Securely Implementing Network Protocols: Detecting and Preventing Logical Flaws

Mathy Vanhoef (KU Leuven)

Black Hat Webcast, 24 August 2017

@vanhoefm

DistriNet

Introduction

Many protocols have been affected by **logical** bugs

	Design flaws	Implementation flaws
TLS	BEAST[11] POODLE[12] Lucky 13[13] …	Early CCS attack[5] FREAK[8] Logjam[10] …
Wi-Fi	WEP Protected setup[7] Key reinstallations[1] …	Bad state machine[4] No downgrade check[4] Bad randomness[6,7] …
SSH	CBC plaintext recovery[2]	Bad state machine[3]

Introduction

Many protocols have been affected by **logical** bugs

We focus on logical **implementation flaws**

Implementation flaws
Early CCS attack[5] FREAK[8] Logjam[10] …
Bad state machine[4] No downgrade check[4] Bad randomness[6,7] …
Bad state machine[3]

How were TLS flaws detected?

Several works audited state machines:

2014
- Kikuchi discovered the early CCS attack[5]
- Manual inspection of CCS transitions in implementations

2015
- Beurdouche et al: manually define state machine of TLS[8]
- Use state machine to generate invalid handshakes

2016
- de Ruiter and Poll: extract state machine automatically[9]
- Manually inspect state machine for anomalies

Lesson: use model-based testing!

→ Model-based testing!

- Test if program behaves according to some abstract model
- Proved successful against TLS

- We applied model-based approach on the Wi-Fi handshake
- Our technique can be used to test other protocols!

Background: the Wi-Fi handshake

Main purposes:

- Network discovery
- Mutual authentication & negotiation of pairwise session key
- Securely select cipher to encrypt data frames

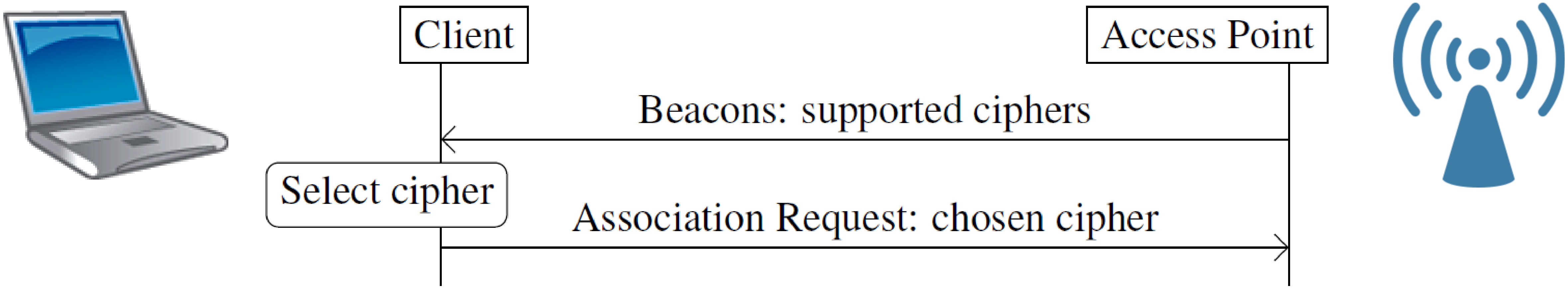
WPA-TKIP

Short-term solution: reduced security so it could run on old hardware

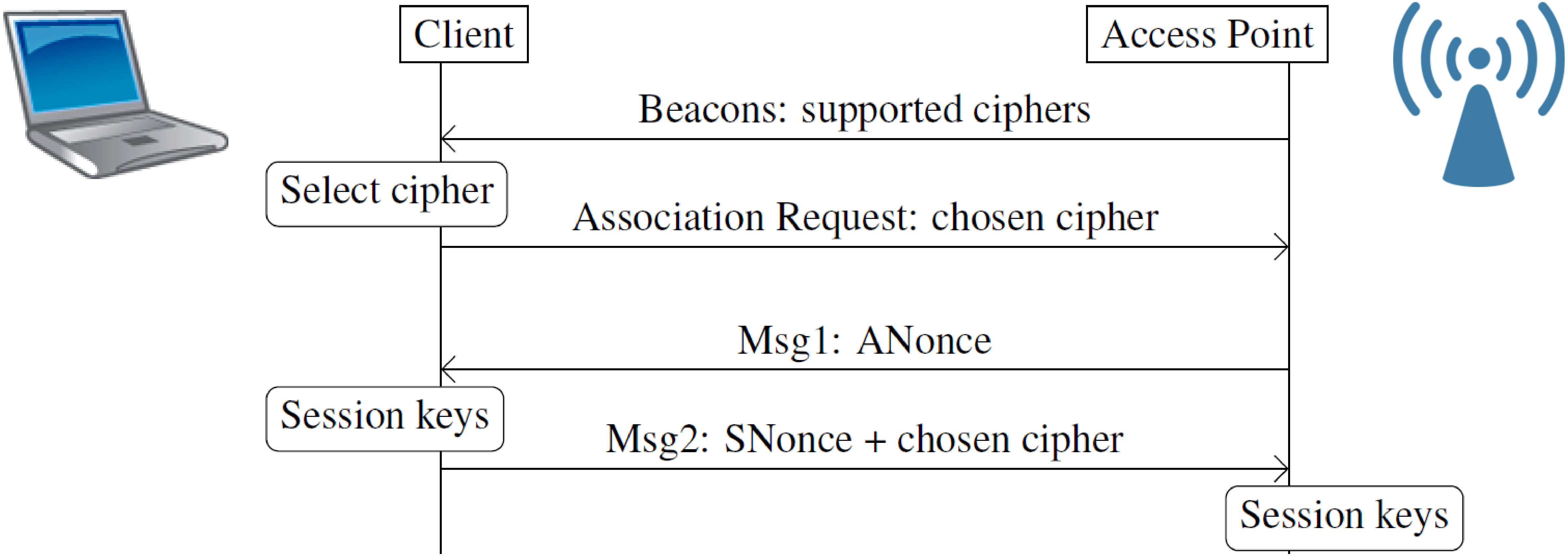
AES-CCMP

Long-term solution based on modern cryptographic primitives

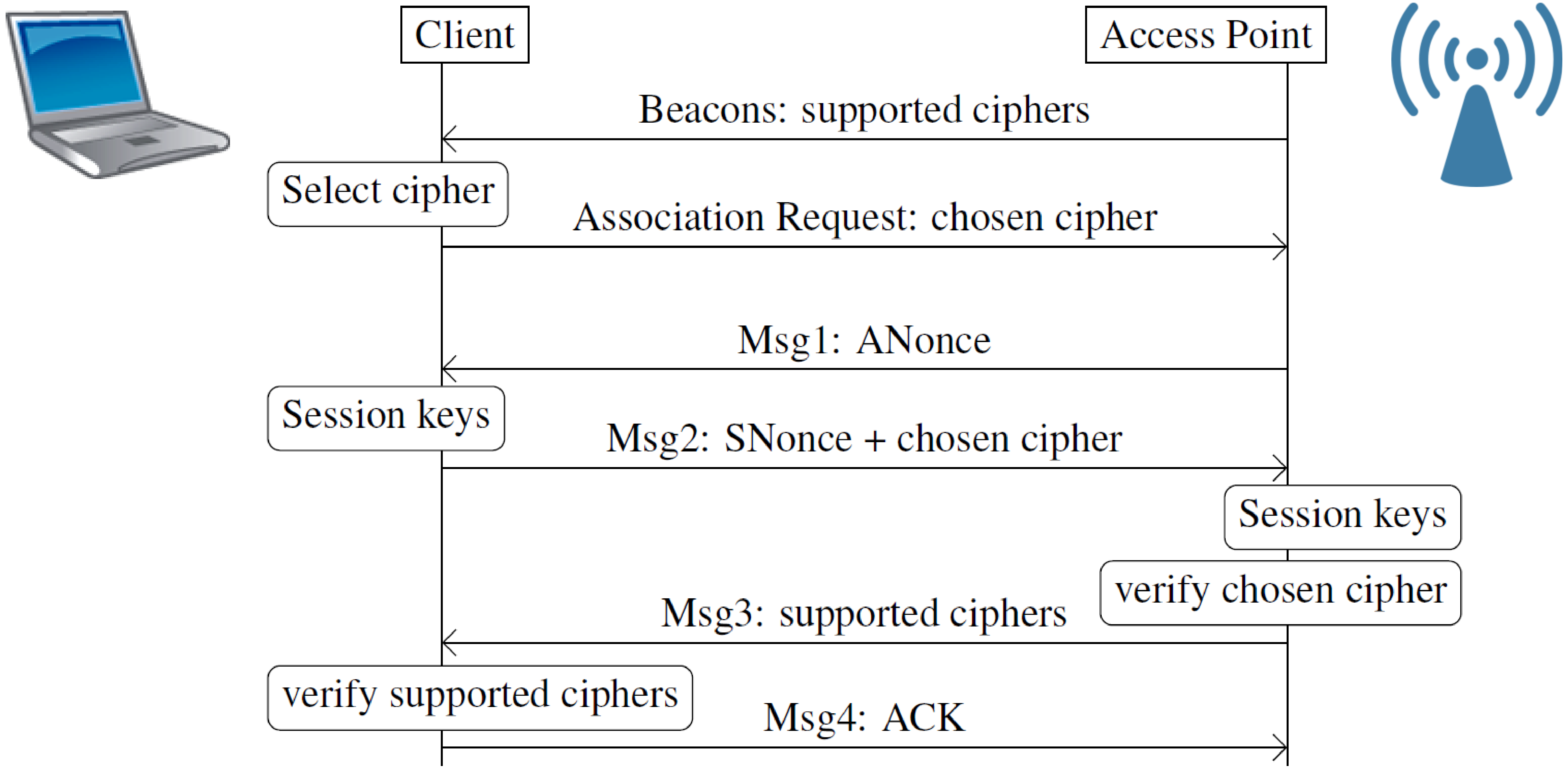
Wi-Fi handshake (simplified)



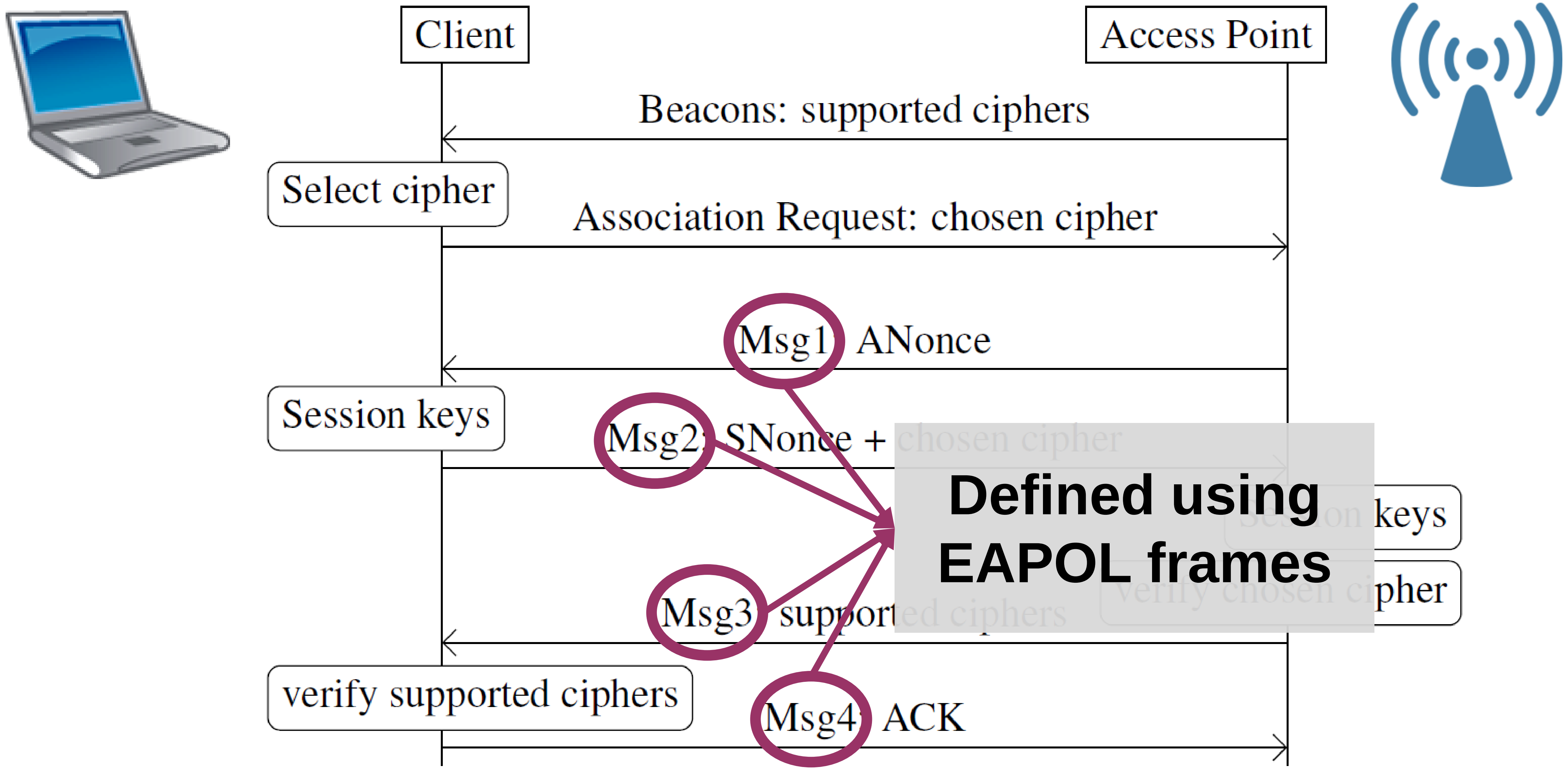
Wi-Fi handshake (simplified)



Wi-Fi handshake (simplified)



Wi-Fi handshake (simplified)

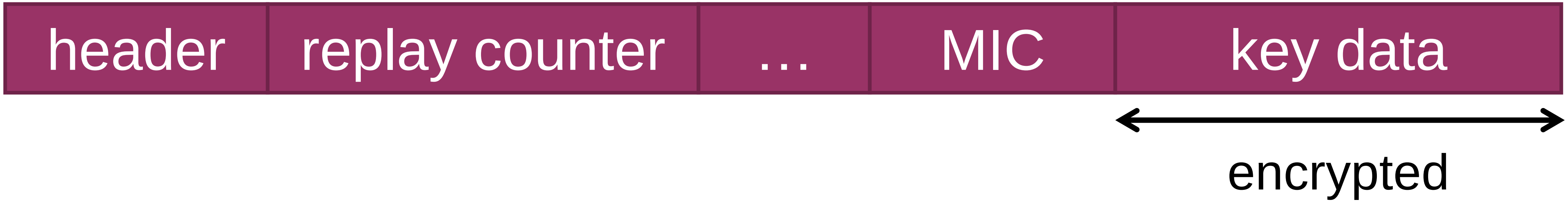


EAPOL frame layout

```
⌄ 802.1X Authentication
    Version: 802.1X-2004 (2)
    Type: Key (3)
    Length: 117
    Key Descriptor Type: EAPOL RSN Key (2)
  › Key Information: 0x008a
    Key Length: 16
    Replay Counter: 0
    WPA Key Nonce: 3e8e967dacd960324cac5b6aa721235bf57b949771c86798...
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 00000000000000000000000000000000
    WPA Key Data Length: 22
  › WPA Key Data: dd14000fac04592da88096c461da246c69001e877f3d
```

EAPOL frame layout

≈



Model-based testing: our approach

Model: normal handshake $\xrightarrow{\text{Test generation rules: (in)correct modifications}}$ Set of test cases

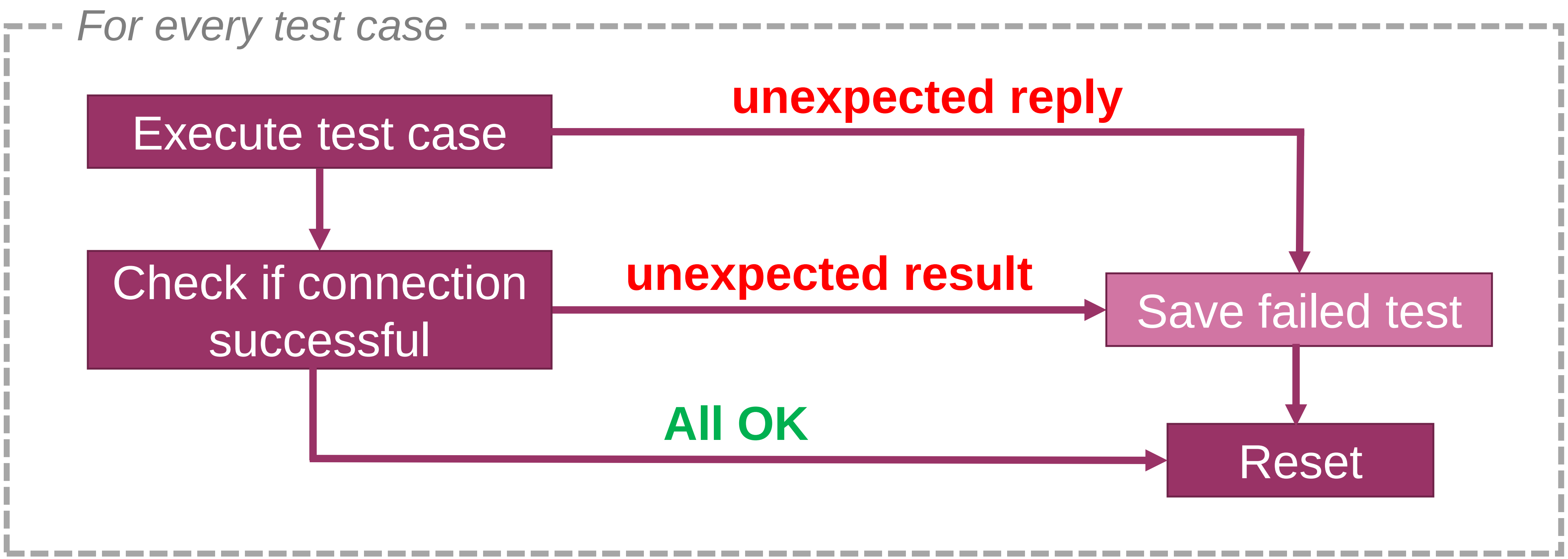
Test generation rules:

- Test various edge cases, allows some creativity
- Are assumed to be independent (avoid state explosion)

A test case defines:

1. Messages to send & expected replies
2. Results in successful connection?

Executing test cases



Afterwards inspect <u>failed</u> test cases

- Experts determines impact and exploitability

Test generation rules

Test generation rules manipulating messages as a whole:

1. Drop a message
2. Inject/repeat a message

Test generation rules that modify fields in messages:

1. Bad EAPOL replay counter
2. Bad EAPOL header (e.g. message ID)
3. Bad EAPOL Message Integrity Check (MIC)
4. Mismatch in selected cipher suite
5. …

Evaluation

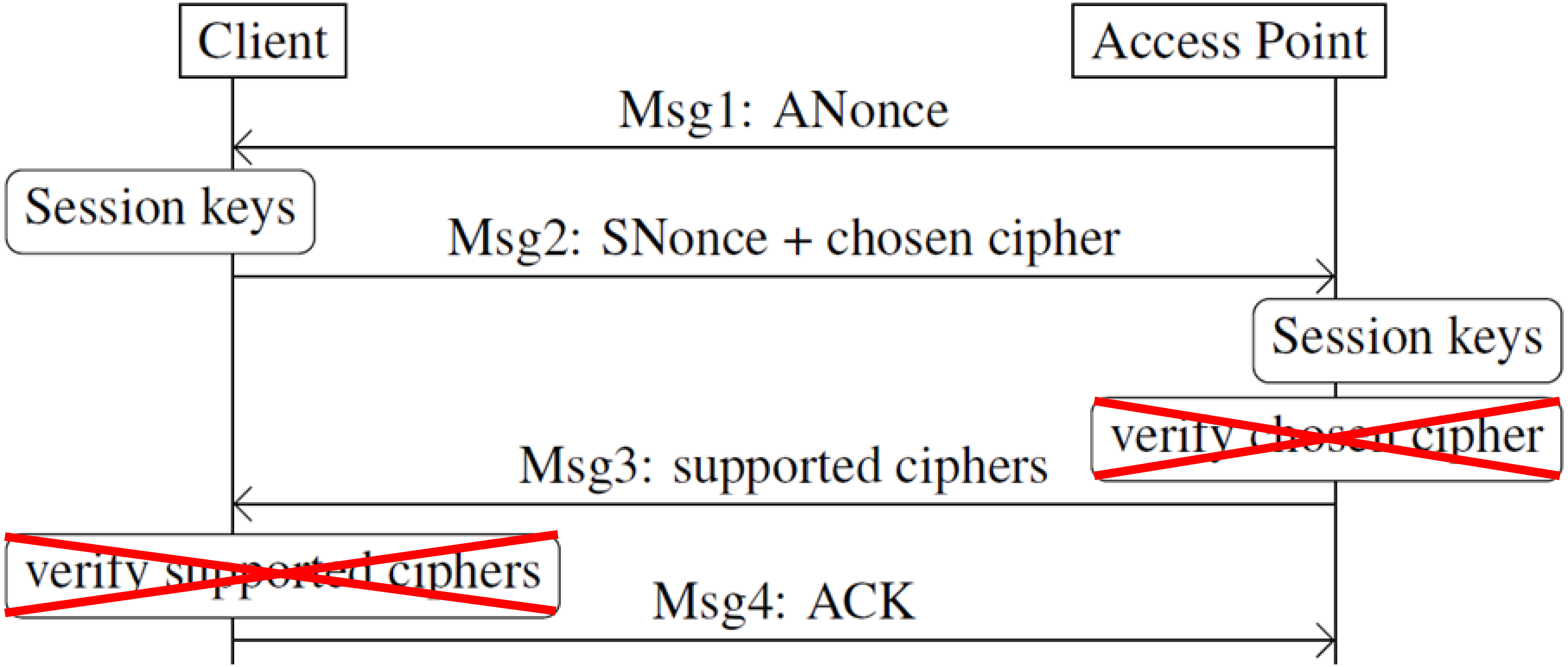
We tested 12 access points:

- Open source: OpenBSD, Linux’s Hostapd
- Leaked source: Broadcom, MediaTek (home routers)
- Closed source: Windows, Apple, …
- Professional equipment: Aerohive, Aironet

Discovered several issues!

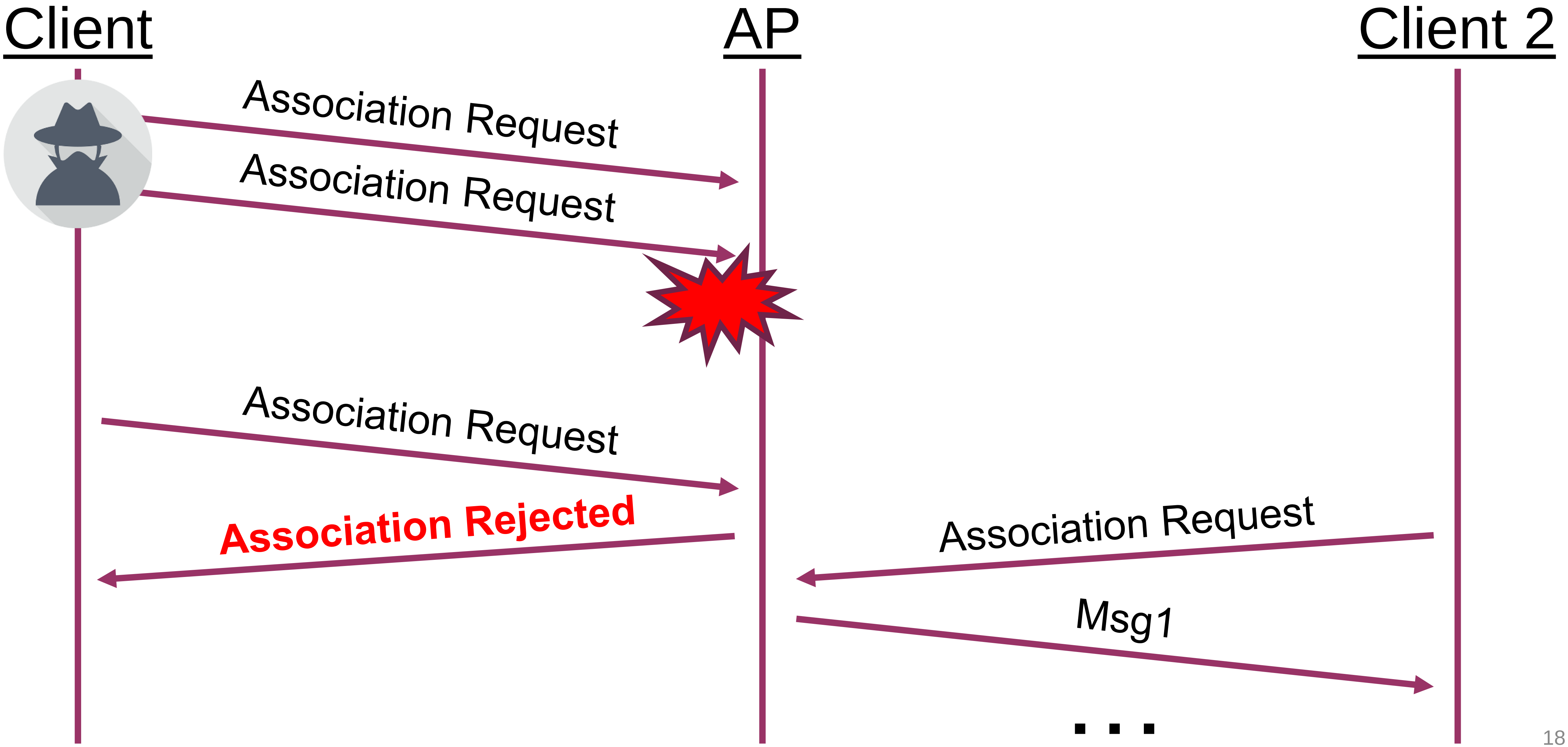
Missing downgrade checks

1. MediaTek & Telenet don't verify selected cipher in message 2
2. MediaTek also ignores supported ciphers in message 3

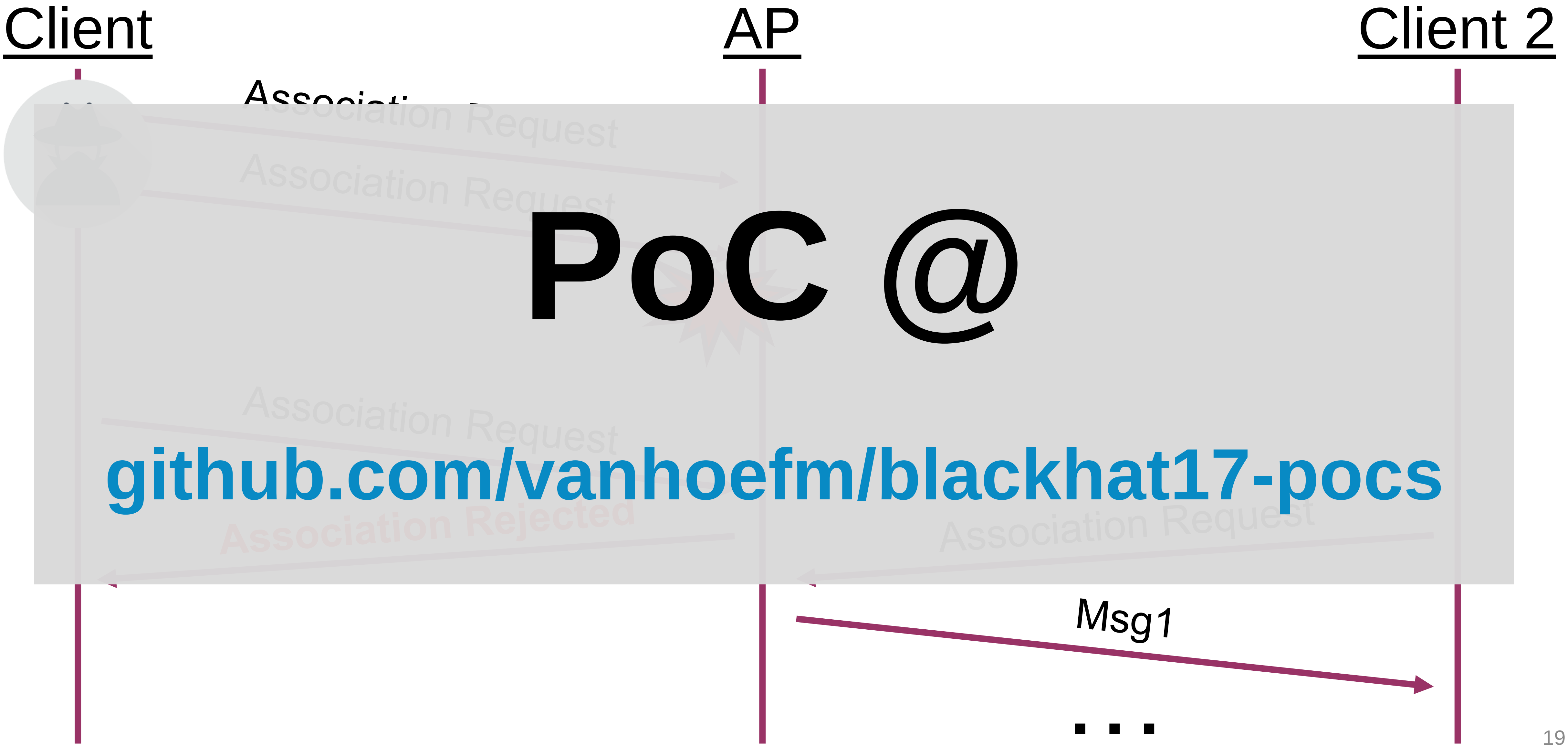


→ Trivial downgrade attack against MediaTek clients

Windows 7 targeted DoS



Windows 7 targeted DoS



PoC @

github.com/vanhoefm/blackhat17-pocs

Broadcom downgrade

Broadcom cannot distinguish message 2 and 4

- Can be abused to downgrade the AP to TKIP

Hence message 4 is essential in preventing downgrade attacks

- This highlights incorrect claims in the 802.11 standard:

> "**While Message 4 serves no cryptographic purpose**, it serves as an acknowledgment to Message 3. **It is required to ensure reliability** and to inform the Authenticator that the Supplicant has installed the PTK and GTK and hence can receive encrypted frames."

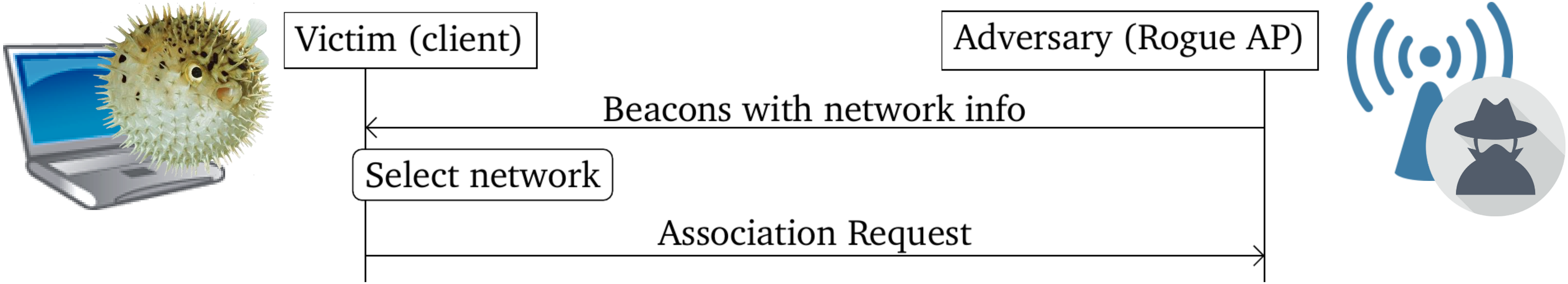
OpenBSD: client man-in-the-middle

Bug in state machine of AP → we also inspected client:

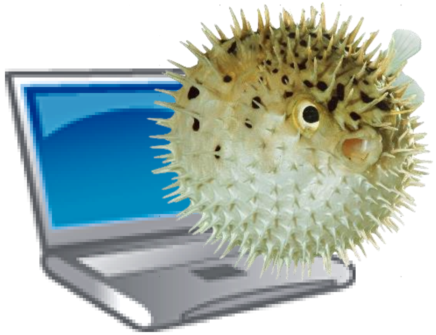
State machine missing!

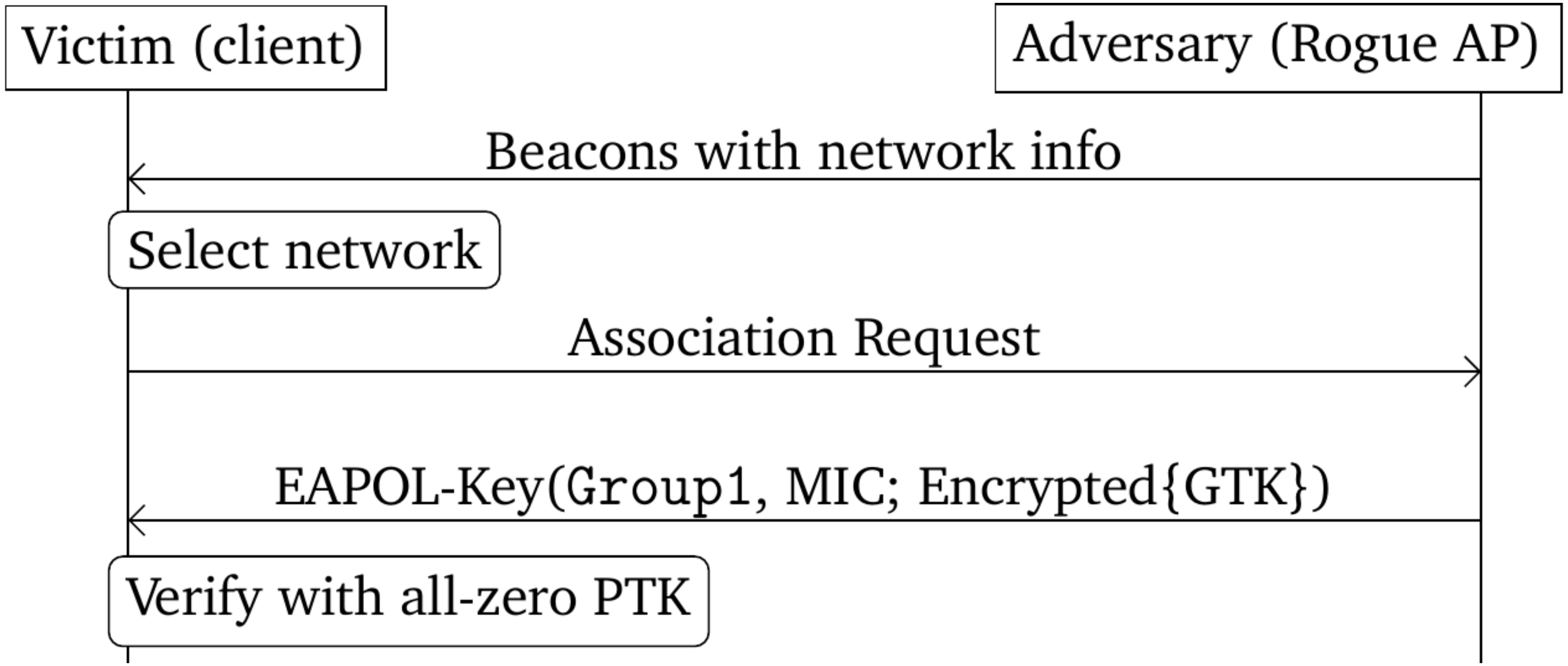
→ Man-in-the-middle against client

OpenBSD: client man-in-the-middle

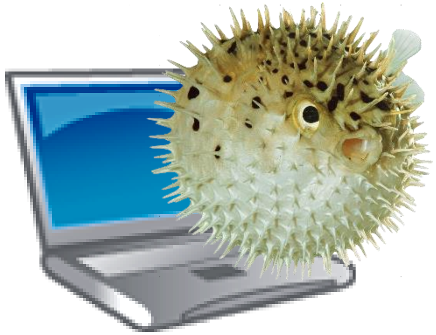


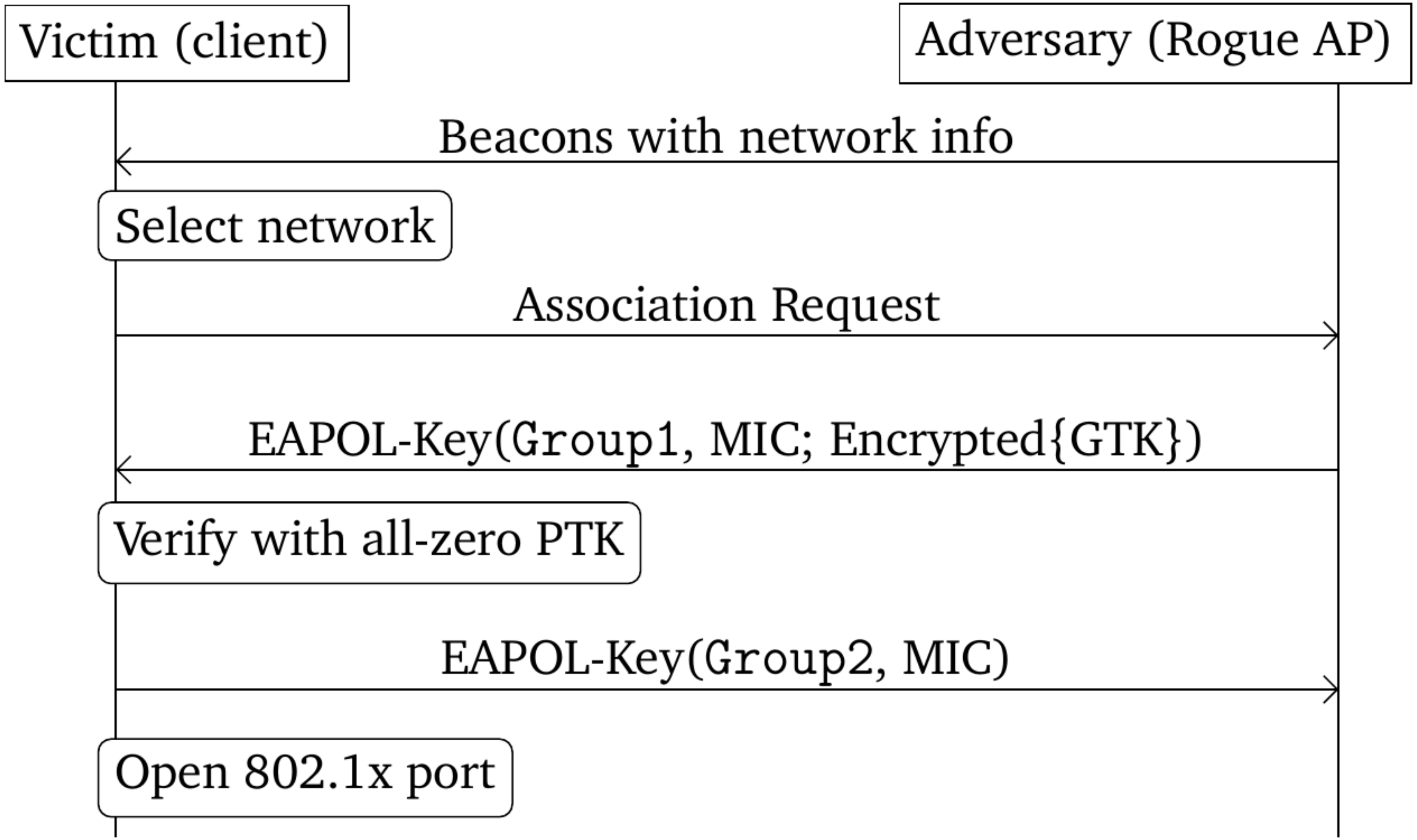
OpenBSD: client man-in-the-middle



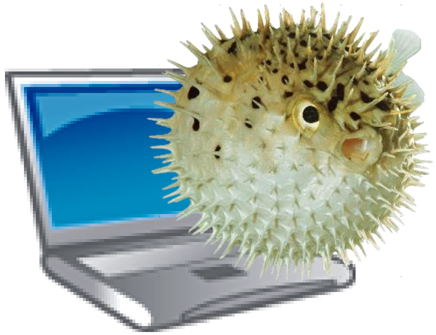


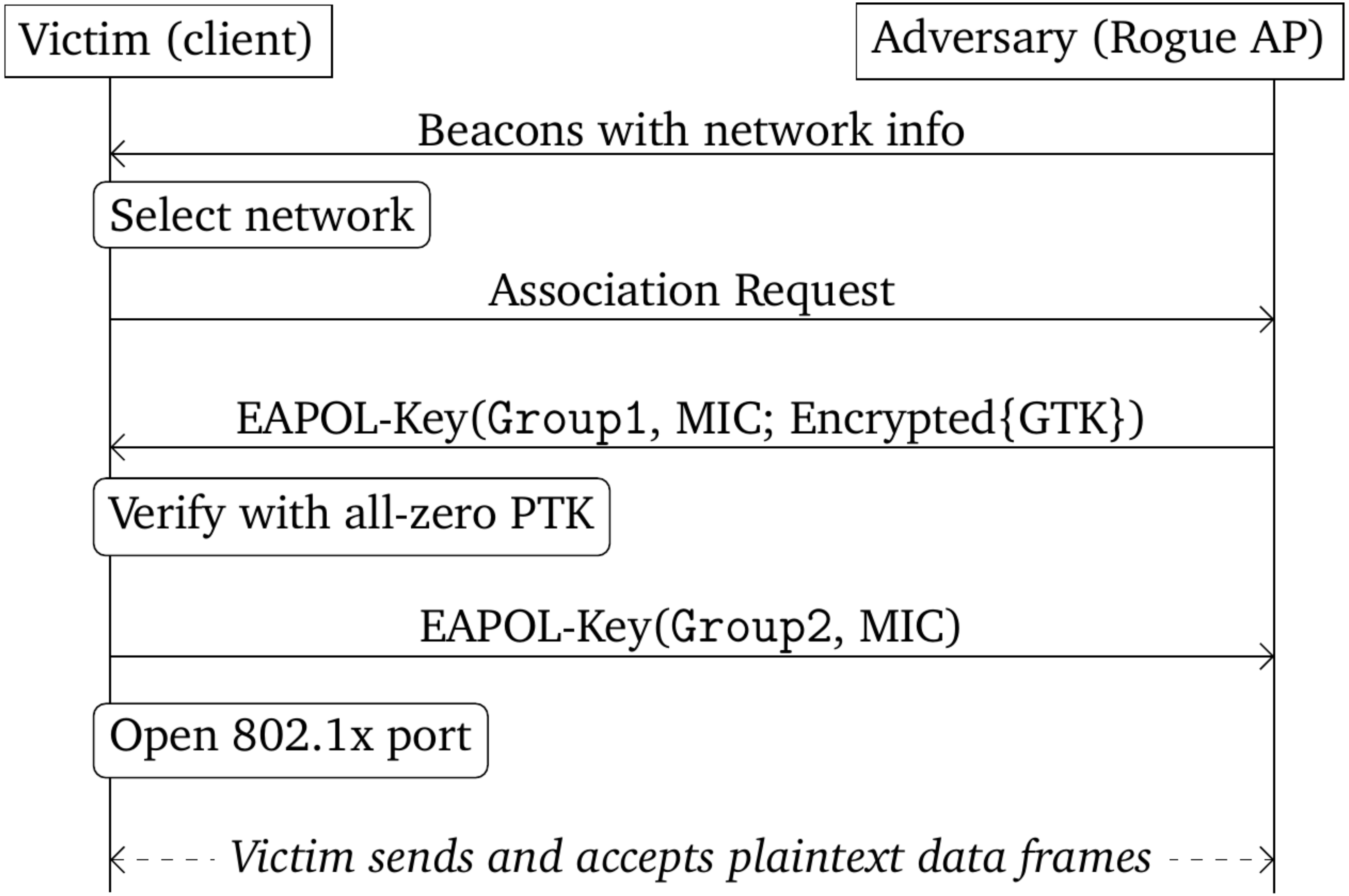
OpenBSD: client man-in-the-middle



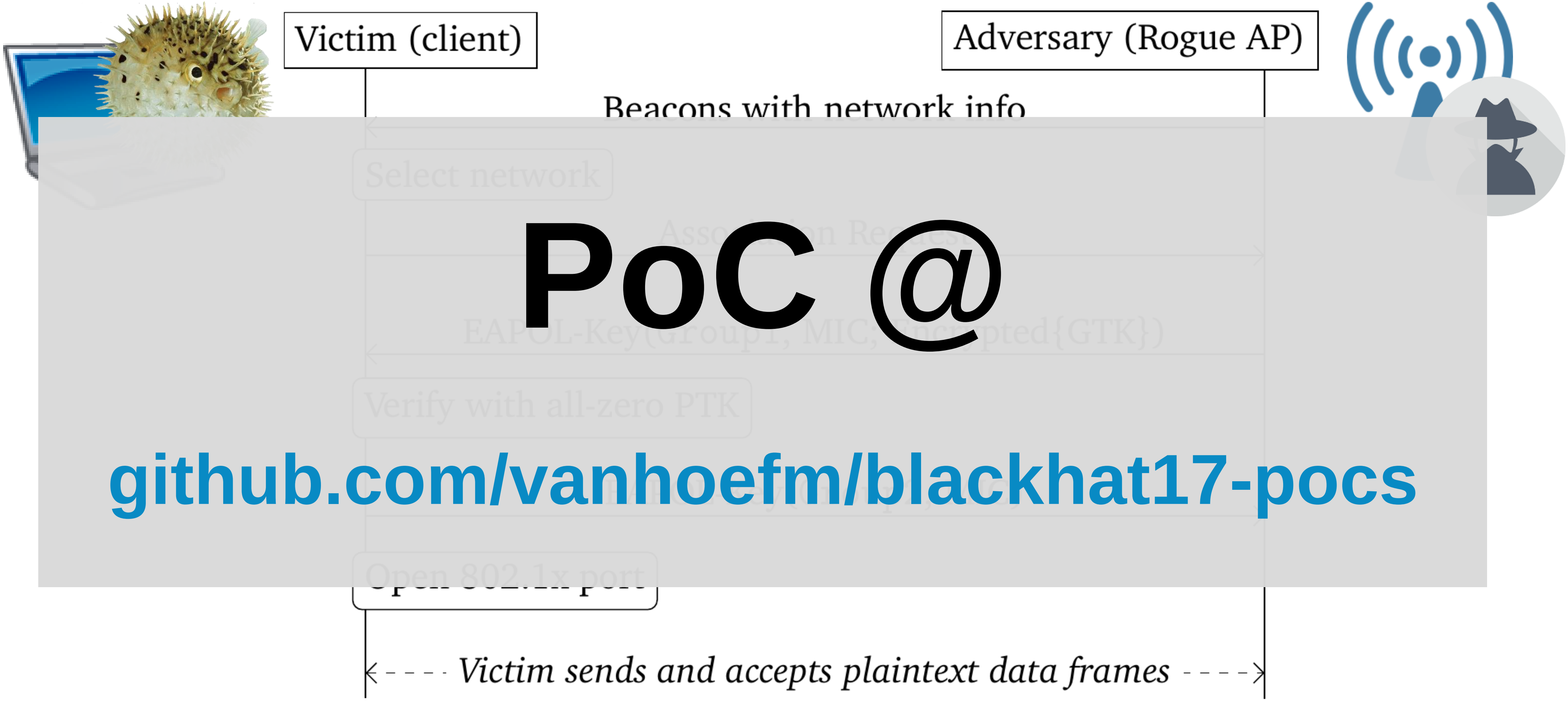


OpenBSD: client man-in-the-middle





OpenBSD: client man-in-the-middle



PoC @

github.com/vanhoefm/blackhat17-pocs

More results

See Black Hat & AsiaCCS paper[4]:

- Benign irregularities → fingerprint
- Permanent DoS attack against Broadcom and OpenBSD
- DoS attack against Windows 10, Broadcom, Aerohive
- Inconsistent parsing of supported cipher suite list
- …

Future work!

Current limitations:

- Amount of code coverage is unknown
- Only used well-formed (albeit invalid) packets
- Test generation rules applied independently

But already a promising technique

- ✓ Black-box testing mechanism: no source code needed
- ✓ Fairly simple handshake, but still several logical bugs!

Conclusion: avoiding logical bugs

What helps:

- Try to generalize known bugs (in your/other products)
- Model-based testing (e.g. this presentation)
- Write rigorous requirements (specification) and review them
- Detailed code reviews (e.g. by domain experts)

Does not help:

- Standard code review (only detects common mistakes)
- Traditional static or dynamic testing

Securely Implementing Network Protocols: Detecting and Preventing Logical Flaws

Mathy Vanhoef (KU Leuven)

Black Hat Webcast, 24 August 2017

@vanhoefm

References

1. M. Vanhoef and F. Piessens. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In CCS, 2017.
2. M. R. Albrecht, K. G. Paterson and G. J. Watson. Plaintext Recovery Attacks Against SSH. In IEEE S&P, 2009.
3. E. Poll and A. Schubert. Verifying an implementation of SSH. In WITS, 2007.
4. M. Vanhoef, D. Schepers, and F. Piessens. Discovering Logical Vulnerabilities in the Wi-Fi Handshake Using Model-Based Testing. In ASIA CCS, 2017.
5. M. Kikuchi. How I discovered CCS Injection Vulnerability (CVE-2014-0224). Retrieved 20 August 2017 from http://ccsinjection.lepidum.co.jp/blog/2014-06-05/CCS-Injection-en/index.html
6. M. Vanhoef and F. Piessens. Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys. In USENIX Security, 2016.
7. D. Bongard. Offline bruteforce attack on WiFi Protected Setup. In Hack Lu, 2014.
8. Beurdouche et al. A Messy State of the Union: Taming the Composite State Machines of TLS. In IEEE S&P, 2015.
9. J. de Ruiter and E. Poll. Protocol State Fuzzing of TLS Implementations. In USENIX Security, 2015.
10. D. Adrian et al. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In CCS, 2015.
11. T. Duong and J. Rizzo. Here come the xor ninjas. In Ekoparty Security Conference, 2011.
12. B. Möller, T. Duong, and K. Kotowicz. This POODLE Bites: Exploiting The SSL 3.0 Fallback. In Google Security Blog, 2014.
13. N. J. Al Fardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In IEEE S&P, 2013.