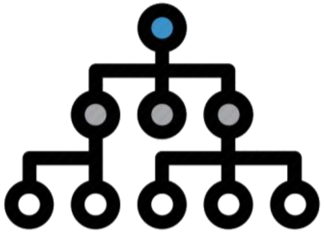
Rooting Routers Using Symbolic Execution

Mathy Vanhoef — @vanhoefm

OPCDE, Dubai, 20 April 2019

Overview



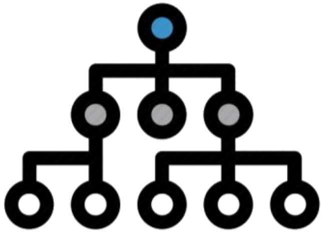
Symbolic Execution

4-way handshake

Handling Crypto

Results

Overview



Symbolic Execution

4-way handshake

Handling Crypto

Results

Symbolic Execution

Mark data as symbolic

```
void recv(data, len) {
  if (data[0] != 1)
    return
  if (data[1] != len)
    return

  int num = len/data[2]
  ...
}
```

Symbolic branch

Symbolic Execution

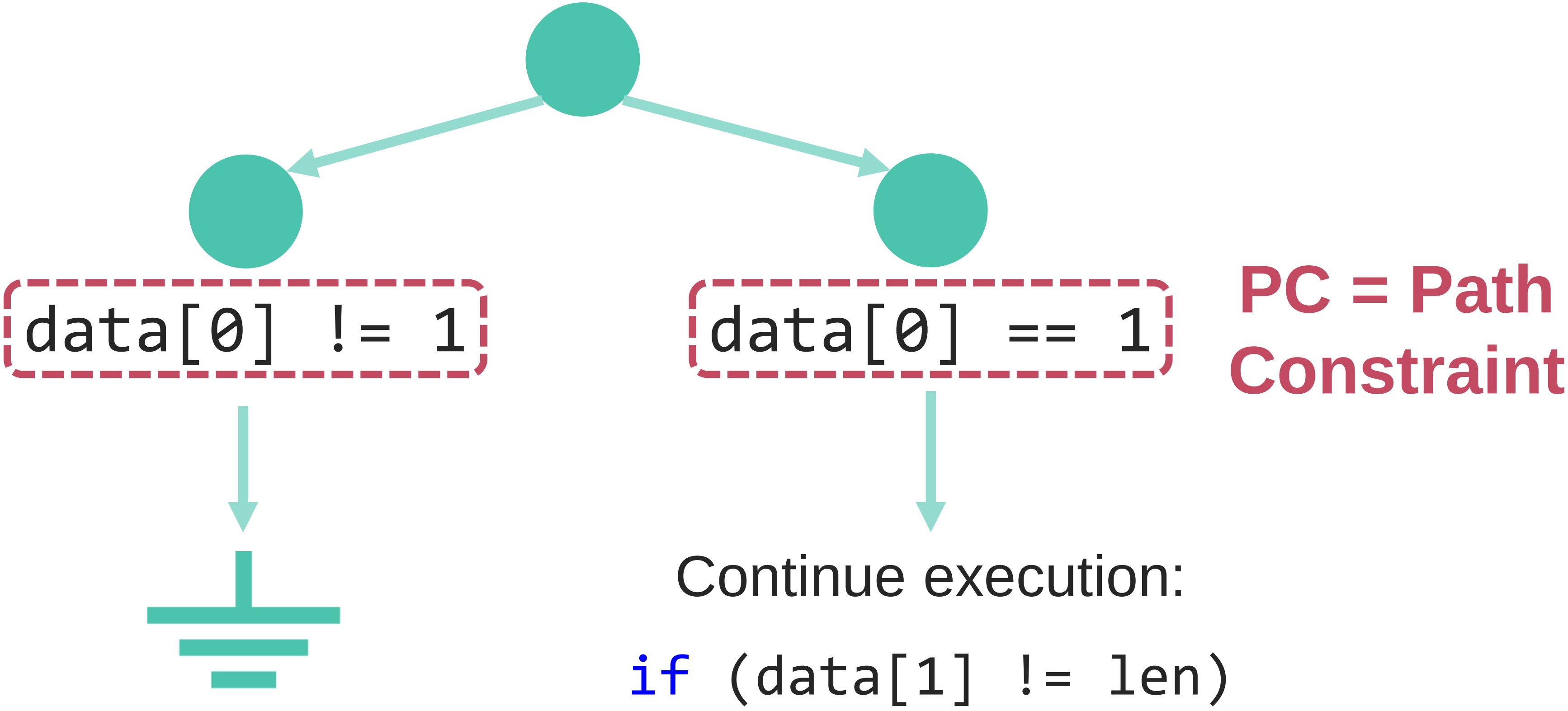
data[0] != 1

```
void recv(data, len) {
  if (data[0] != 1)
    return
  if (data[1] != len)
    return

  int num = len/data[2]
  ...
}
```

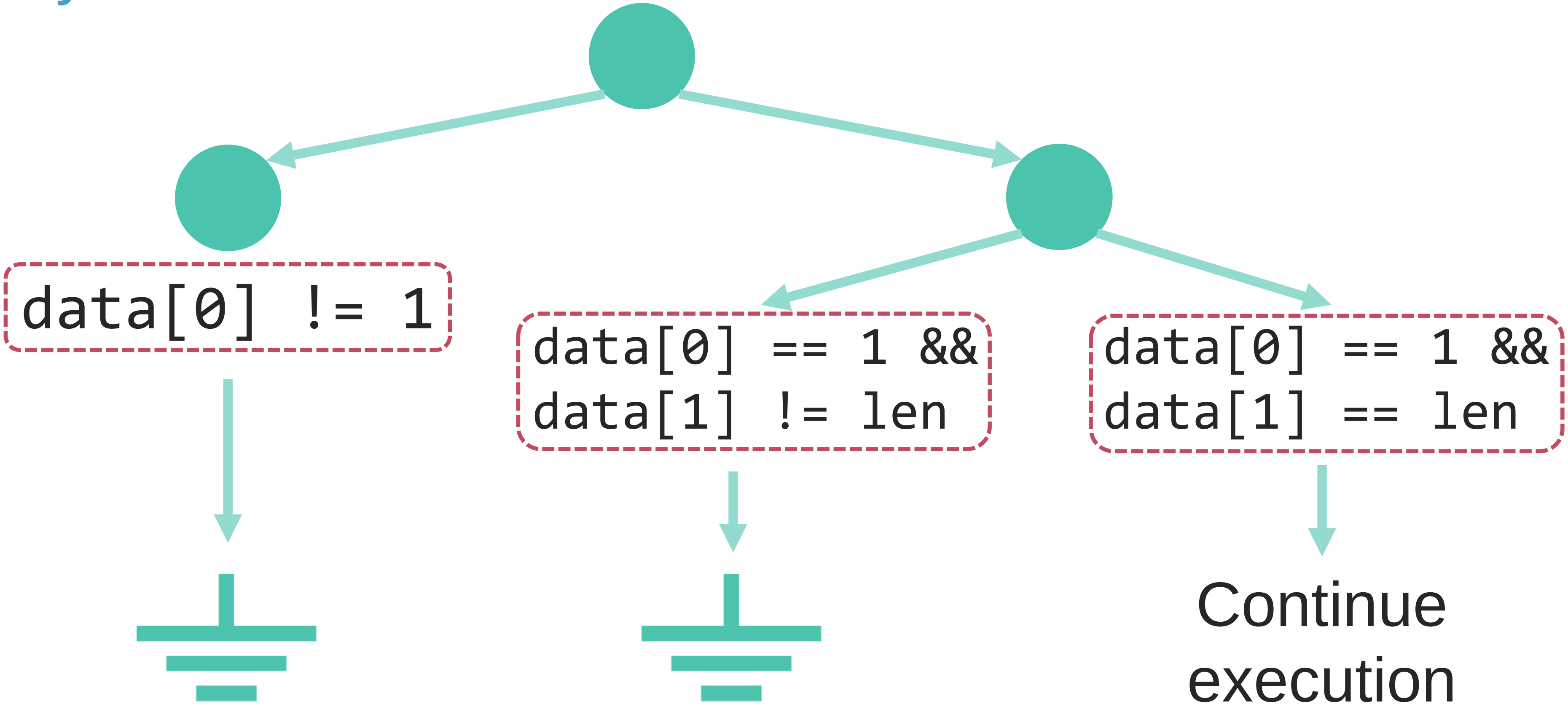
data[0] == 1

```
void recv(data, len) {
  if (data[0] != 1)
    return
  if (data[1] != len)
    return

  int num = len/data[2]
  ...
}
```

Symbolic Execution



Symbolic Execution



Symbolic Execution

```
data[0] == 1 &&
data[1] == len
-----------------------------
void recv(data, len) {
  if (data[0] != 1)
    return
  if (data[1] != len)
    return

  int num = len/data[2]
  ...
```

Can data[2] equal zero under the current PC?

Yes! Bug detected!

Implementations

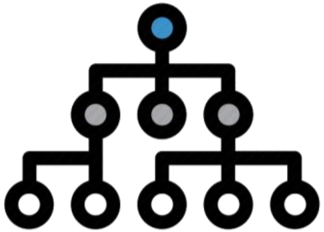
We build upon KLEE

- Works on LLVM bytecode
- Actively maintained

Practical limitations:

- $|paths| = 2^{|if-statements|}$
- Infinite-length paths
- SMT query complexity

Overview



Symbolic Execution

4-way handshake

Handling Crypto

Results

Motivating Example

Mark data as symbolic

```
void recv(data, len) {
  plain = decrypt(data, len)
  if (plain == NULL) return

  if (plain[0] == COMMAND)
    process_command(plain)
  else
    ...
}
```

Summarize crypto algo. (time consuming)

Analyze crypto algo. (time consuming)

Won't reach this function!

Efficiently handling decryption?

Comment out crypto code?

≈

Create fresh variables

Example

Mark data as symbolic

```
void recv(data, len) {
  plain = decrypt(data, len)
  if (plain == NULL) return

  if (plain[0] == COMMAND)
    process_command(plain)
  else
    ...
}
```

Create fresh symbolic variable

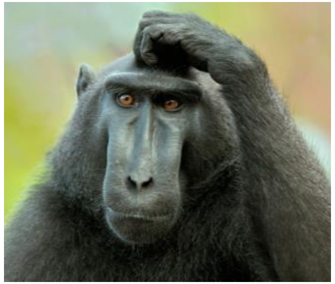
Normal analysis

→ Can now **analyze code that parses decrypted data**

Other than handling decryption

Handle hash functions

› Output = fresh symbolic variable



Track use of crypto primitives

› Save relationship between input & output

Detecting Crypto Misuse

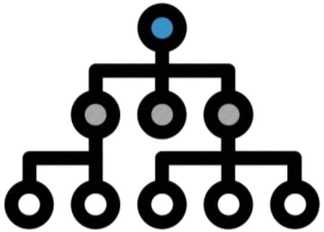
Timing side-channels

- All bytes of MAC in path constraint?
- If not: exits on first byte difference

Decryption oracles

- Behavior depends on unauth. decrypted data
- Decrypt data is in path constraint, but not in MAC

Overview



Symbolic Execution

4-way handshake

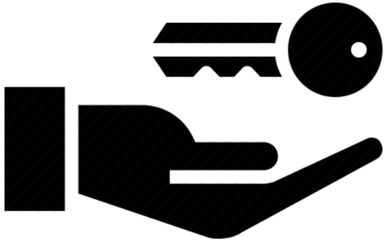
Handling Crypto

Results

The 4-way handshake

Used to connect to any protected Wi-Fi network

Mutual authentication



Negotiates fresh PTK: pairwise transient key

Connection process in WPA3

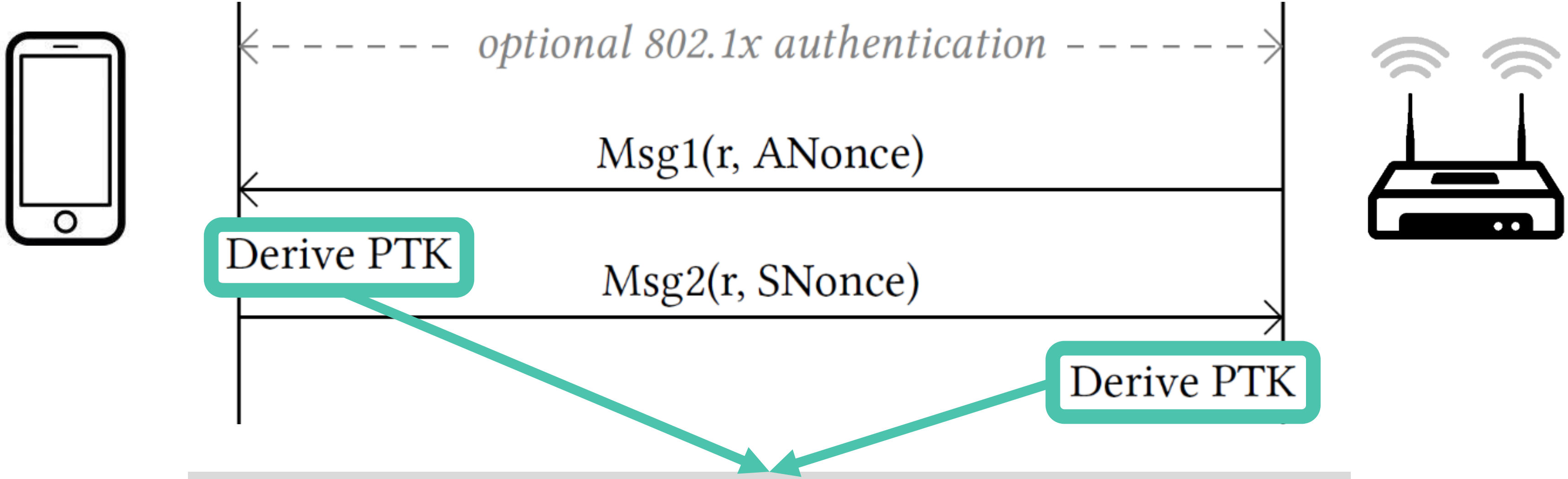
1. Dragonfly handshake negotiates high-entropy key
2. This key is subsequently used in 4-way handshake

So the **4-way handshake is still used**! We found:

- Denial-of-service
- Buffer overflows
- Decryption oracles

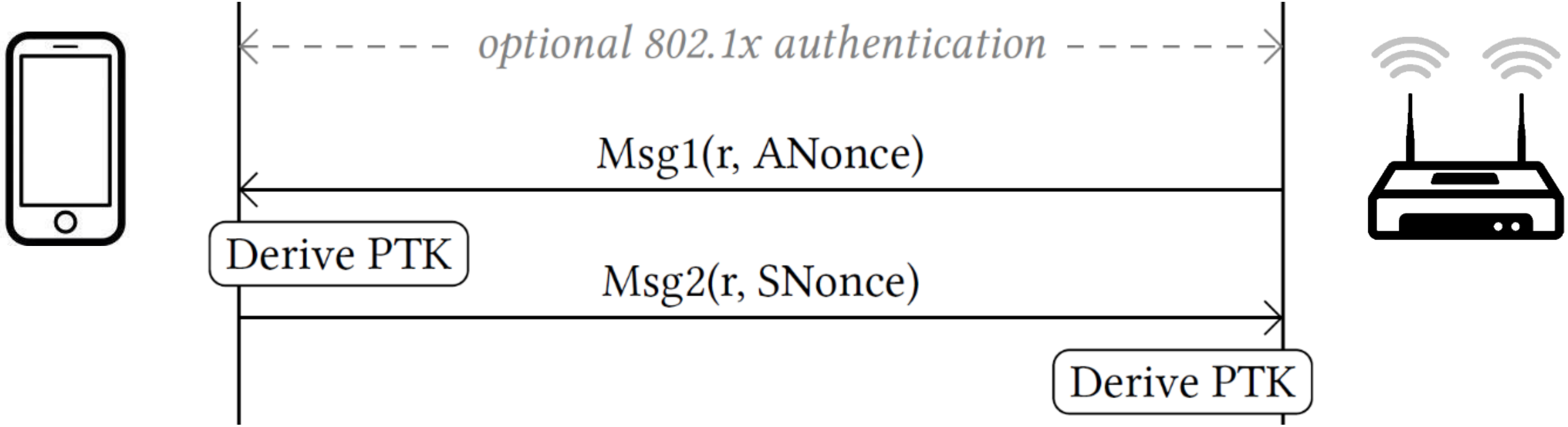
4-way handshake (simplified)

4-way handshake (simplified)

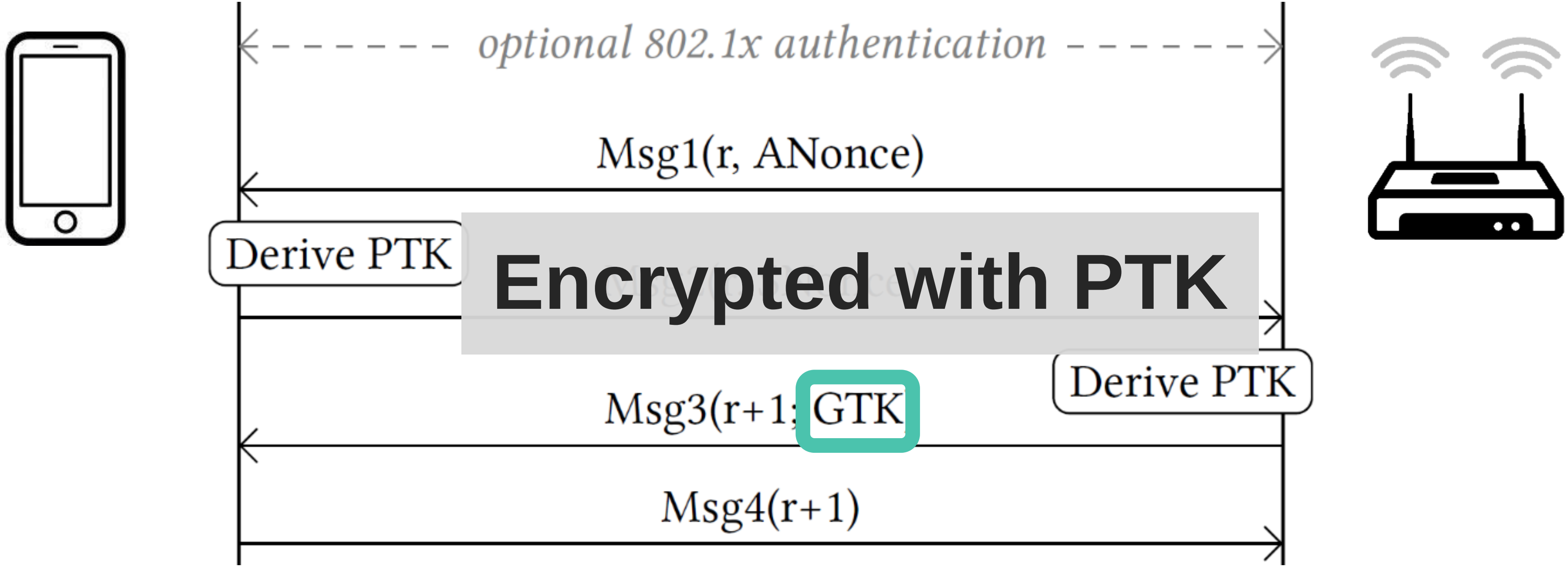


PTK = Combine(shared secret, ANonce, SNonce)

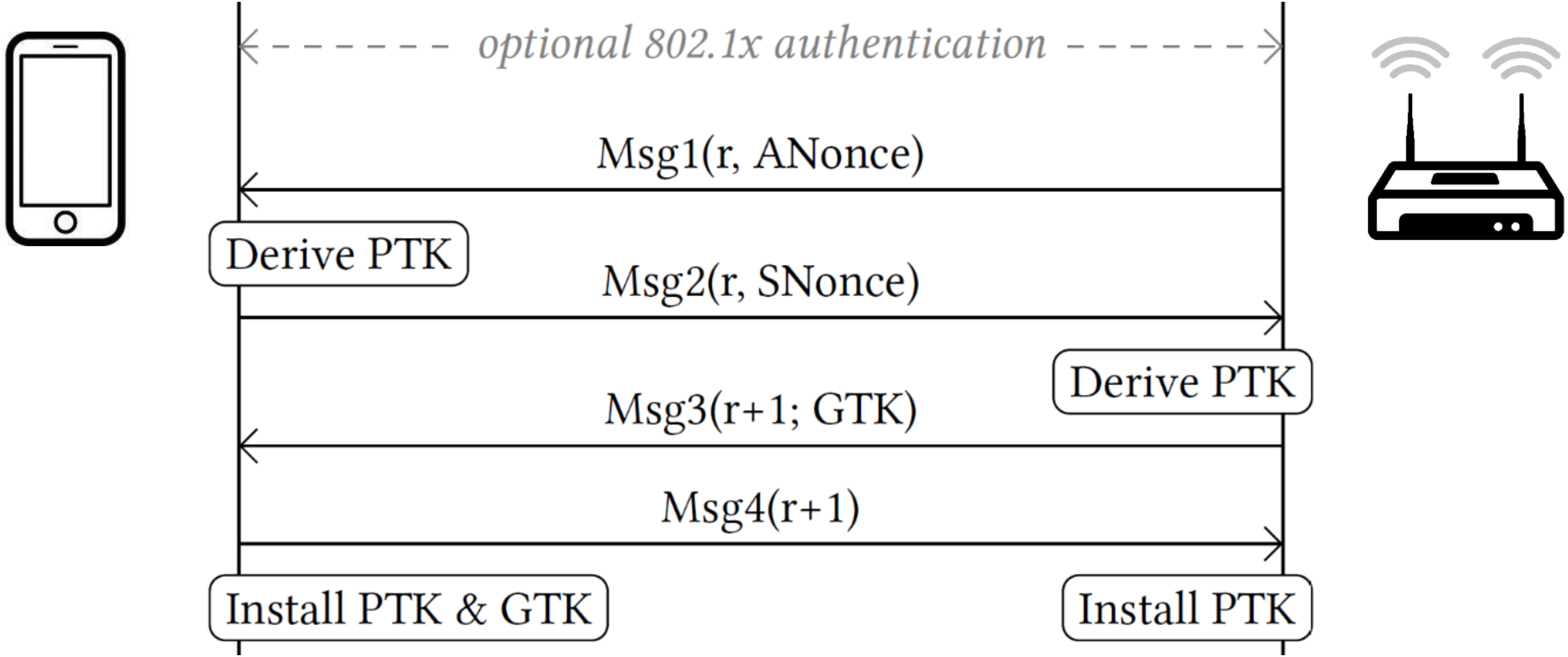
4-way handshake (simplified)



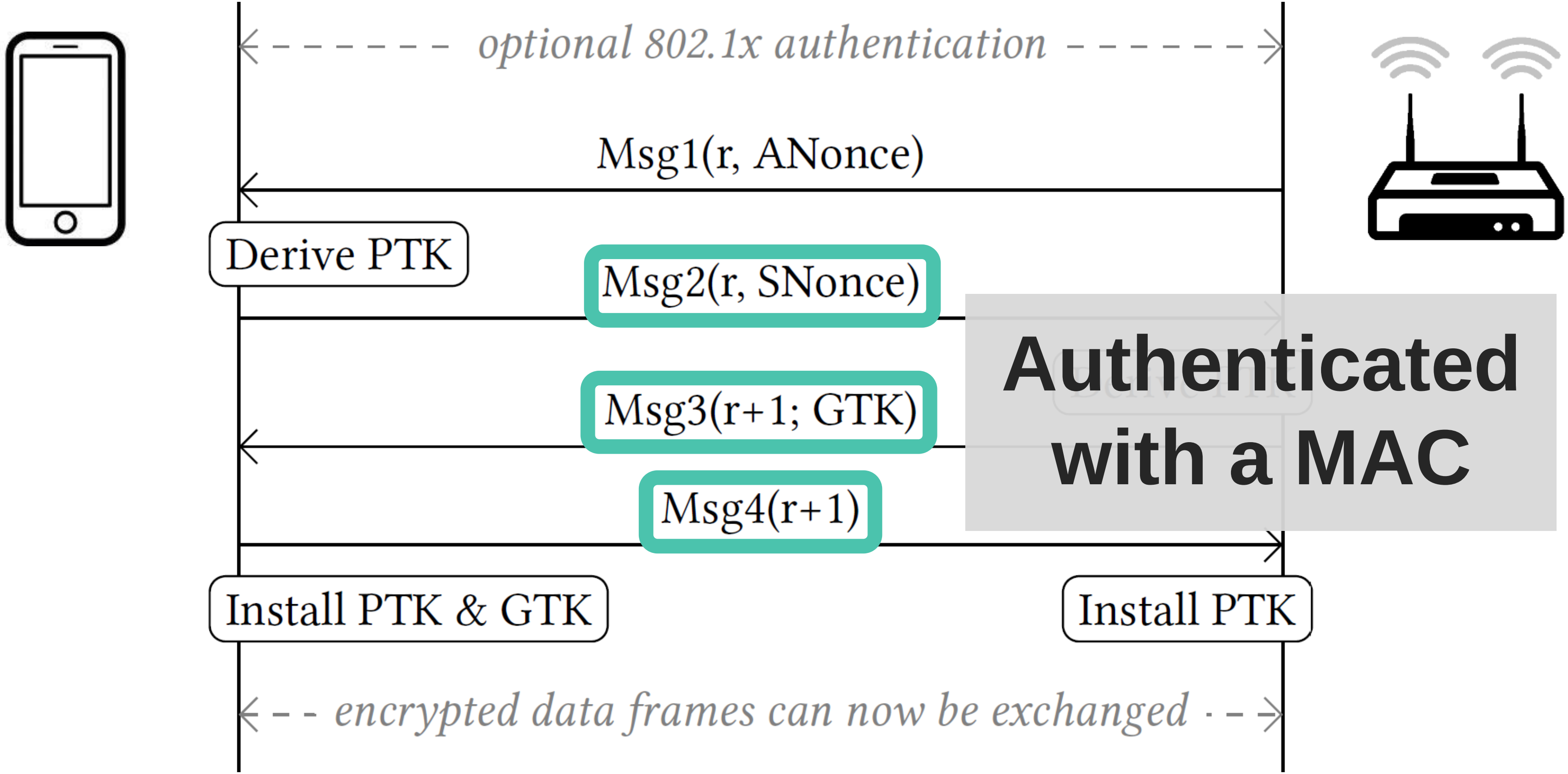
4-way handshake (simplified)



4-way handshake (simplified)



4-way handshake (simplified)



Authenticated with a MAC

We focus on the client

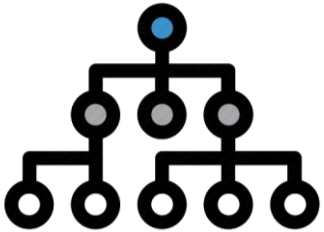
Symbolic execution of

Intel’s iwd deamon

wpa_supplicant

kernel driver

Overview



Symbolic Execution

4-way handshake

Handling Crypto

Results

Discovered Bugs I

Timing side-channels

- Authenticity tag not checked in constant time
- MediaTek and iwd are vulnerable

Denial-of-service in iwd

- Caused by integer underflow
- Leads to huge malloc that fails

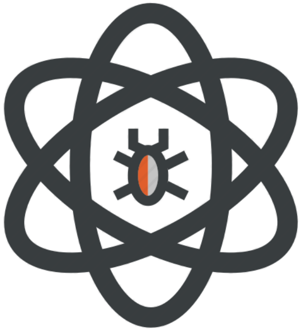
Decryption oracle in wpa_supplicant

Decryption oracle:

- Authenticity of Msg3 not checked
- But **decrypts and processes data**

→ Decrypt group key in Msg3

Discovered Bugs II



Buffer overflow in MediaTek kernel module

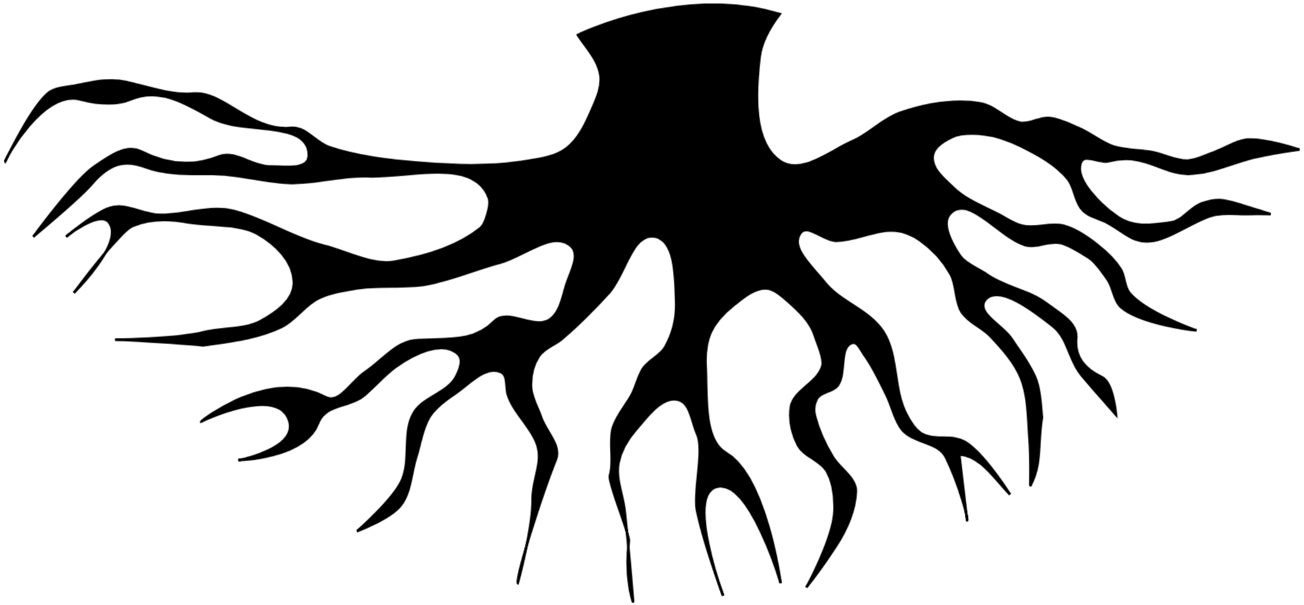
- Occurs when copying the group key
- **Remote code execution (details follow)**

Flawed AES unwrap crypto primitive

- Also in MediaTek's kernel driver
- **Manually discovered**

Rooting Routers:

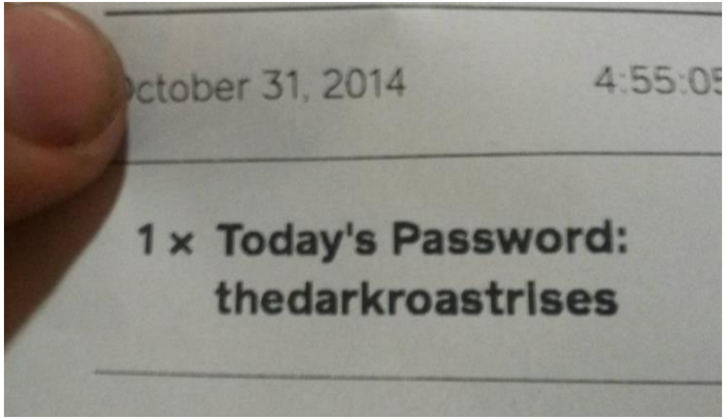
Buffer overflow in MediaTek kernel module



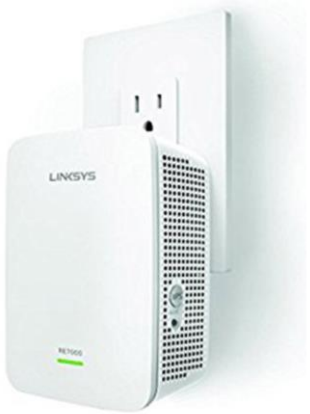
MediaTek buffer overflow preconditions I

Triggered when the **client** processes Msg3

- Adversary needs password of network
- Examples: Wi-Fi at conferences, hotels, etc.



MediaTek buffer overflow preconditions II



Which clients use the MediaTek driver?

- Not part of Linux kernel source tree
- **Used in repeater modes of routers**

Our target:

- RT-AC51U running Padavan firmware
- Original firmware has no WPA2 repeater

Popularity of Padavan firmware

Download repository	916.6 MB			
RT-AC54U_3.4.3.9-099_base.trx	7.0 MB	padavan	37142	2016-03-05
RT-AC51U_3.4.3.9-099_full.trx	9.6 MB	padavan	51270	2016-03-05
RT-AC51U_3.4.3.9-099_base.trx	7.0 MB	padavan	5380	2016-03-05
RT-N11P_3.4.3.9-099_nano.trx	2.9 MB	padavan	5134	2016-03-05
RT-N11P_3.4.3.9-099_base.trx	4.1 MB	padavan	8045	2016-03-05
RT-N14U_3.4.3.9-099_full.trx	9.2 MB	padavan	13856	2016-03-05

We exploit this version

The vulnerable code (simplified)

```
void RMTPParseEapolKeyData(pKeyData, KeyDataLen, MsgType) {
  UCHAR GTK[MAX_LEN_GTK];

  if (MsgType == PAIR_MSG3 || MsgType == GROUP_MSG_1) {
    P[illegible]ataLen, WPA2GTK);
    GTK_KDE *pKdeGtk = (GTK_KDE*)pKDE->octet;
    UCHAR GTKLEN = pKDE->Len - 6;
    NdisMoveMemory(GTK, pKdeGtk->GTK, GTKLEN);
  }
  [illegible]
  APCliInstallSharedKey(GTK, GTKLEN);
}
```

Len controlled by attacker

Destination buffer 32 bytes

d

Gaining kernel code execution

How to control return address & where to return?

- Kernel **doesn't use stack canaries**
- Kernel stack has **no address randomization**
- And the kernel stack is **executable**

Return to shellcode on stack & done?

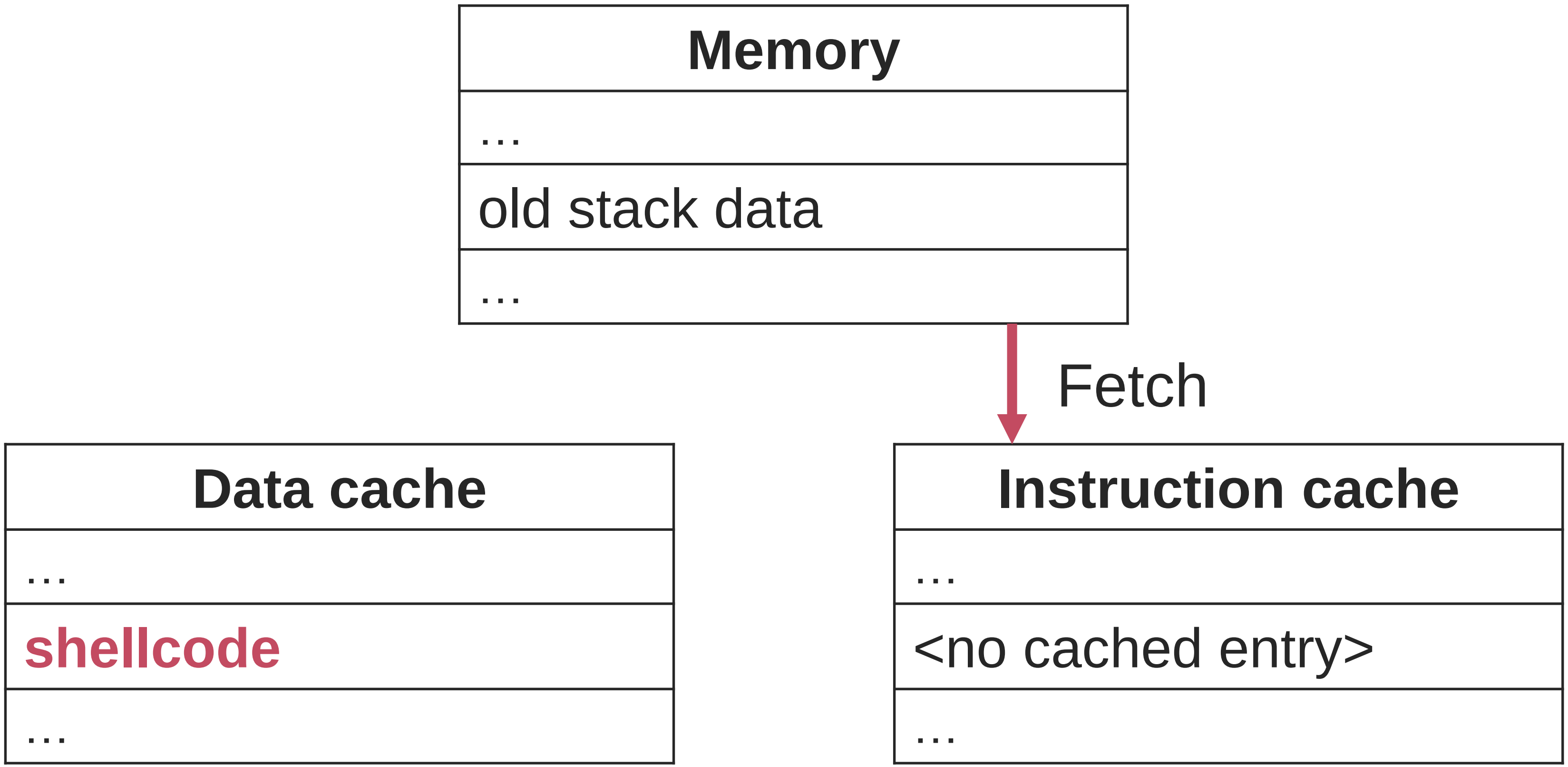
Nope… our shellcode crashes

Problem: cache incoherency on MIPS

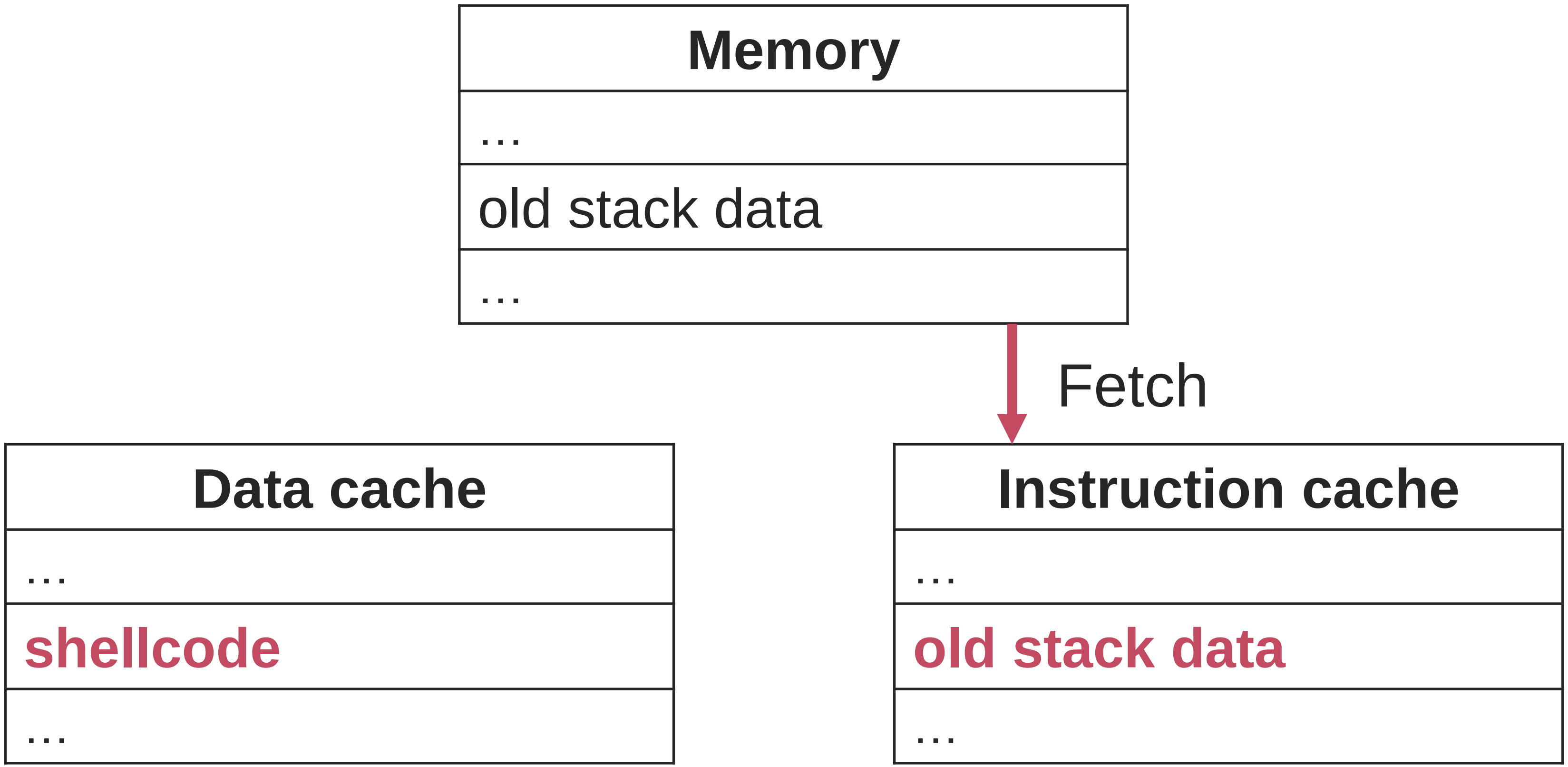
Memory
…
old stack data
…

Data cache
…
old stack data
…

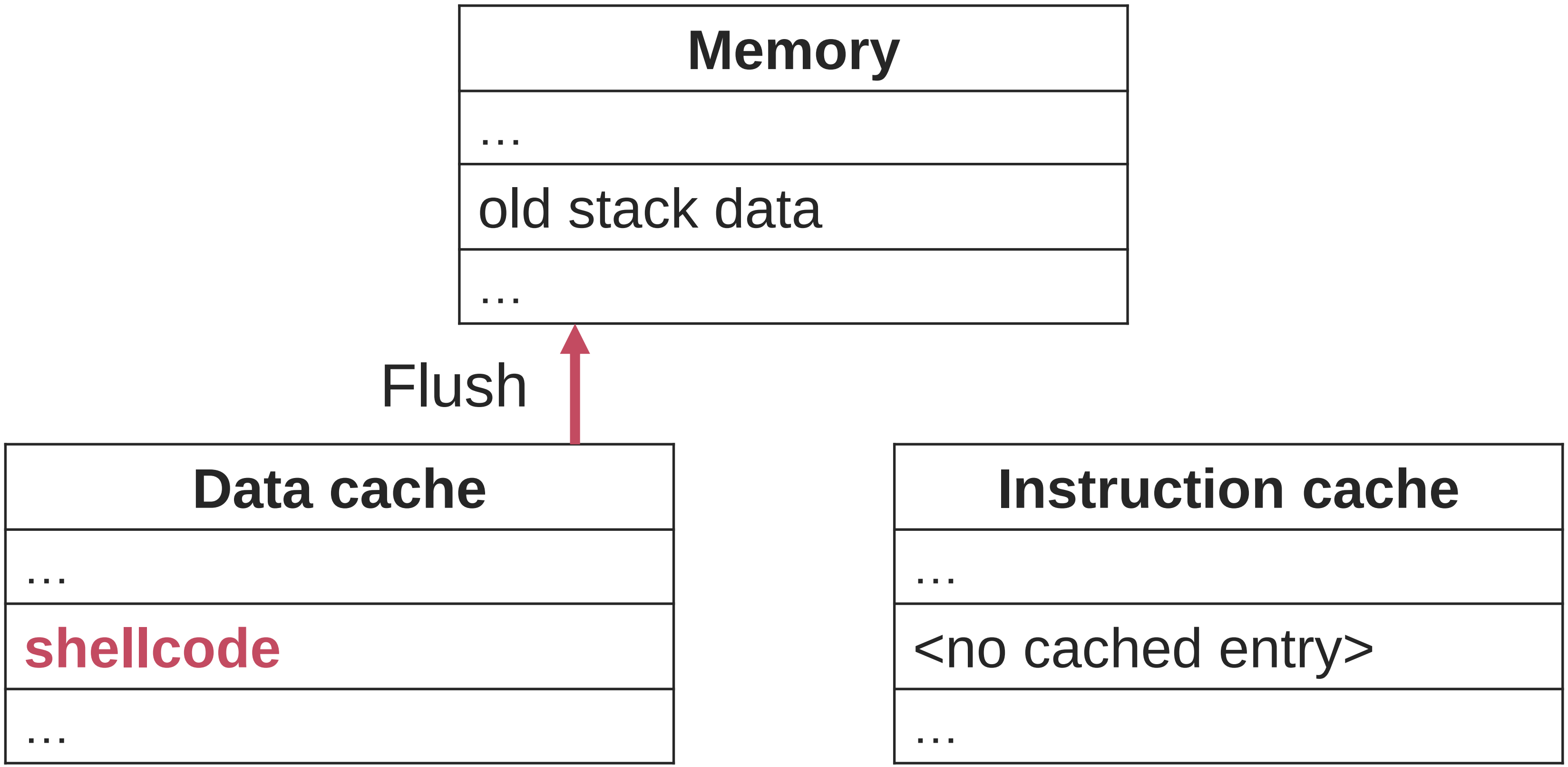
Problem: cache incoherency on MIPS



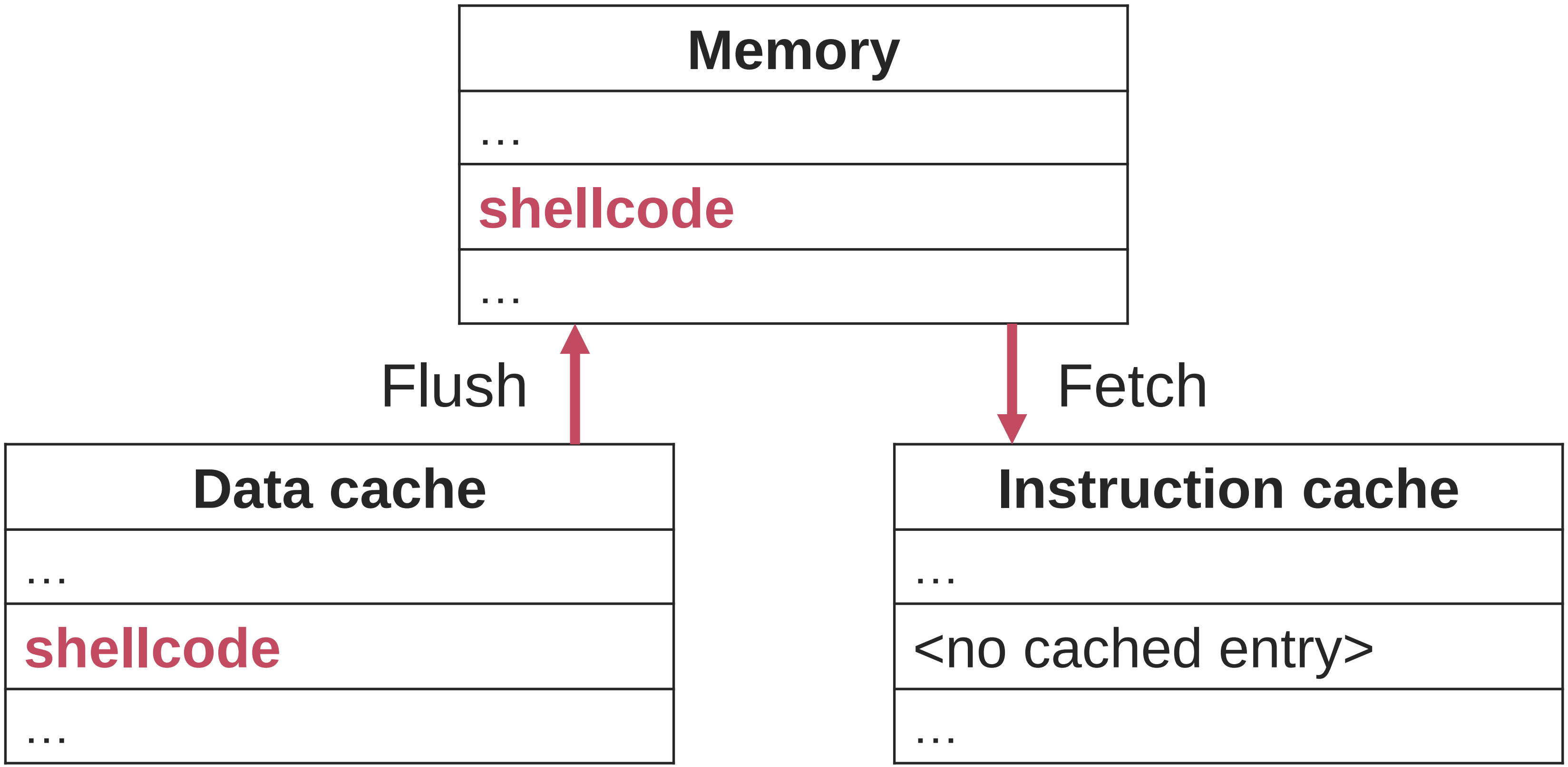
Problem: cache incoherency on MIPS



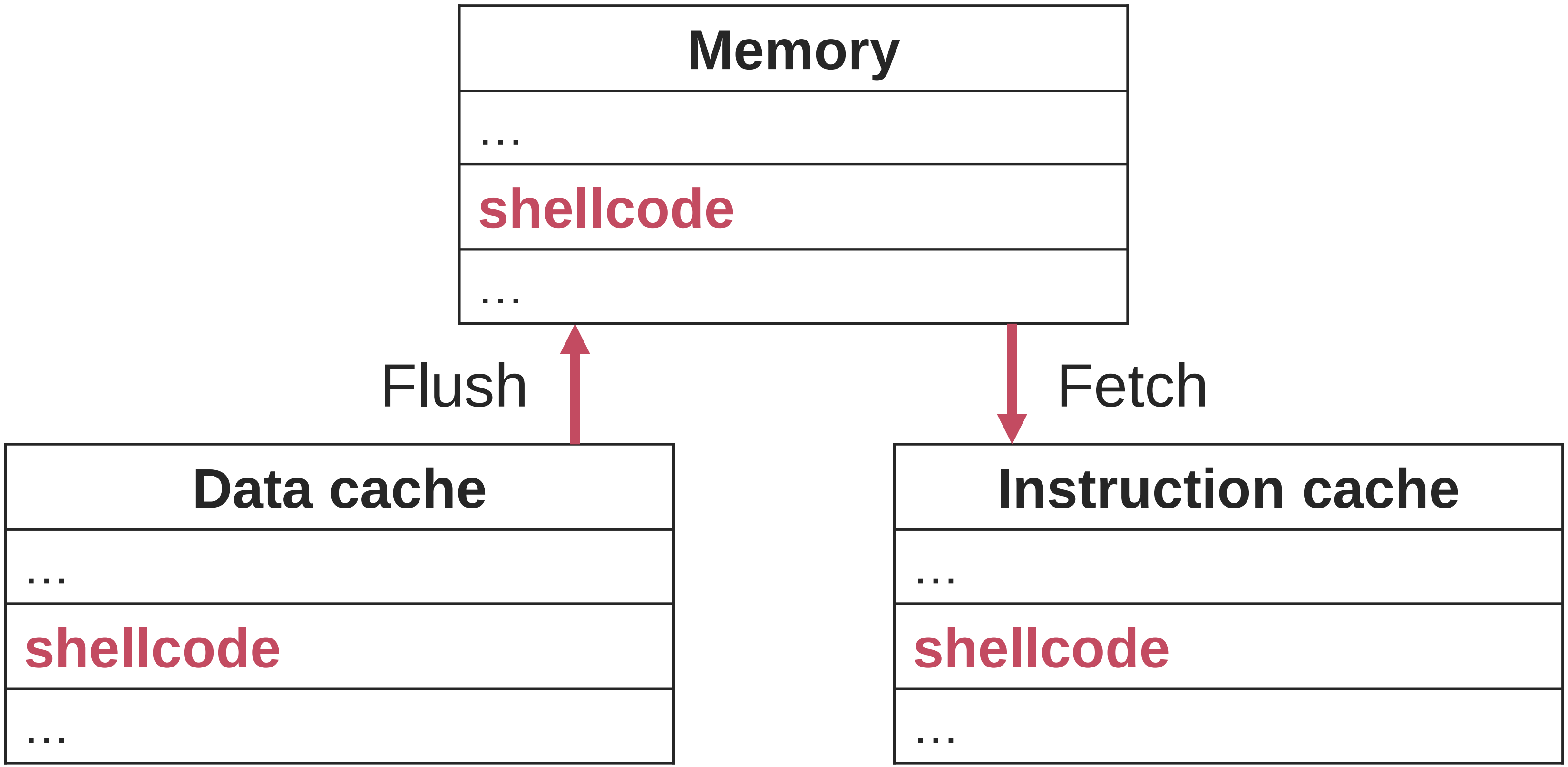
Solution: flush cache after write



Solution: flush cache after write



Solution: flush cache after write



How to flush the cache?

Execute kernel function to flush cache

- Rely on Return Oriented Programming (ROP)
- Use mipsrop tool of Craig Heffner

```
MIPS ROP Finder activated, found 1292 controllable jumps between 0x00000000 and 0x00078FE8
Python>mipsrop.tails()
----------------------------------------------------------------------------------------------------------
|  Address     |  Action                                                 |  Control Jump                   |
----------------------------------------------------------------------------------------------------------
|  0x0005E99C  |  move $t9,$a2                                           |  jr    $a2                      |
|  0x00061858  |  move $t9,$a2                                           |  jr    $a2                      |
|  0x00062D68  |  move $t9,$a2                                           |  jr    $a2                      |
----------------------------------------------------------------------------------------------------------
Found 3 matching gadgets
```

→ Building ROP chain is **tedious but doable**

Main exploitation steps

- Code execution in kernel
- **Obtain a process context**
- Inject shellcode in process
- Run injected shellcode

Let's spawn a shell?

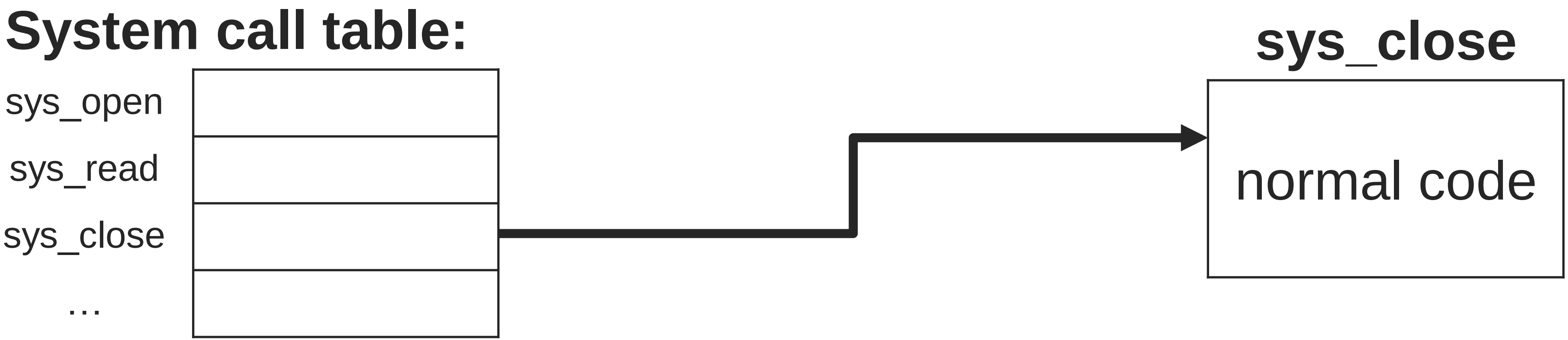
Tricky when in interrupt context

- Easier in process context: access to address space

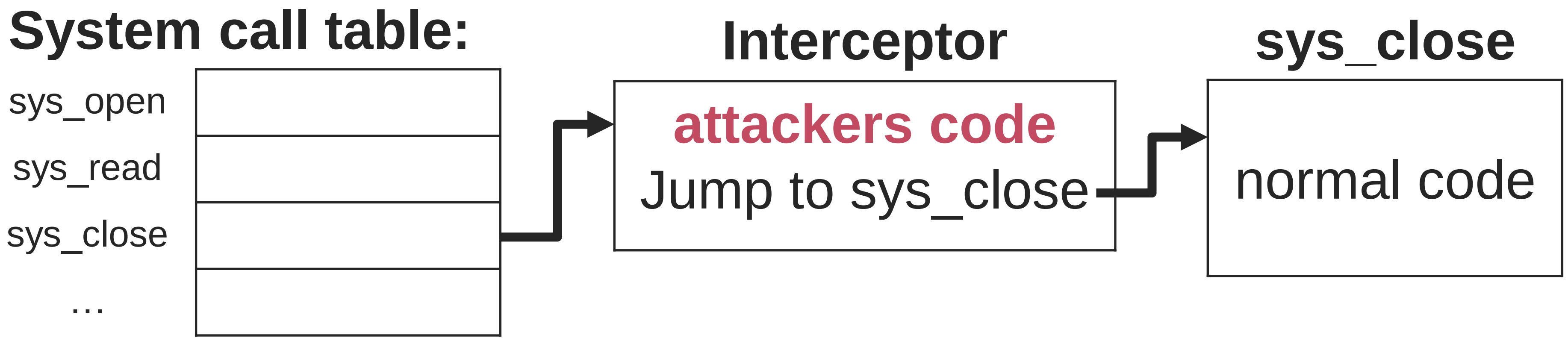
How to obtain a process context?

- System calls run in process context …
- … so intercept a close() system call

Intercepting system calls



Intercepting system calls



Main exploitation steps

- Code execution in kernel
- Obtain a process context
- **Inject shellcode in process**
- Run injected shellcode

Hijacking a process

When a process calls sys_close

- Hijack unimportant detect_link process
- Recognize by its predictable PID

Spawn a shell in the process:

1. Call **mprotect** to mark process code writable
2. **Copy user space shellcode** to return address
3. **Flush caches**

Main exploitation steps

- Code execution in kernel
- Obtain a process context
- Inject shellcode in process
- **Run injected shellcode**

User space shellcode

When close() returns, shellcode is triggered

- It runs “`telnetd -p 1337 -l /bin/sh`” using execve
- Adversary can now connect to router

Important remaks:

- Original process is killed, but causes no problems
- Used telnetd to keep shellcode small

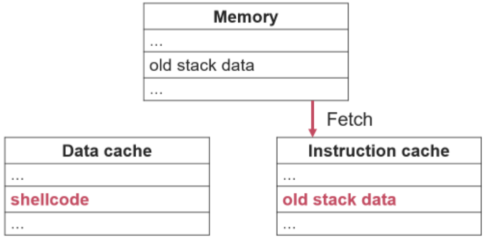
Running the full exploit

Multi-chain exploit. Space for shellcode?

- For initial stage we have 250 bytes
- Handshake frame can transport ~2048 bytes
- We can even use null bytes!

```
BusyBox v1.24.1 (2016-02-01 01:51:01 KRAT) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/home/root # uname -a
uname -a
Linux RT-AC51U 3.4.110 #1 Mon Feb 1 02:10:25 KRAT 2016 mips GNU/Linux
```

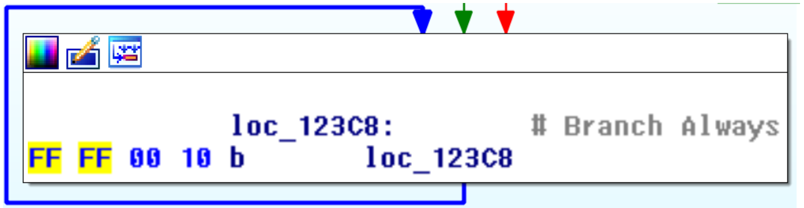
Exploit recap & lessons learned



Cache incoherence

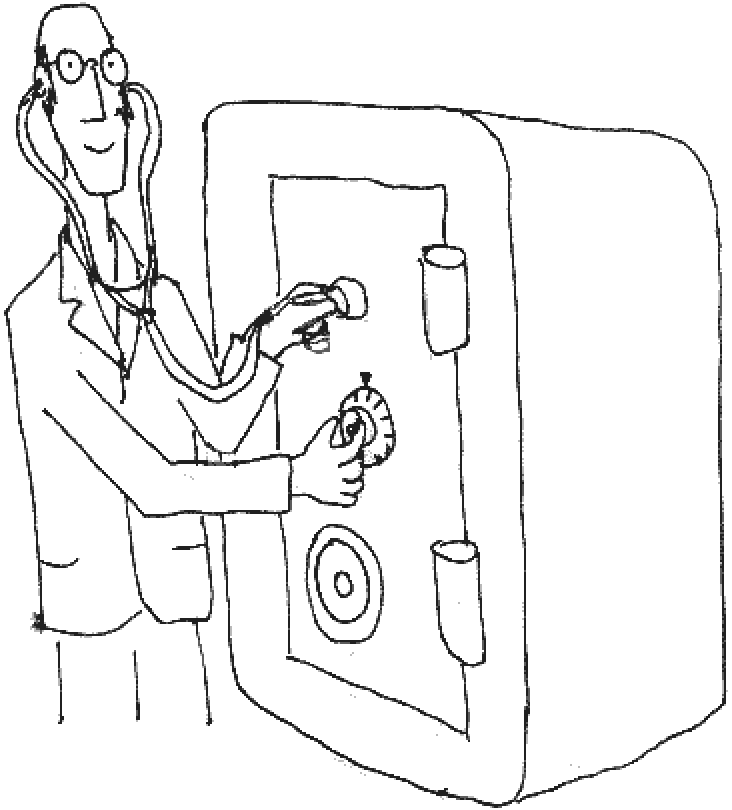
```c
idx = __NR_close - __NR_Linux;
real_close = (void*)*(sys_call_table +
*(sys_call_table + idx * 2) = (unsigned
flush_data_cache_page(sys_call_table +
printk("real_close = %p\n", real_close)
```

First test ideas in C



Debug with infinite loops

io.netgarage.org



Decryption Oracle

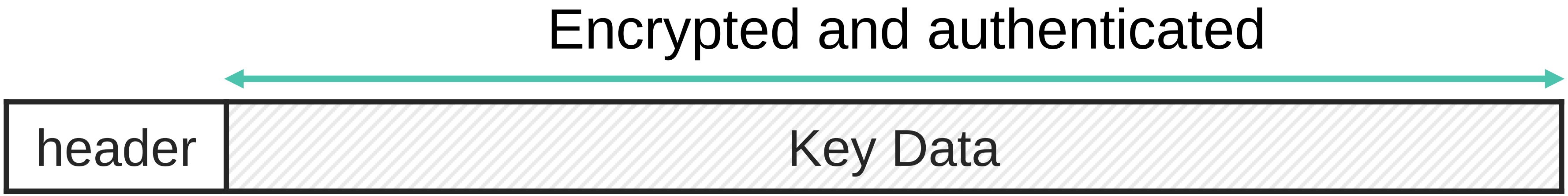
Recall: decryption oracle in wpa_supplicant

Decryption oracle:

- Authenticity of Msg3 not checked
- Does **decrypt and process data**

How can this be abused to leak data?

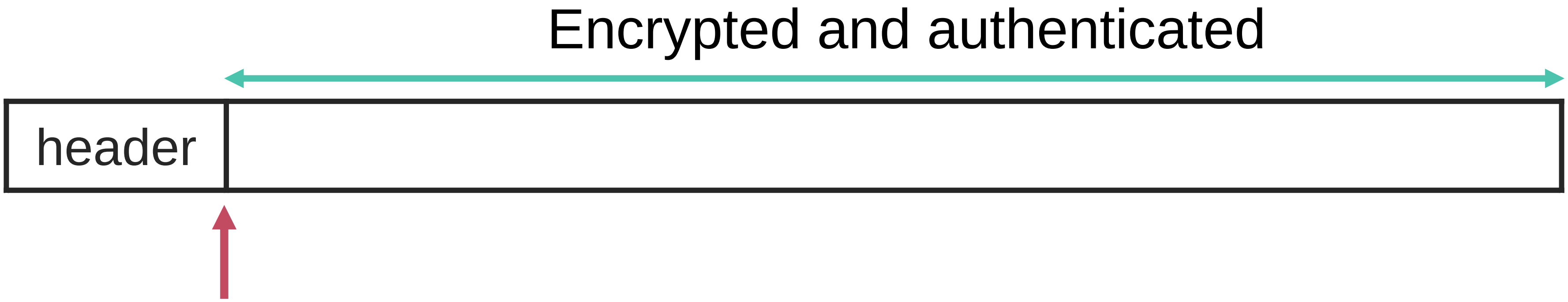
Background: process ordinary Msg3



On reception of Msg3 the receiver:

1. Decrypts the Key Data field

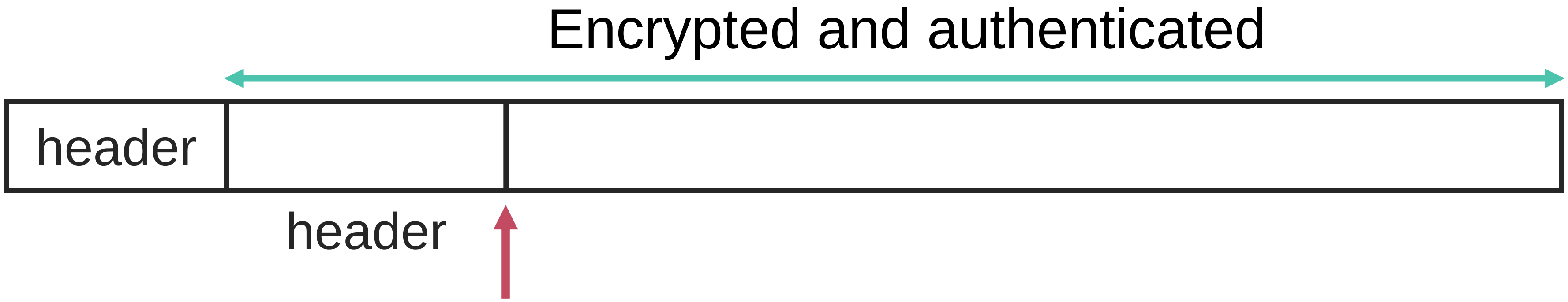
Background: process ordinary Msg3



On reception of Msg3 the receiver:

1. Decrypts the Key Data field
2. Parse payload header & content

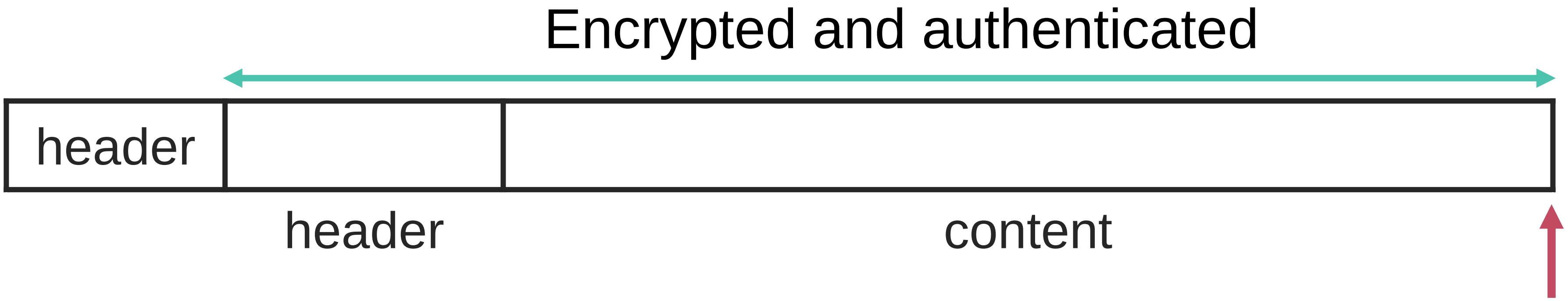
Background: process ordinary Msg3



On reception of Msg3 the receiver:

1. Decrypts the Key Data field
2. Parse payload header & content

Background: process ordinary Msg3



On reception of Msg3 the receiver:

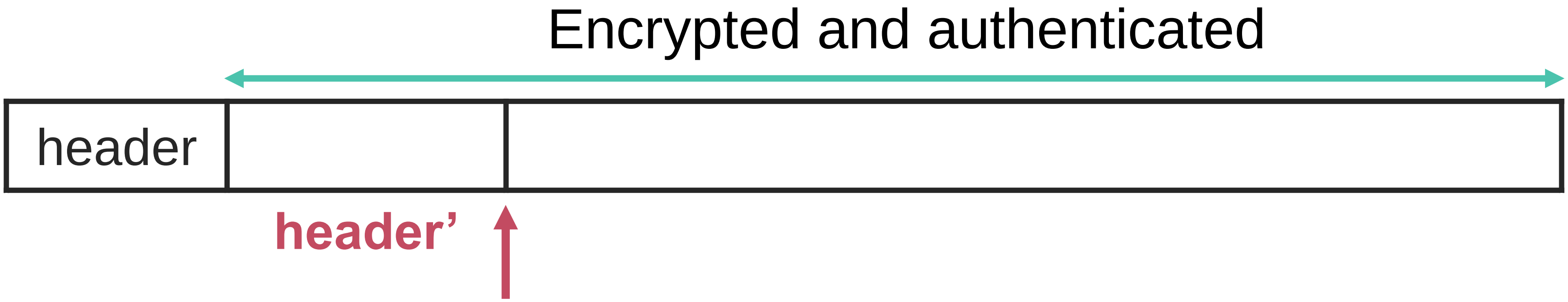
1. Decrypts the Key Data field
2. Parse payload header & content

How to turn parsing into an oracle?

Background: process ordinary Msg3

Adversary can modify the header

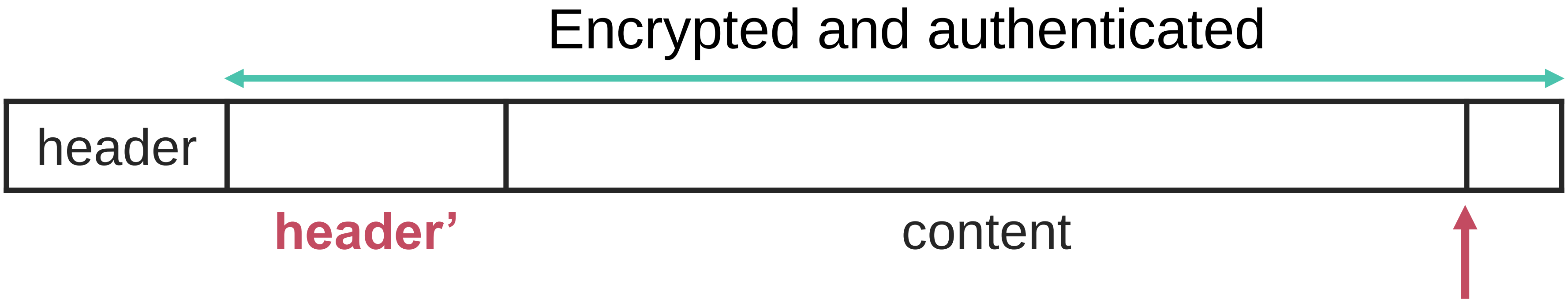
Background: process ordinary Msg3



Adversary can modify the header:

1. Receiver parser header successfully

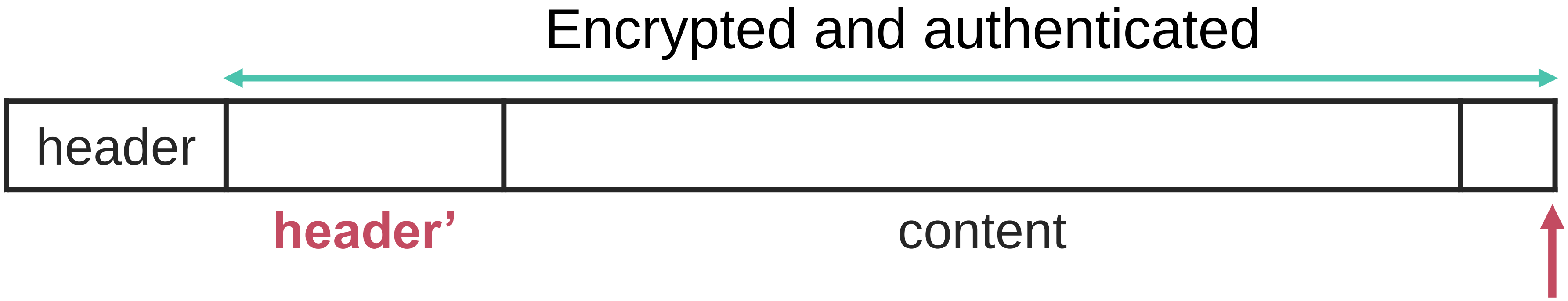
Background: process ordinary Msg3



Adversary can modify the header:

1. Receiver parser header successfully
2. Receiver interprets content differently (shorter)

Background: process ordinary Msg3



Adversary can modify the header:

1. Receiver parser header successfully
2. Receiver interprets content differently (shorter)
3. Parsing now **only succeeds if last byte is zero**

Practical aspects

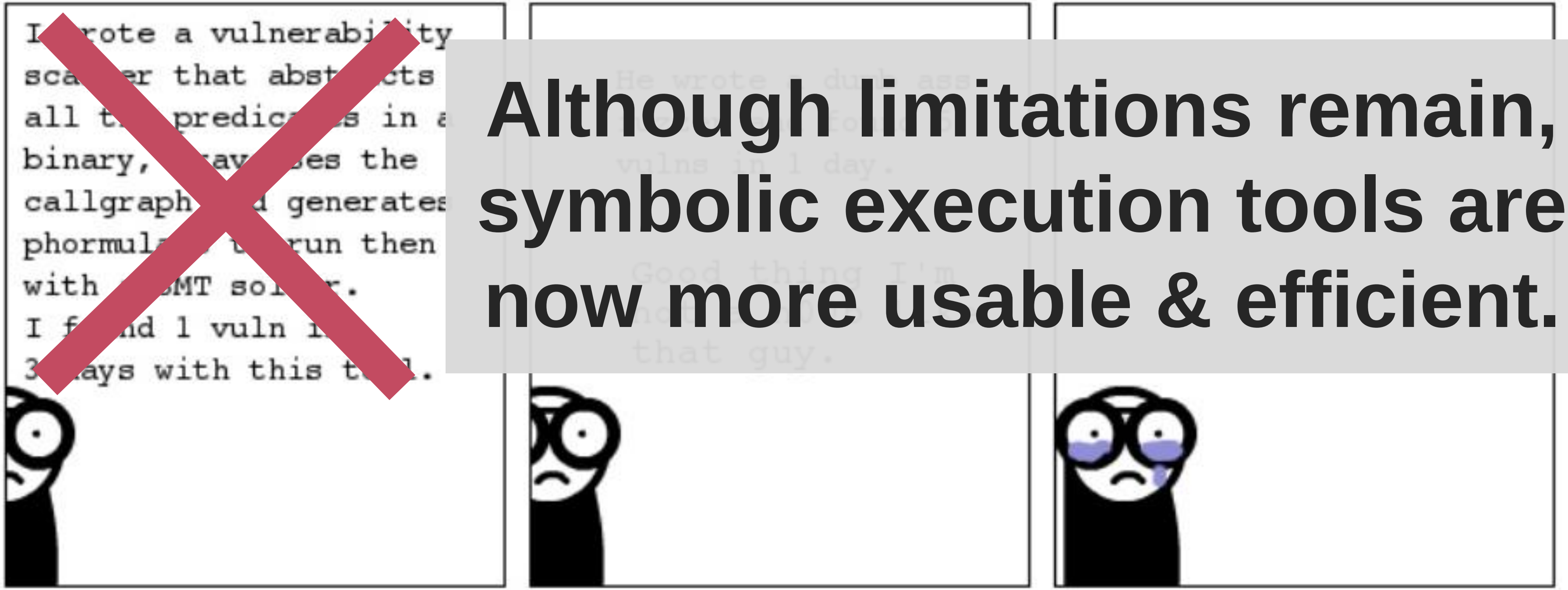
Test against Debian 8 client:

- Adversary can guess a value every 14 seconds
- Decrypting 16-byte group key takes ~8 hours

Attack can be made faster by:

- Attacking several clients simultaneously
- Can brute-force the last 4 bytes

The big picture



Although limitations remain, symbolic execution tools are now more usable & efficient.

Conclusion

- Symbolic execution of protocols
- Simple simulation of crypto
- Root exploit & decryption oracle
- Interesting future work

Thank you!

Questions?