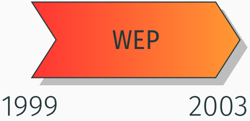
Dragonblood is Still Leaking: Practical Cache-based Side-Channel in the Wild

Daniel De Almeida Braga
Pierre-Alain Fouque
Mohamed Sabt

CORGIS - March, 15^{th} 2021

Context and Motivations

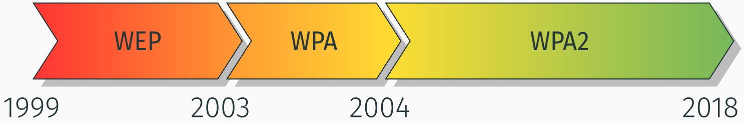
Context and Motivations



Context and Motivations

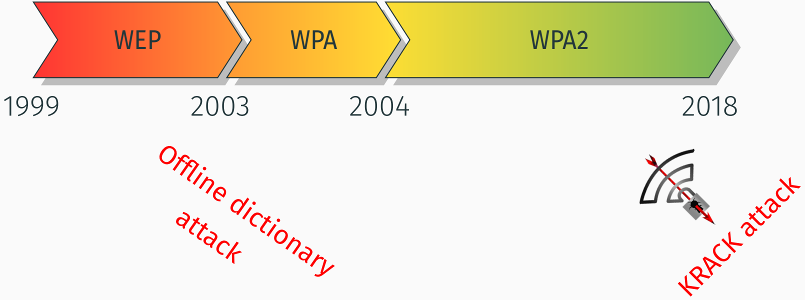
Context and Motivations

Context and Motivations

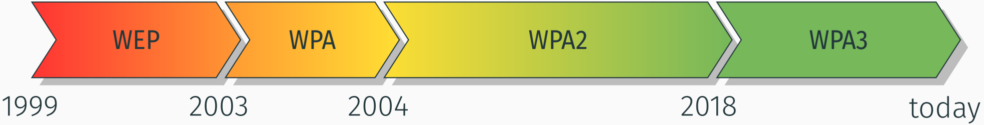


Offline dictionary attack

Context and Motivations



Context and Motivations



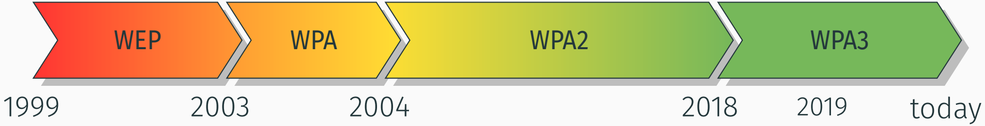
+ More secure
+ Based on a PAKE (Dragonfly[1])

[1] D. Harkins, 2015, *Dragonfly Key Exchange*, RFC 7664

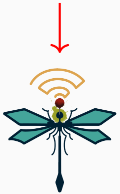
PAKE: Password Authenticated Key Exchange

- PAKE protocols aim to combine the Key Exchange and authentication parts
- Password is used to:
  - Authenticate the user
  - Derive strong cryptographic material
- No offline dictionary attack

Context and Motivations



+ More secure
+ Based on a PAKE (Dragonfly)



Dragonblood[1] attacks

[1] M. Vanhoef et al. *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020

Side Channel Attacks

```python
def processPassword(pwd):
    if "a" in pwd:
        res = long_processing(pwd)
    else:
        res = short_processing(pwd)
    return res
```

Side Channel Attacks

```python
def processPassword(pwd):
    if "a" in pwd:
        res = long_processing(pwd)
    else:
        res = short_processing(pwd)
    return res
```

Gain information through timing:

⏱ 0.5 seconds ⇒ no *a*

⏱ 10 seconds ⇒ *a*

Side Channel Attacks

```python
def processPassword(pwd):
    if "a" in pwd:
        res = long_processing(pwd)
    else:
        res = short_processing(pwd)
    return res
```

```python
def processPassword2(pwd):
    if "a" in pwd:
        res = long_processing(pwd)
    else:
        res = long_processing2(pwd)
    return res
```

Gain information through timing:

- 0.5 seconds ⇒ no *a*
- 10 seconds ⇒ *a*

Side Channel Attacks

```
def processPassword(pwd):
    if "a" in pwd:
        res = long_processing(pwd)
    else:
        res = short_processing(pwd)
    return res
```

Gain information through timing:

0.5 seconds ⇒ no *a*

10 seconds ⇒ *a*

```
def processPassword2(pwd):
    if "a" in pwd:
        res = long_processing(pwd)
    else:
        res = long_processing2(pwd)
    return res
```

Gain information execution flow:

- Execute `long_processing` ⇒ *a*
- Else, no *a* in pwd

Contributions

1. Show that current countermeasures are not sufficient for cache-based side-channel

Contributions

1. Show that current countermeasures are not sufficient for cache-based side-channel
2. Mount an offline dictionary attack to recover the password

Contributions

1. Show that current countermeasures are not sufficient for cache-based side-channel
2. Mount an offline dictionary attack to recover the password
3. Provide a PoC on Real-World-like scenarios (**IWD** and FreeRadius)

Contributions

1. Show that current countermeasures are not sufficient for cache-based side-channel
2. Mount an offline dictionary attack to recover the password
3. Provide a PoC on Real-World-like scenarios (**IWD** and FreeRadius)
4. Raise awareness on how practical these attacks are

Our main result

A cache based side channel attack

let us extract information during

the password conversion with

an offline dictionary attack

Our main result

Unintended information leakage

A cache based side channel attack

let us extract information during

the password conversion with

an offline dictionary attack

Our main result

FLUSH+RELOAD[1] and PDA[2]

Unintended information leakage

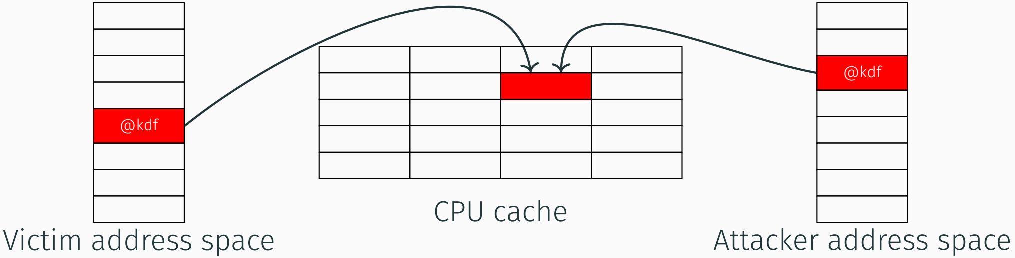
A cache based side channel attack

let us extract information during

the password conversion with

an offline dictionary attack

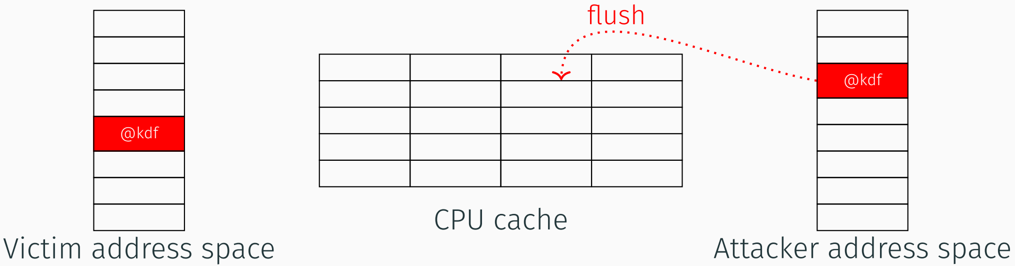
[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.
[2] T. Allan et al. *Amplifying side channels through performance degradation*. In ACSAC. 2016



1. Maps the victim's address space

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

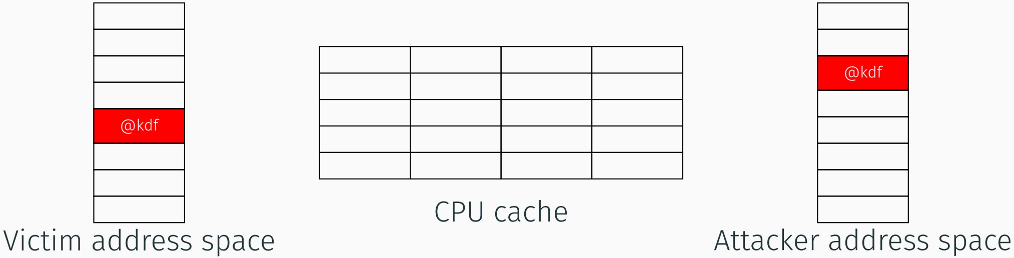
FLUSH+RELOAD[1]



1. Maps the victim's address space
2. Flush the instruction we monitor

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

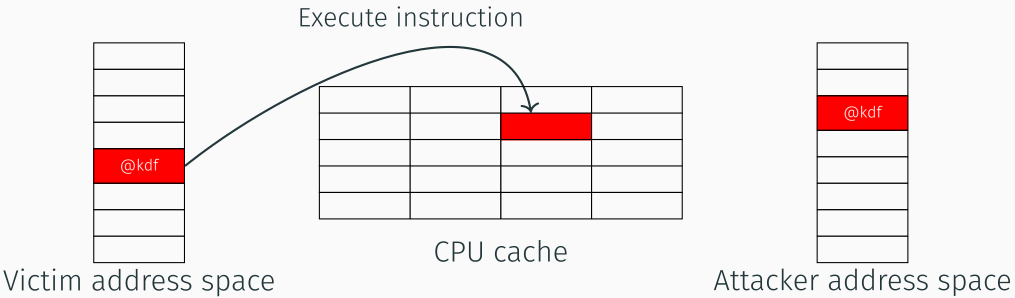
Flush+Reload[1]



1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

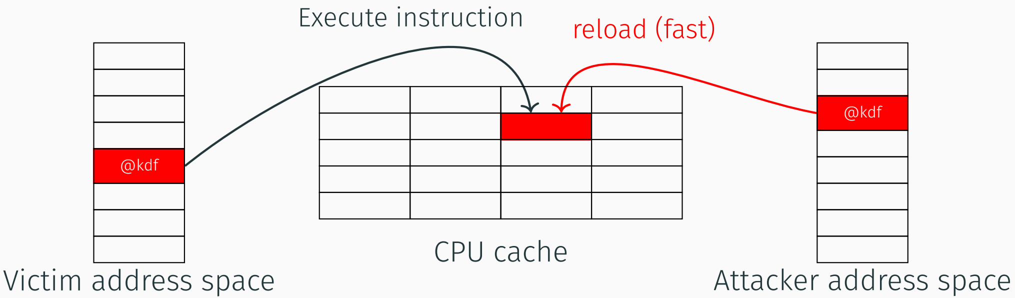
FLUSH+RELOAD[1]



1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

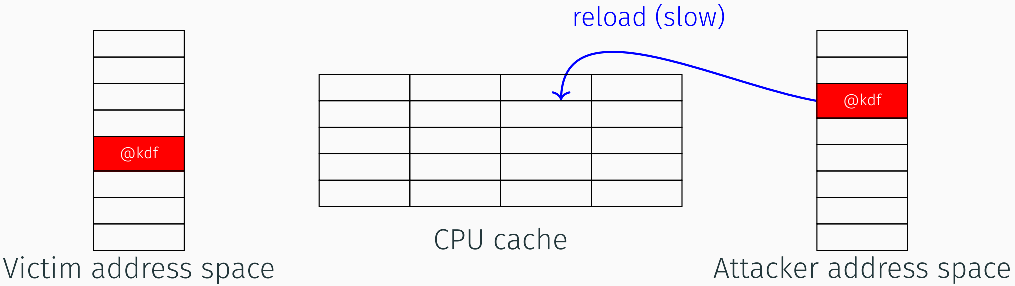
Flush+Reload[1]



1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

Flush+Reload[1]



1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

Our main result

FLUSH+RELOAD and PDA

Unintended information leakage

A cache based side channel attack

let us extract information during

the password conversion with

an offline dictionary attack

Our main result

FLUSH+RELOAD and PDA

Unintended information leakage

A **cache** based **side channel** attack

let us extract information during

the **password conversion** with

an **offline dictionary attack**

Password to point on an Elliptic Curve

Our main result

FLUSH+RELOAD and PDA

Unintended information leakage

A cache based side channel attack

let us extract information during

the password conversion with

an offline dictionary attack

Password to point on an Elliptic Curve

Passive attacker can eliminate wrong passwords from a list

Dragonfly Protocol Overview

A and B agree on a prime order group $E(\mathbb{F}_p)$, of order q

Dragonfly

Alice (A)		**Bob (B)**
$P \leftarrow \mathbf{p2g}(pwd, A, B)$		$P \leftarrow \mathbf{p2g}(pwd, A, B)$
	Commit $\longrightarrow$	
	$\longleftarrow$	
Key derivation		Key derivation
	Confirmation $\longrightarrow$	
	$\longleftarrow$	

Dragonfly Protocol Overview

A and B agree on a prime order group $E(\mathbb{F}_p)$, of order q

Dragonfly

Alice (A)		**Bob (B)**
$P \leftarrow \mathbf{p2g}(pwd, A, B)$		$P \leftarrow \mathbf{p2g}(pwd, A, B)$
	Commit $\longrightarrow$	
	$\longleftarrow$	
Key derivation		Key derivation
	Confirmation $\longrightarrow$	
	$\longleftarrow$	

Dragonfly - Password Conversion (EC)

HuntingAndPecking($pwd, A, B, k = 40$)

$found,\ i =$ **false**, 1
while not $found$ **or** $i < k$:
    $seed = Hash(A, B, pwd, i)$
    $x_{cand} = KDF(seed, label)$
    if x_{cand} is a point's coordinate :
        if not $found$:
            $found,\ x, seed_x =$ **true**, $x_{cand},\ seed$
            $pwd = get\_random()$
    $i\ = i + 1$
$y = set\_compressed\_point\_coordinate(x, seed_x)$
return (x, y)

Dragonfly - Password Conversion (EC)

HuntingAndPecking($pwd, A, B, k = 40$)

```
found, i = false, 1
while not found or i < k :
    seed = Hash(A, B, pwd, i)
    x_cand = KDF(seed, label)
    if x_cand is a point's coordinate :
        if not found :
            found, x, seed_x = true, x_cand, seed
            pwd = get_random()
    i = i + 1
y = set_compressed_point_coordinate(x, seed_x)
return (x, y)
```

← : new iteration

Dragonfly - Password Conversion (EC)

HuntingAndPecking($pwd, A, B, k = 40$)

$found,\ i = \textbf{false},\ 1$

while not $found$ **or** $i < k$:

  $seed = Hash(A, B, pwd, i)$

  $x_{cand} = KDF(seed, label)$ ← : new iteration

  if x_{cand} is a point's coordinate :

    if not $found$:

      $found,\ x, seed_x = \textbf{true},\ x_{cand},\ seed$ ← : successful conversion

      $pwd = get\_random()$

  $i\ = i + 1$

$y = set\_compressed\_point\_coordinate(x, seed_x)$

return (x, y)

Dragonfly - Password Conversion (EC)

HuntingAndPecking($pwd, A, B, k = 40$)

$found,\ i = $ **false**, 1

while not $found$ **or** $i < k$:

  $seed = Hash(A, B, pwd, i)$

  $x_{cand} = KDF(seed, label)$ ← : new iteration

  if x_{cand} is a point's coordinate : → `mask = get_random()` `do_blind_verif(`x_{cand}`, mask)`

    if not $found$:

      $found,\ x, seed_x =$ **true**, $x_{cand},\ seed$ ← : successful conversion

      $pwd = get\_random()$

  $i\ = i + 1$

$y = set\_compressed\_point\_coordinate(x, seed_x)$

return (x, y)

Dragonfly - Password Conversion (EC)

HuntingAndPecking($pwd, A, B, k = 40$)

```
found, i = false, 1
while not found or i < k :
    seed = Hash(A, B, pwd, i)
    x_cand = KDF(seed, label)                        ← : new iteration
    if x_cand is a point's coordinate :           →  mask = get_random()
                                                     do_blind_verif(x_cand, mask) ← PDA
        if not found :
            found, x, seed_x = true, x_cand, seed    ← : successful conversion
            pwd = get_random()
    i = i + 1
y = set_compressed_point_coordinate(x, seed_x)
return (x, y)
```

Dictionary Reduction

	Iter. required for A, B	Iter. required for A, B’
Leakage	3	
password1		
password2		
password3		
password4		
…		
passwordn		

Dictionary Reduction

	Iter. required for A, B	Iter. required for A, B’
Leakage	3	
password1	1	
password2	3	
password3	3	
password4	4	
…	…	
passwordn	3	

Dictionary Reduction

	Iter. required for A, B	Iter. required for A, B'
Leakage	3	
password1	1	
password2	3	
password3	3	
password4	4	
...	...	
passwordn	3	

Dictionary Reduction

	Iter. required for A, B	Iter. required for A, B'
Leakage	3	2
password1	1	
password2	3	
password3	3	
password4	4	
...	...	
passwordn	3	

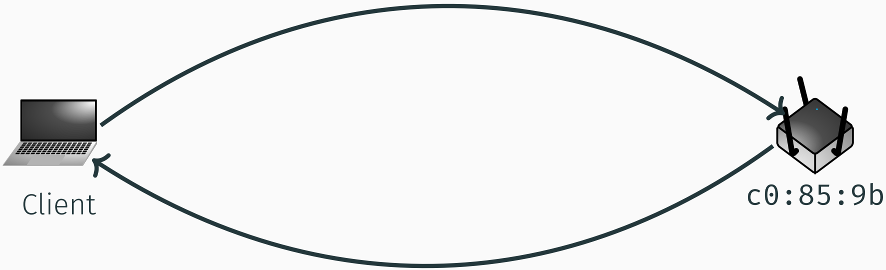
Dictionary Reduction

	Iter. required for A, B	Iter. required for A, B’
Leakage	3	2
password1	1	X
password2	3	8
password3	3	2
password4	4	X
…	…	…
passwordn	3	1

Dictionary Reduction

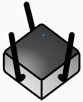
	Iter. required for A, B	Iter. required for A, B'
Leakage	3	2
password1	1	X
password2	3	8
password3	3	2
password4	4	X
...	...	...
passwordn	3	1

Attacker Model



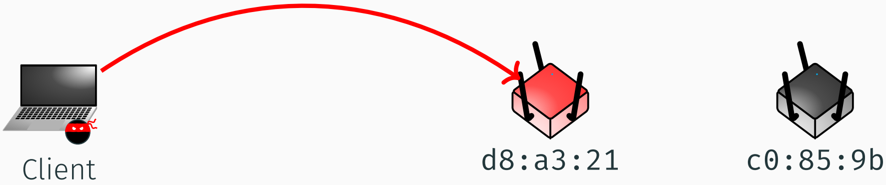
Attacker Model

Client

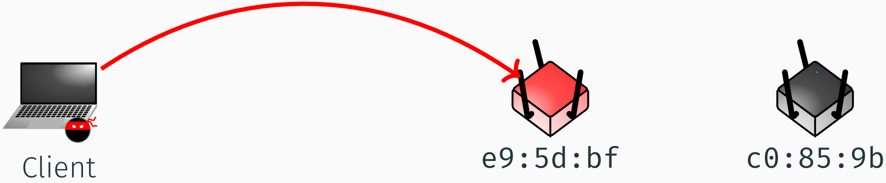


`c0:85:9b`

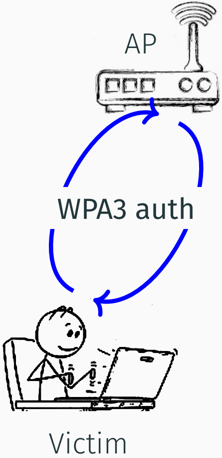
Attacker Model



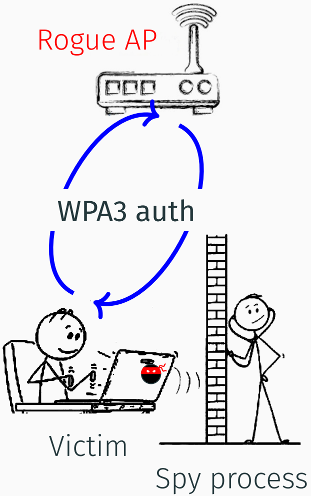
Attacker Model



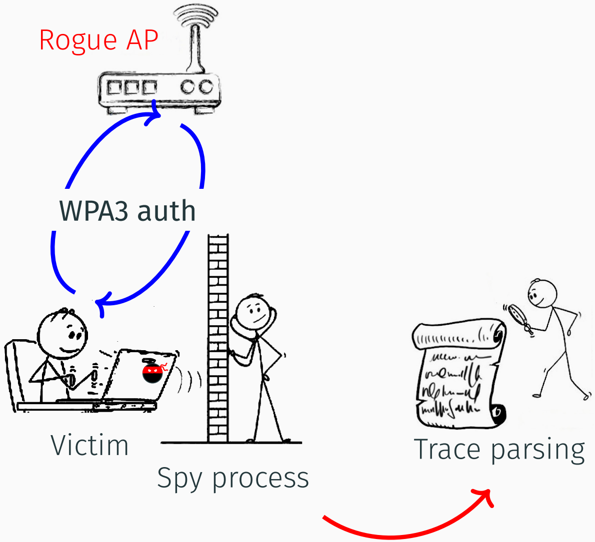
Core Idea



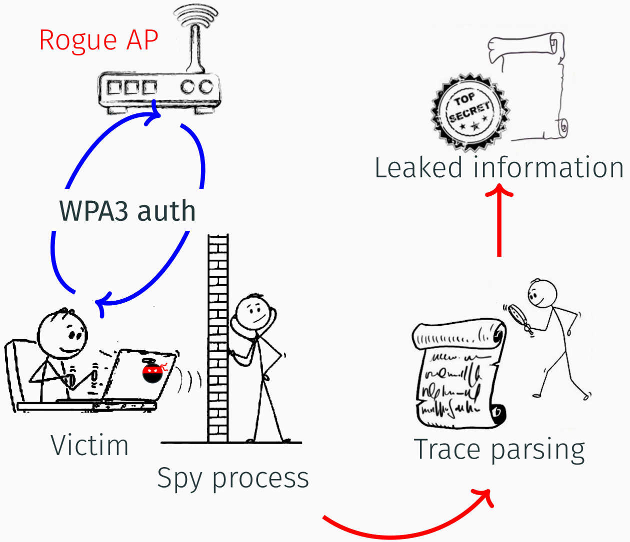
Core Idea

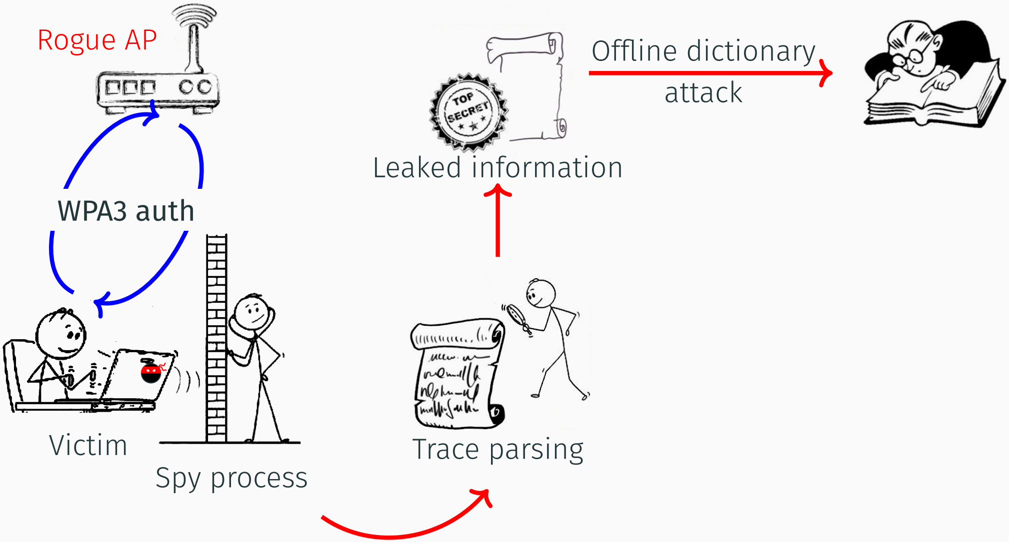


Core Idea

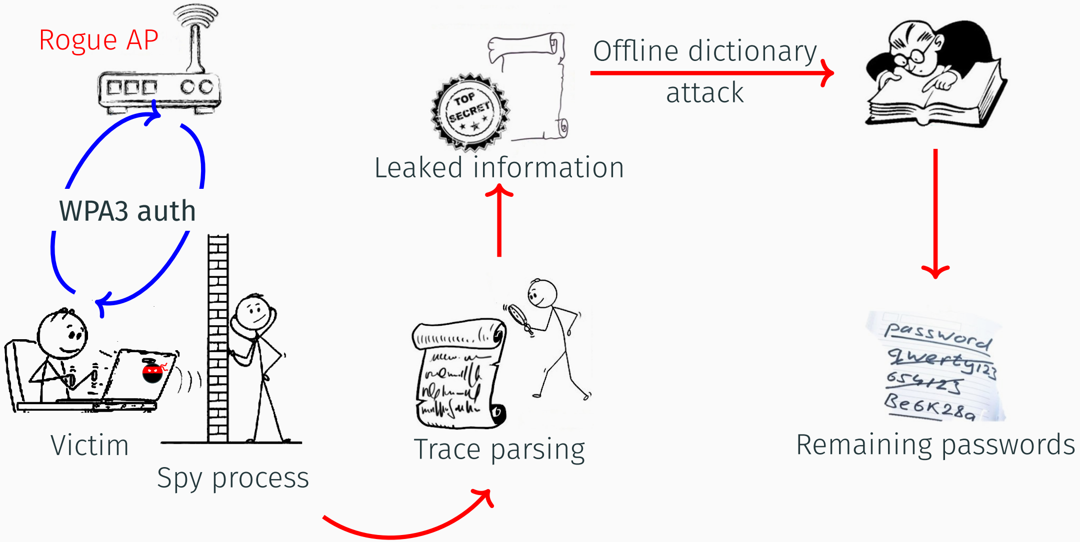


Core Idea


Rogue AP
WPA3 auth
Victim
Spy process
Leaked information
TOP SECRET
Trace parsing


Rogue AP
WPA3 auth
Victim
Spy process
Trace parsing
Leaked information
Offline dictionary attack

Core Idea


Rogue AP
WPA3 auth
Victim
Spy process
Trace parsing
Leaked information
TOP SECRET
Offline dictionary attack
Remaining passwords

Practical Results

- Need multiple measurement to achieve high accuracy

Practical Results

- Need multiple measurement to achieve high accuracy
- Very reliable results with only 10 measurements per MAC address

Practical Results

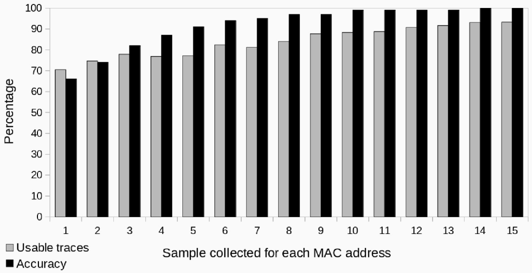
- Need multiple measurement to achieve high accuracy
- Very reliable results with only 10 measurements per MAC address
- More than 1 bit of information for each MAC

Practical Results

- Need multiple measurement to achieve high accuracy
- Very reliable results with only 10 measurements per MAC address
- More than 1 bit of information for each MAC
- Original attack: 20 measurement for exactly one bit of information

Practical Results

- Need multiple measurement to achieve high accuracy
- Very reliable results with only 10 measurements per MAC address
- More than 1 bit of information for each MAC
- Original attack: 20 measurement for exactly one bit of information



Practical Results

	Dict. size	Cost on AWS	Avg traces for full reduction
Rockyou	$1.4 \cdot 10^7$	$0,00037$ €	16
CrackStation	$3.5 \cdot 10^7$	$0,0011$ €	17
HaveIBeenPwned	$5.5 \cdot 10^8$	0,014 €	20
8 characters	$4.6 \cdot 10^{14}$	$11848,2$ €	32

Number of the Required Traces / Cost to Prune all Wrong Passwords

Practical Results

	Dict. size	Cost on AWS	Avg traces for full reduction
Rockyou	$1.4 \cdot 10^7$	$0,00037$ €	16
CrackStation	$3.5 \cdot 10^7$	$0,0011$ €	17
HaveIBeenPwned	$5.5 \cdot 10^8$	0,014 €	20
8 characters	$4.6 \cdot 10^{14}$	$11848,2$ €	32

Number of the Required Traces / Cost to Prune all Wrong Passwords

Practical Results

	Dict. size	Cost on AWS	Avg traces for full reduction
Rockyou	$1.4 \cdot 10^7$	$0,00037$ €	16
CrackStation	$3.5 \cdot 10^7$	$0,0011$ €	17
HaveIBeenPwned	**$5.5 \cdot 10^8$**	**0,014 €**	**20**
8 characters	$4.6 \cdot 10^{14}$	$11848,2$ €	32

Number of the Required Traces / Cost to Prune all Wrong Passwords

IWD v1.9 ✓

2020-08-03	sae: Fix a side channel leak on the password	Daniel DE ALMEIDA BRAGA	2	-40/+135

FreeRadius to be fixed in 3.0.22

merge constant time fixes from "master" ...

```
Based on a patch from Daniel De Almeida Braga.

The code is now largely the same between master and v3.0.x,
which makes it easier to see that it's correct
```

Additional vulnerability (found after the paper submission)

HuntingAndPecking(*pwd*, *A*, *B*, *k*)

```
found, i = false, 1
while not found or i < k :
    seed = Hash(A, B, pwd, i)
    x_cand = KDF(seed, label)
    if x_cand is a point's coordinate :
        if not found :
            found, x, save_seed = true, x_cand, seed
    i = i + 1
y = set_compressed_point_coordinate(x, save_seed)
return (x, y)
```

Additional vulnerability (found after the paper submission)

HuntingAndPecking(pwd, A, B, k)

$found,\ i = \textbf{false},\ 1$

while not $found$ **or** $i < k$:

    $seed = Hash(A, B, pwd, i)$

    $x_{cand} = KDF(seed, label)$

    if x_{cand} is a point's coordinate :

        if not $found$:

            $found,\ x, save\_seed = \textbf{true},\ x_{cand},\ seed$

    $i\ = i + 1$

$y = set\_compressed\_point\_coordinate(x, save\_seed)$ $\leftarrow$: leaks the seed's parity

return (x, y)

Dictionary Reduction

	seed's parity for A, B	seed's parity for A, B'
Leakage	0	
password1		
password2		
password3		
password4		
...		
passwordn		

Dictionary Reduction

	seed's parity for A, B	seed's parity for A, B'
Leakage	0	
password1	1	
password2	0	
password3	0	
password4	1	
...	...	
passwordn	0	

Dictionary Reduction

	seed's parity for A, B	seed's parity for A, B'
Leakage	0	
password1	1	
password2	0	
password3	0	
password4	1	
...	...	
passwordn	0	

Dictionary Reduction

	seed's parity for A, B	seed's parity for A, B'
Leakage	0	1
password1	1	
password2	0	
password3	0	
password4	1	
...	...	
passwordn	0	

Dictionary Reduction

	seed's parity for A, B	seed's parity for A, B'
Leakage	0	1
password1	1	X
password2	0	0
password3	0	1
password4	1	X
…	…	…
passwordn	0	0

Dictionary Reduction

	seed's parity for A, B	seed's parity for A, B'
Leakage	0	1
password1	1	X
password2	0	0
password3	0	1
password4	1	X
...	...	...
passwordn	0	0

Additional vulnerability (found after the paper submission)

HuntingAndPecking(*pwd*, *A*, *B*, *k*)

```
found, i = false, 1
while not found or i < k :
    seed = Hash(A, B, pwd, i)
    x_cand = KDF(seed, label)
    if x_cand is a point's coordinate :
        if not found :
            found, x, seed_x = true, x_cand, seed
    i = i + 1
y = set_compressed_point_coordinate(x, seed_x)
return (x, y)
```

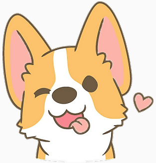
← : leaks the seed's parity

Underlying crypto library call

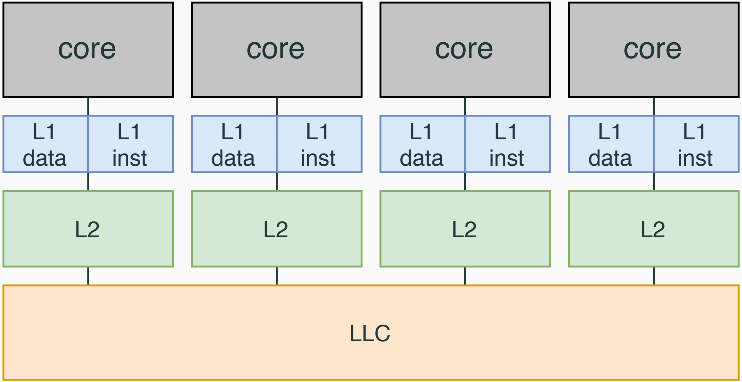
Future / Current work

- Find / adapt tools to perform thorough analysis of WPA3
  - Complete/Sound tools do not scale well
  - Scalable tools are (often) not complete
- Analyze various implementations
- Patch remaining vulnerabilities
- Enjoy secure WPA3 implementations

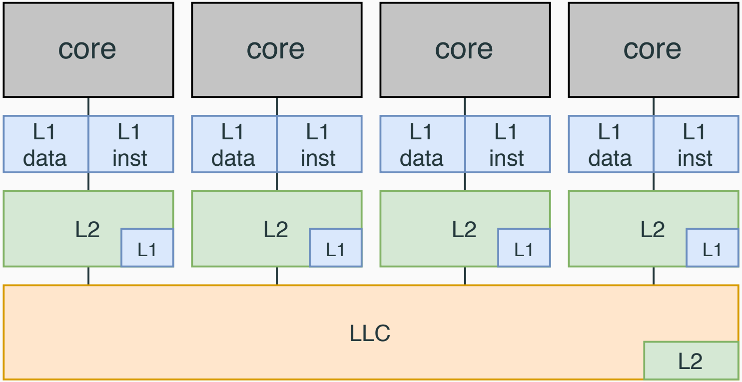
Questions?



Inclusive CPU cache

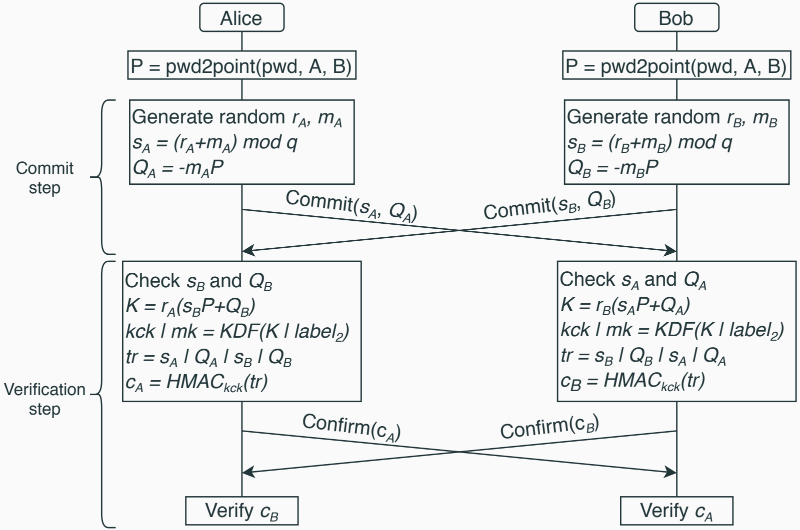


Inclusive CPU cache



Inclusive cache

Dragonfly workflow



Is (x, y) a point on a curve ?

Need to check if $x^3 + ax + b$ is a a quadratic residue on $\mathbb{F}_p$

is_x_on_curve(x)

```
y_sqr = x^3 + ax + b
return legendre_symbol(y_sqr, p) == 1
```

Is (x, y) a point on a curve ?

Need to check if $x^3 + ax + b$ is a a quadratic residue on $\mathbb{F}_p$

is_x_on_curve(x, qr, nqr)

```
mask = get_random()
y_sqr = x^3 + ax + b
blind_sqr = y_sqr × mask^2
if mask is odd :
    blind_sqr = blind_sqr × qr
    return legendre_symbol(blind_sqr) == −1
else
    blind_sqr = blind_sqr × nqr
    return legendre_symbol(blind_sqr) == 1
```