td>
</tr>
<tr>
<td>Frag2(s, n+1)</td>
</tr>
</tbody>
</table>
Cache Attack

Store fragment under victim MAC address

Connect with real credentials

Disconnect

Login to website

Frag1(s’, n)

Frag2(s’, n+1)

Frag2(s, n+1)

Reassemble and process frame
### Cache Attack

<table>
<thead>
<tr>
<th>Header (Frag1)</th>
<th>Payload (Frag2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2 to 3.5.1.1</td>
<td>POST /login.php HTTP/1.1 user=admin&amp;pass=SeCr3t</td>
</tr>
</tbody>
</table>

**Diagram:**

- Login to website
- Frag1(s', n) → Frag2(s', n+1)
- Frag2(s, n+1) → Reassemble and process frame

**Packet:**

```
192.168.1.2 to 3.5.1.1
Login to website
Frag1(s', n)
Frag2(s', n+1)
Frag2(s, n+1)
Reassemble and process frame
```

```
POST /login.php HTTP/1.1
user=admin&pass=SeCr3t
```
Cache Attack

Header (Frag1) | Payload (Frag2)
---|---
192.168.1.2 to 3.5.1.1 | POST /login.php HTTP/1.1 user=admin&pass=SeCr3t

Login to website

Frag1(s’, n) → Frag2(s’, n+1) → Frag2(s, n+1)

Reassemble and process frame

→ Login data in Frag2 is now sent to the attacker
Experiments

Roughly **half of all tested devices** are vulnerable
› Seems to depend on driver & network card
› My four home routes were all affected
› None of the three professional APs were affected

Also possible to exploit clients under non-trivial threat model
› Can inject packets towards the client
› See paper for details
Recap: attacker can exfiltrate/inject packets

Fragments aren’t cleared from memory after (re)connecting
   › Adversary can poison the cache

Threat model and impact:
   › Exploiting an AP: against enterprise network, client sends fragmented frames, can **exfiltrate client data**
   › Exploiting a client: non-trivial threat model, AP send fragmented frames, can **inject packets towards the client**
Implementation Flaws: trivial plaintext injection
Accepted plaintext frames (CVE-2020-26140 / 26143)

Accepting plaintext frames (CVE-2020-26140)
› Examples: some routers, some dongles on Linux/Windows

Accepting fragmented plaintext frames (CVE-2020-26143)
› Examples: many dongles on Windows, some FreeBSD APs

→ Can inject frames independent of network config
Plaintext broadcast fragments (CVE-2020-26145)

Some devices accept plaintext broadcast fragments

› Sometimes only accepted while connecting
› Treated as full frames!
› Examples: MacOS, iOS, and Free/NetBSD APs

→ Can inject frames independent of network config
Cloacked aggregated frames (CVE-2020-26144)

Implementations must accept plaintext EAPOL frames

› Abuse to inject plaintext aggregated frames to low #devices:

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>metadata</td>
<td>length</td>
</tr>
<tr>
<td></td>
<td>packet1</td>
<td>metadata</td>
</tr>
<tr>
<td></td>
<td>length</td>
<td>packet2</td>
</tr>
</tbody>
</table>
 Cloacked aggregated frames (CVE-2020-26144)

Implementations must accept plaintext EAPOL frames

› Abuse to inject plaintext aggregated frames to low #devices:

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>EAPOL hdr</td>
<td>length</td>
</tr>
<tr>
<td></td>
<td>ignored</td>
<td>metadata</td>
</tr>
<tr>
<td></td>
<td>length</td>
<td>packet2</td>
</tr>
</tbody>
</table>

› Set metadata to start of EAPOL ⇒ plaintext frame is accepted
**Cloaked aggregated frames (CVE-2020-26144)**

Implementations must accept plaintext EAPOL frames

› Abuse to inject plaintext aggregated frames to low #devices:

<table>
<thead>
<tr>
<th>header</th>
<th>aggregated?</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td><strong>EAPOL hdr</strong></td>
<td>length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ignored</td>
</tr>
<tr>
<td></td>
<td></td>
<td>metadata</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>packet2</td>
</tr>
</tbody>
</table>

› Set metadata to start of EAPOL → plaintext frame is accepted

› 1<sup>st</sup> sub-packet will be ignored, but 2<sup>nd</sup> one is processed

→ Can trivially inject plaintext frames
Implementation flaws with other impact
Pre-auth EAPOL forwarding (CVE-2020-26139)

Some APs forwards EAPOL frames before sender is authenticated

Examples: Net/FreeBSD APs and $\frac{2}{4}$ home routers

→ Abuse to *inject frames* in combination with aggregation attack (CVE-2020-24588)
Non-consecutive packet numbers (CVE-2020-26146)

<table>
<thead>
<tr>
<th>header</th>
<th>$s$</th>
<th>$n$</th>
<th>0</th>
<th>More</th>
<th>fragment1</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>$s$</td>
<td>$n + 1$</td>
<td>1</td>
<td>More</td>
<td>fragment2</td>
</tr>
<tr>
<td>header</td>
<td>$s$</td>
<td>$n + 2$</td>
<td>2</td>
<td>Last</td>
<td>fragment3</td>
</tr>
</tbody>
</table>

- Recap: fragments must have consecutive packet numbers
- **Almost nobody checks this!** Only Linux does.
- Can do mixed key “exfiltration” attack without periodic rekeys
  - Client must still use fragmentation & connect to attacker server
Mixed plain/encrypted fragments (CVE-2020-26147)

Many devices only require the first fragment to be encrypted
- Most Windows & Linux drivers, some Free/NetBSD drivers
- Aggregation attack possible without the victim needing to connect to the attacker’s website (can inject packets)
- Cache attack against client possible when even when the AP doesn’t send fragmented frames (can inject packets)

Some devices only require the last fragment to be encrypted
- Several Free/NetBSD drivers
- Trivial to inject data if fragmentation is used
No fragmentation support (CVE-2020-26142)

Some devices don’t support fragmentation

› They treat fragmented frames as full frames
› Examples: OpenBSD and ESP12-F

Abuse to inject frames when:

› Another device sends fragmented frames
› This other device visits the attacker’s server
Discussion
Practicality vs. impact

Perhaps we’re lucky:
› Widespread flaws $\rightarrow$ relatively trickly to exploit in practice
› Trivial to exploit flaws $\rightarrow$ not widespread in practice (?)

Important concerns remain:
› Significant #devices affected by trivial to exploit flaws
› Every Wi-Fi device affected by one or more flaws
› Combining flaws increases practicality of certain attacks

$\rightarrow$ Patch now before attack improve!
Conclusion

› Aggregation & fragmentation design flaws
› Several common implementation flaws
› Need patched drivers to test if affected
› Impact varies per device & network