The Mummy 2018 - Microsoft Summons Back Ugly Attacks From The Past

Who am I

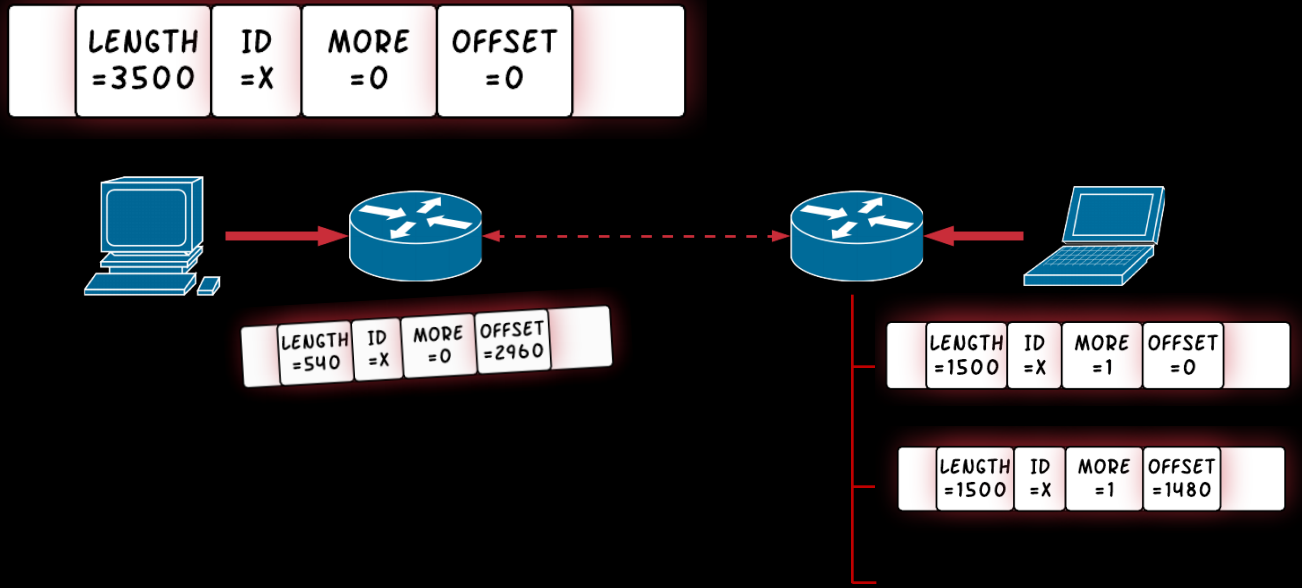
- Ran Menscher
  - Israel
- Independent Software Researcher
  - Reverse Engineering
    - OS internals, Embedded, Applications…
- Past: VP Research, XM Cyber 

  - Vulnerabilities
    - Yes ☺

I’m going to tell you about

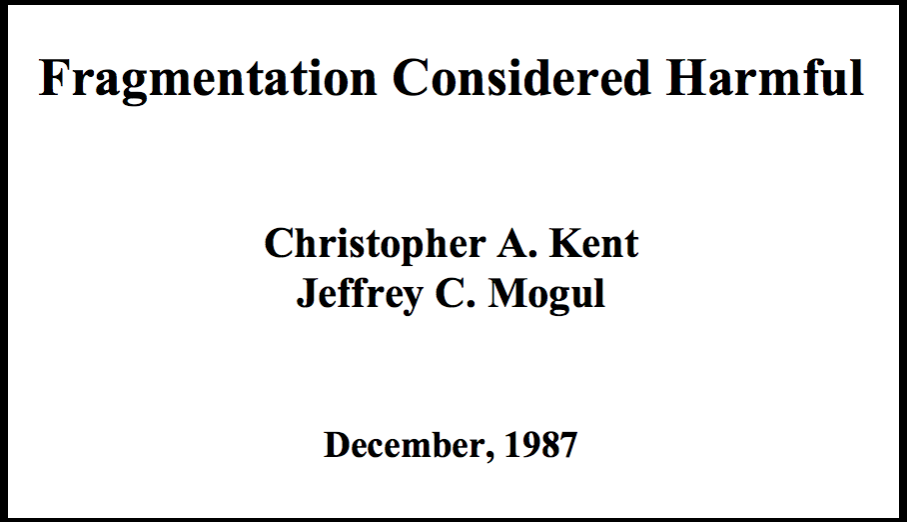
- An unusual bug in Windows IP stack
- Fragmentation and IP ID randomization
  - Overview, past attacks
  - The bug (CVE-2018-8493)
  - Exploitation
- Other cool consequences

Fragmentation and Reassembly



Undeniably Cursed

Undeniably Cursed

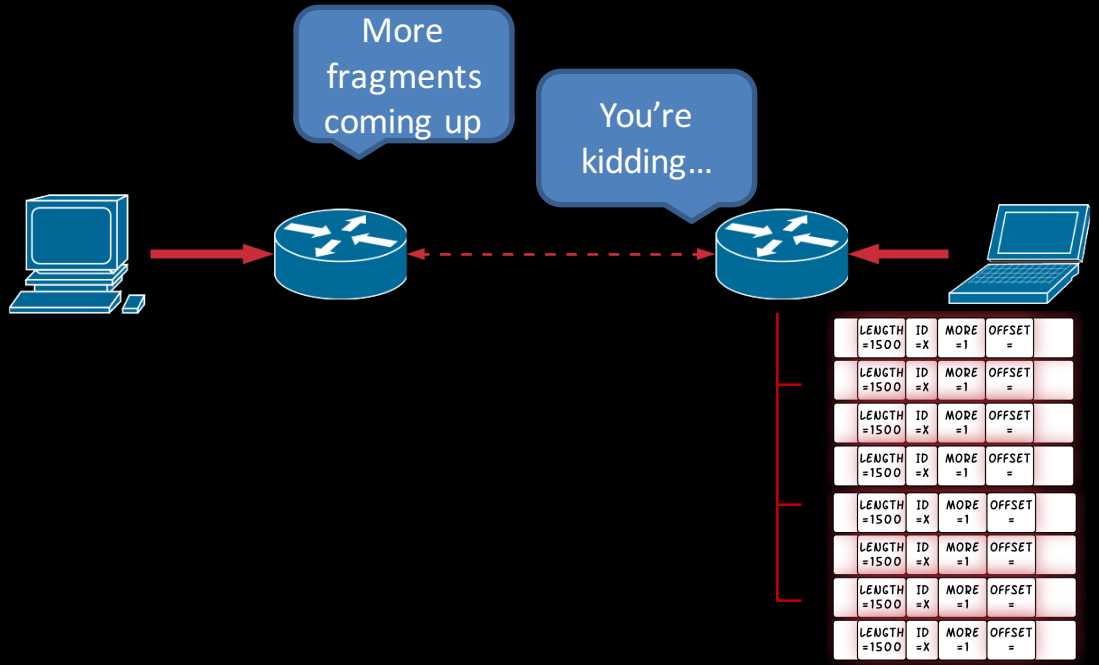


Undeniably Cursed

- Reassembly sensitive to resource exhaustion / other DoS

Undeniably Cursed

- Reassem



Undeniably Cursed

- Reassembly sensitive to resource exhaustion / other DoS
- Lots of attack surface to evade IDS

Undeniably

- Reassembly
- Lots of atta

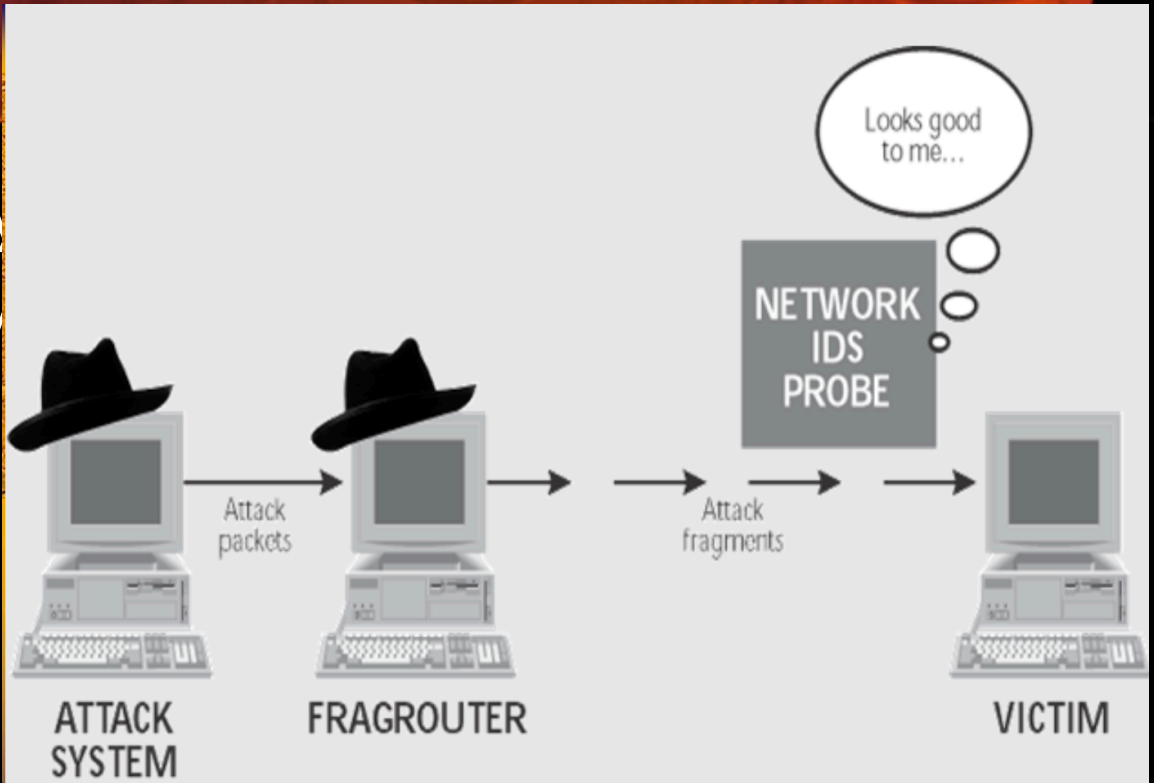


Figure 6.22 Using FragRouter to evade IDS detection

(source: online presentation by Tobias Renwick)

Undeniably Cursed

- Reassembly sensitive to resource exhaustion / other DoS
- Lots of attack surface to evade IDS
- Most Implementations: IP IDs as Global Counter

Curse of Global Counter

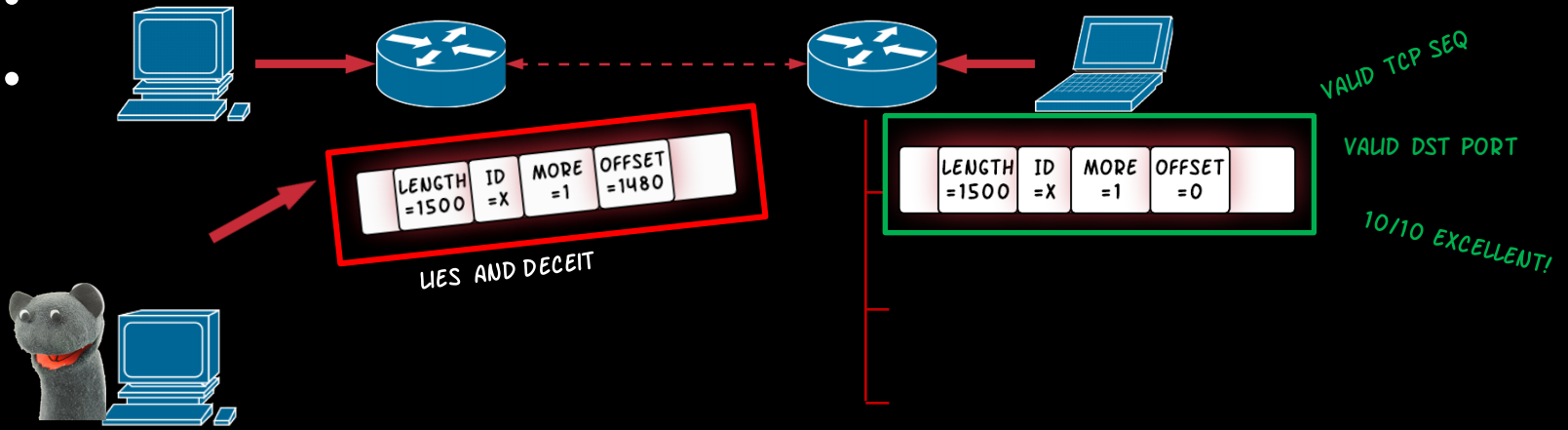
- DeNATing
- Idle Scanning

Curse of Global Counter

- DeNATing
- Idle Scanning
- Blind packet injection (Zalewski 03)

Curse of Global Counter

- DeNATing
-
-

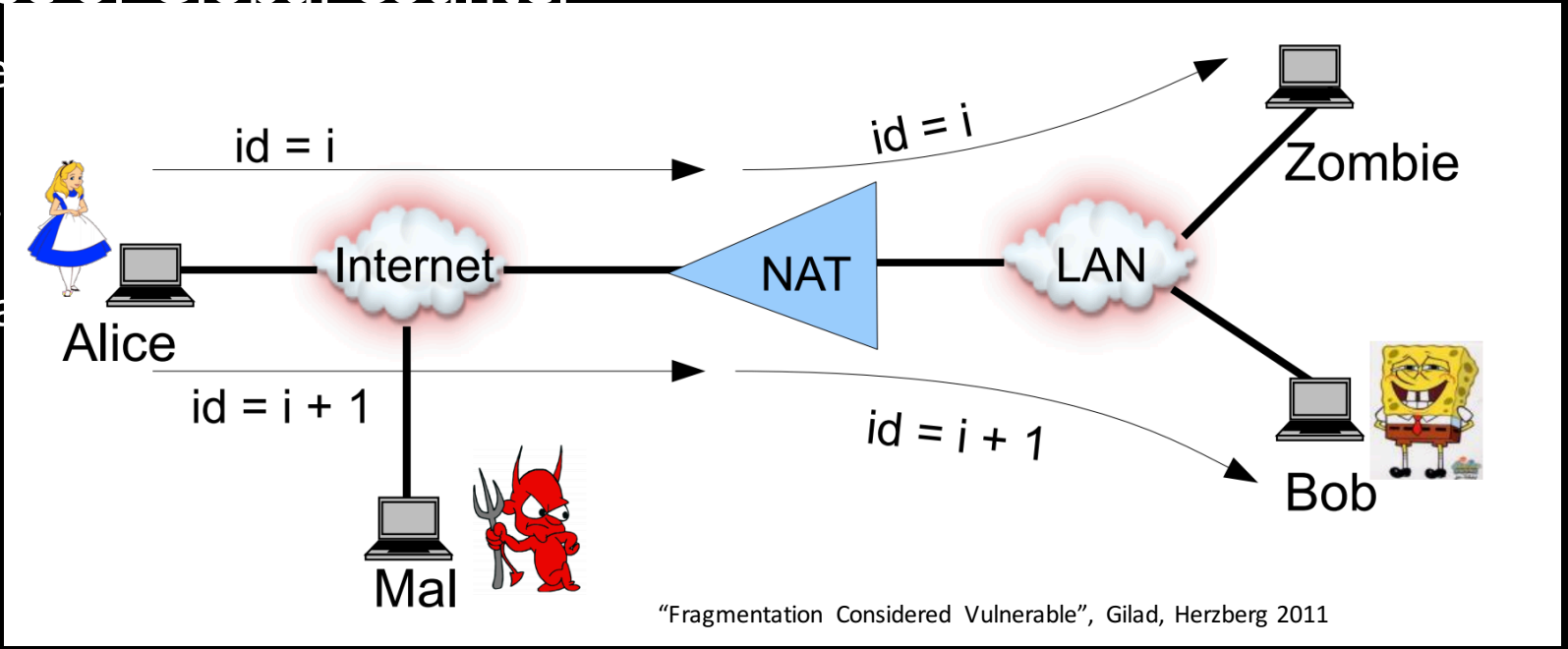


Curse of Global Counter

- DeNATing
- Idle Scanning
- Blind packet injection (Zalewski 03)
- Traffic interception by NAT/Tunnel (Gilad, Herzberg 11)

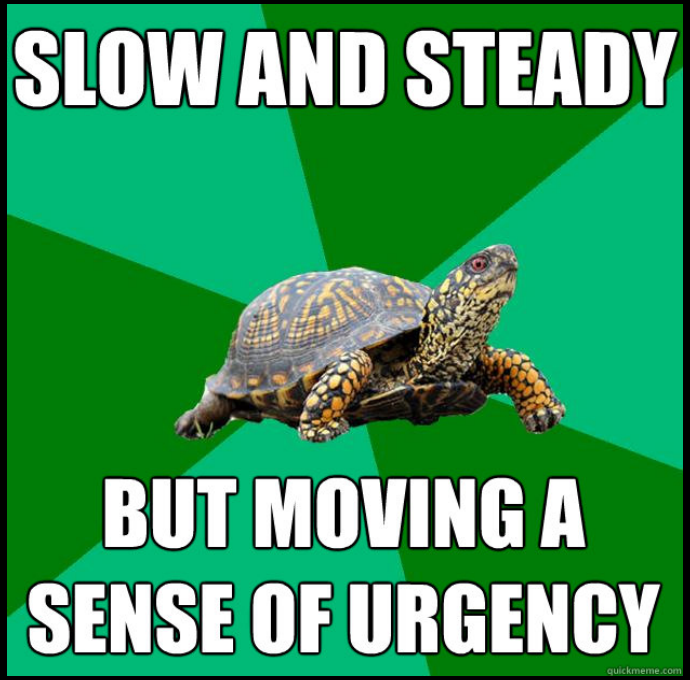
Curse of Global Counter

- De
- Idl
- Bli
- Tra



"Fragmentation Considered Vulnerable", Gilad, Herzberg 2011

So the vendors were quick to seal the curse



So the vendors were quick to seal the curse

- Global Counter in Windows until 2012 (per interface)
- windows 8
- Different IP ID per IP path
- And they were safe and happy

For 8.1, a “major” refactor had taken place for IP IDs:

Most prominent changes:

- A function isn’t inline’d anymore
  - (but that could be the compiler)
- An array was changed to a pointer
- Why did they change it?

IP ID GENERATION

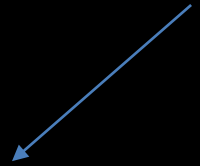
- Is about **IP PATH**

- Is about **IP PATH**

IDENTIFICATION = BASE + INCREMENT

- Is about **IP PATH**

IDENTIFICATION = BASE + INCREMENT



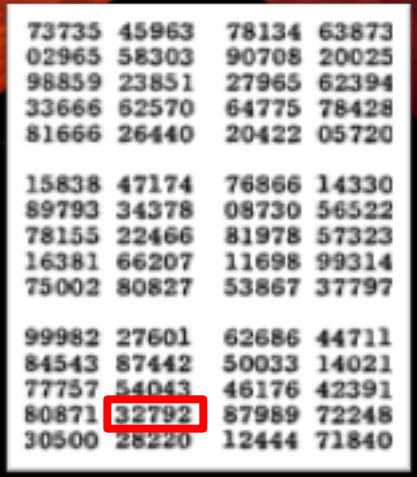
Random 4 bytes (init @boot)

⊕

hash(KEY, IP PATH)

- Is about **IP PATH**

IDENTIFICATION = BASE + INCREMENT



Random 4 bytes (init @boot)

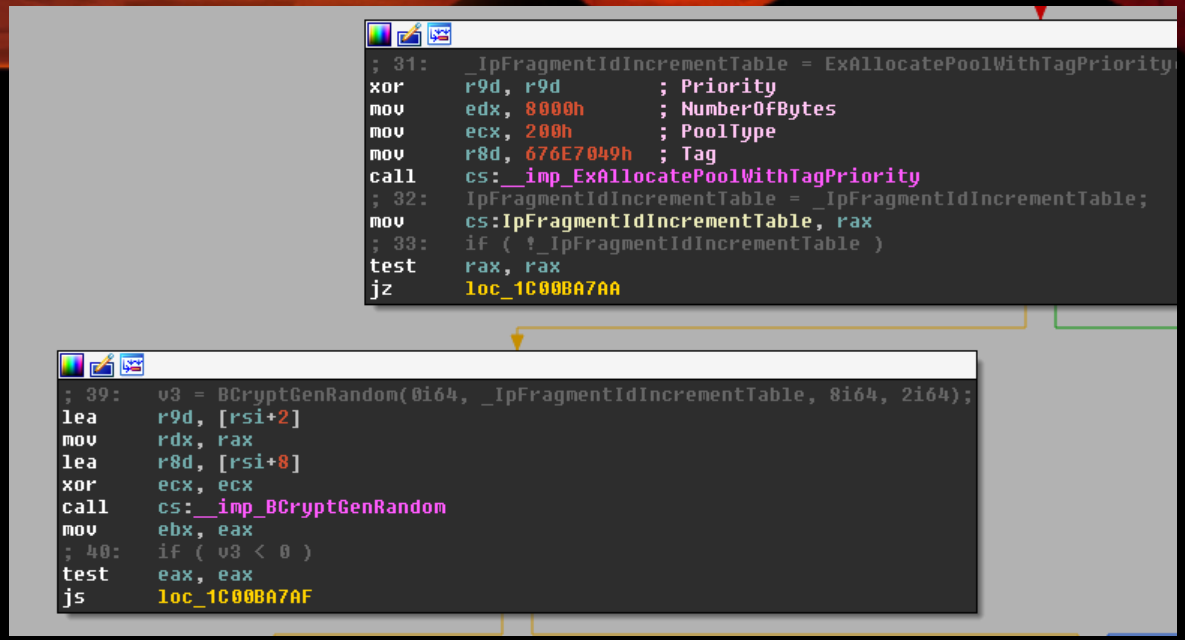
⊕

hash(KEY, IP PATH)

increments[hash(KEY, IP PATH)]

Oops

- Allocate 0x8000
- Initialize 8 … bytes
- Sizeof(int *)
- Mostly zeros



Oops

- Allocate 0x8000
- Initialize 8 … bytes
- Sizeof(int *)
- Mostly zeros

```
; 31:   _IpFragmentIdIncrementTable = ExAllocatePoolWithTagPriority
xor     r9d, r9d        ; Priority
mov     edx, 8000h      ; NumberOfBytes
mov     ecx, 200h       ; PoolType
mov     r8d, 676E7049h  ; Tag
call    cs:__imp_ExAllocatePoolWithTagPriority
; 32:   IpFragmentIdIncrementTable = _IpFragmentIdIncrementTable;
mov     cs:IpFragmentIdIncrementTable, rax
```

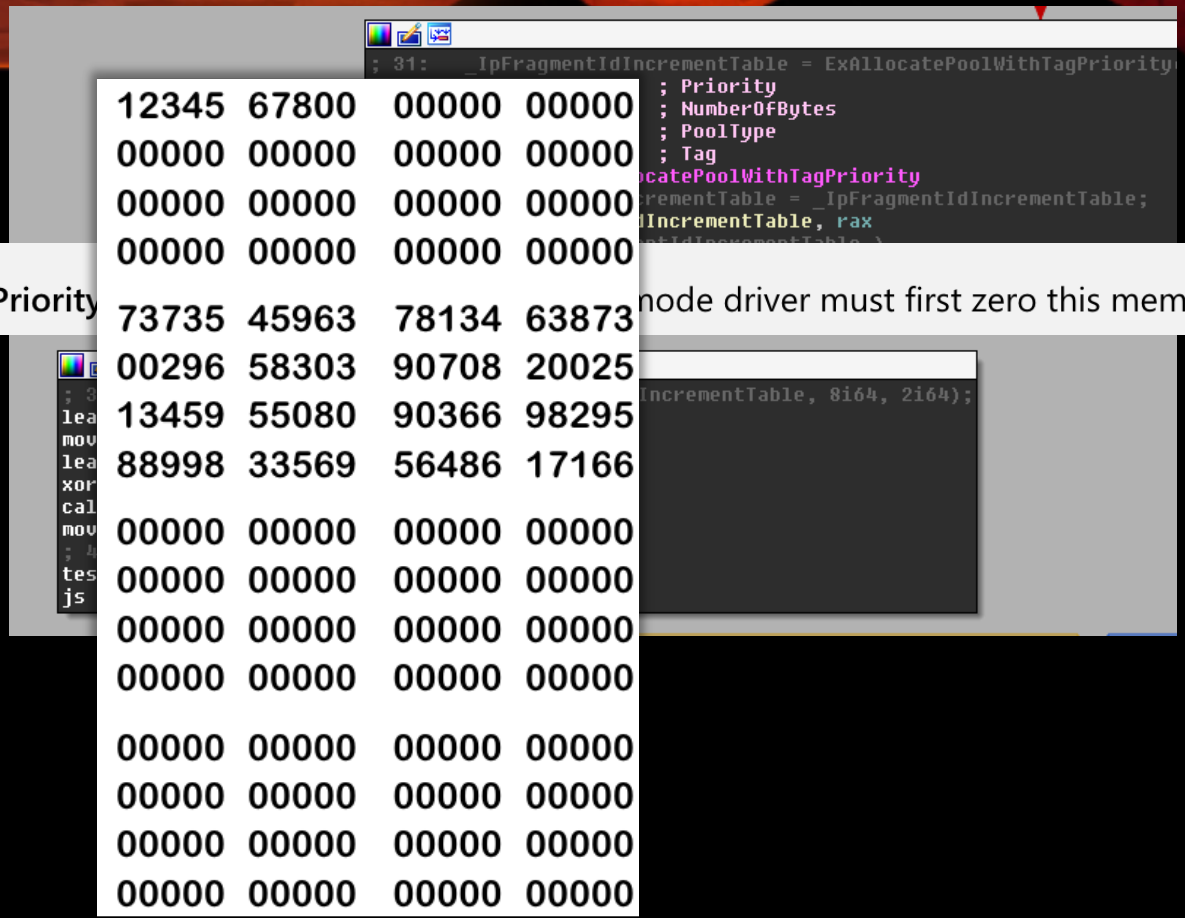
Note Memory that **ExAllocatePoolWithTagPriority** allocates is uninitialized. A kernel-mode driver must first zero this memory

```
; 39:   v3 = BCryptGenRandom(0i64, _IpFragmentIdIncrementTable, 8i64, 2i64);
lea     r9d, [rsi+2]
mov     rdx, rax
lea     r8d, [rsi+8]
xor     ecx, ecx
call    cs:__imp_BCryptGenRandom
mov     ebx, eax
; 40:   if ( v3 < 0 )
test    eax, eax
js      loc_1C00BA7AF
```

Oops

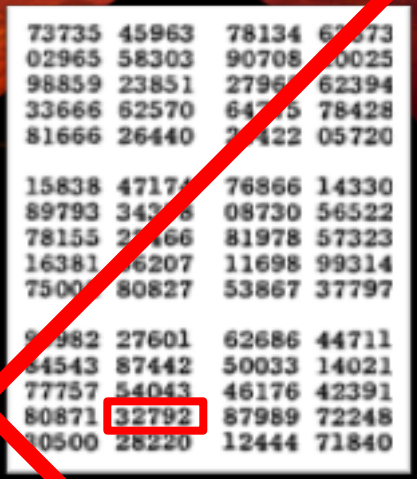
Note Memory that **ExAllocatePoolWithTagPriority**

node driver must first zero this memory



- Is about IP PATH

IDENTIFICATION = BASE ~~+ INCREMENT~~



Random 4 bytes (init @boot)

⊕

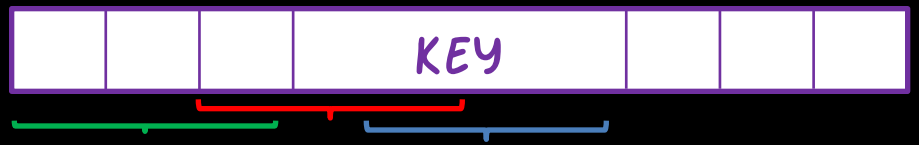
hash(KEY, IP PATH)

increments[hash(KEY, IP PATH)]

KEY is 40 random bytes

hash is a Toeplitz hash (RSS)

Toeplitz matrices



00000	45678	67456	78674
20384	09234	93759	12987
47823	28002	23532	75930
66783	48759	28465	93732

,

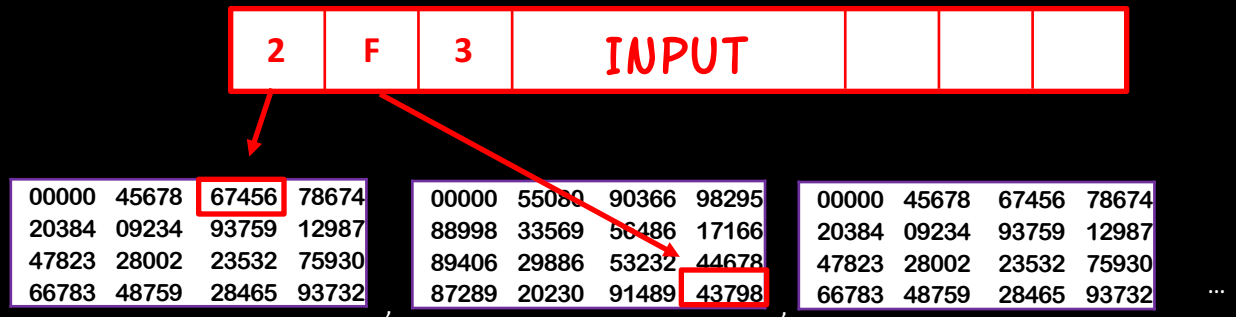
00000	55080	90366	98295
88998	33569	56486	17166
89406	29886	53232	44678
87289	20230	91489	43798

...

KEY is 40 random bytes

hash is a Toeplitz hash

Toeplitz matrices



Hash = tbl_1[0x2] ⊕ tbl_2[0xF] ⊕ ...

KEY is

hash is

Toeplit

0	rol(key[i],3)	rol(key[i],2)	rol(key[i],3) ⊕ rol(key[i],2)
rol(key[i],1)	rol(key[i],3) ⊕ rol(key[i],1)	rol(key[i],2) ⊕ rol(key[i],1)	rol(key[i],3) ⊕ rol(key[i],2) ⊕ rol(key[i],1)
rol(key[i],0)	rol(key[i],3) ⊕ rol(key[i],0)	rol(key[i],2) ⊕ rol(key[i],0)	rol(key[i],3) ⊕ rol(key[i],2) ⊕ rol(key[i],0)
rol(key[i],1) ⊕ rol(key[i],0)	rol(key[i],3) ⊕ rol(key[i],1) ⊕ rol(key[i],0)	rol(key[i],2) ⊕ rol(key[i],1) ⊕ rol(key[i],0)	rol(key[i],3) ⊕ rol(key[i],2) ⊕ rol(key[i],1) ⊕ rol(key[i],0)

00000	45678	67456	78674
20384	09234	93759	12987
47823	28002	23532	75930
66783	48759	28465	93732

,

00000	55089	90366	98295
88998	33569	56486	17166
89406	29886	53232	44678
87289	20230	91489	43798

,

00000	45678	67456	78674
20384	09234	93759	12987
47823	28002	23532	75930
66783	48759	28465	93732

...

Hash = tbl_1[0x2] ⊕ tbl_2[0xF] ⊕ ...

KEY is

hash is

Toeplit

0	rol(key[i],3)	rol(key[i],2)	rol(key[i],3) ⊕ rol(key[i],2)
rol(key[i],1)	rol(key[i],3) ⊕ rol(key[i],1)	rol(key[i],2) ⊕ rol(key[i],1)	rol(key[i],3) ⊕ rol(key[i],2) ⊕ rol(key[i],1)
rol(key[i],0)	rol(key[i],3) ⊕ rol(key[i],0)	rol(key[i],2) ⊕ rol(key[i],0)	rol(key[i],3) ⊕ rol(key[i],2) ⊕ rol(key[i],0)
rol(key[i],1) ⊕ rol(key[i],0)	rol(key[i],3) ⊕ rol(key[i],1) ⊕ rol(key[i],0)	rol(key[i],2) ⊕ rol(key[i],1) ⊕ rol(key[i],0)	rol(key[i],3) ⊕ rol(key[i],2) ⊕ rol(key[i],1) ⊕ rol(key[i],0)

00000	45678	67456	78674
20384	09234	93759	12987
47823	28002	23532	75930
66783	48759	28465	93732

,

00000	55089	90366	98295
88998	33569	56486	17166
89406	29886	53232	44678
87289	20230	91489	43798

,

00000	45678	67456	78674
20384	09234	93759	12987
47823	28002	23532	75930
66783	48759	28465	93732

...

Hash = tbl_1[0x2] ⊕ tbl_2[0xF] ⊕ ...

Nibbles of input, that are XOR 8 of each other –

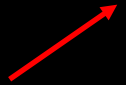
Their hashes are XOR KEY[i] of each other!

Hash = tbl_1[0x2] ⊕ tbl_2[0xF] ⊕ …

Nibbles of input, that are XOR 8 of each other –

Their hashes are XOR KEY[i] of each other!

Inputs that differ only by a nibble will output a cell's content!



$Hash = tbl_1[\ 0x2\ ] \oplus tbl_2[\ 0xF\ ] \oplus \ldots$

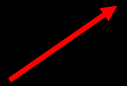
Nibbles of input, that are XOR 8 of each other –

Their hashes are XOR KEY[i] of each other!

Hash(10.0.0.1,10.0.0.2) = 1234 ⊕ 5453 ⊕ … ⊕ 0

Hash(0x80|10.0.0.1,10.0.0.2) = 1234 ⊕ 5453 ⊕ … ⊕ key[i]

Inputs that differ only by a nibble will output a cell's content!



Hash = tbl_1[0x2] ⊕ tbl_2[0xF] ⊕ …

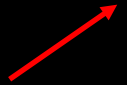
Nibbles of input, that are XOR 8 of each other –

Their hashes are XOR KEY[i] of each other!

$ID(10.0.0.1,10.0.0.2) = 1234 \oplus 5453 \oplus \dots \oplus 0 \oplus secret$

$ID(0x80|10.0.0.1,10.0.0.2) = 1234 \oplus 5453 \oplus \dots \oplus key[i] \oplus secret$

Inputs that differ only by a nibble will output a cell's content!



$Hash = tbl_1[\ 0x2\ ] \oplus tbl_2[\ 0xF\ ] \oplus \dots$

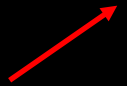
Nibbles of input, that are XOR 8 of each other –

Their hashes are XOR KEY[i] of each other!

ID(10.0.0.1,10.0.0.2) = ~~1234 ⊕ 5453 ⊕~~ … ⊕ 0 ~~⊕ secret~~

ID(0x80|10.0.0.1,10.0.0.2) = ~~1234 ⊕ 5453 ⊕~~ … ⊕ key[i] ~~⊕ secret~~

Inputs that differ only by a nibble will output a cell's content!



Hash = tbl_1[0x2] ⊕ tbl_2[0xF] ⊕ …

Nibbles of input, that are XOR 8 of each other –

Their hashes are XOR KEY[i] of each other!

ID(10.0.0.1,10.0.0.2) = ~~1234 ⊕ 5453 ⊕~~ … ⊕ 0 ~~⊕ secret~~

ID(0x80|10.0.0.1,10.0.0.2) = ~~1234 ⊕ 5453 ⊕~~ … ⊕ key[i] ~~⊕ secret~~

ID1 ⊕ ID2 = KEY[i]

ATTACK (key recovery):

- Get two samples of IP IDs
- For IP PATHs that differ by a nibble. XOR 8 of each other.
- Key[0] = ID_1 ^ ID_2 (if we hit increment=0)
- Repeat until confident of key[0]
- Repeat for other key parts

$$IDENTIFICATION_1 = KEY[i] \oplus IDENTIFICATION_2$$

$$\overbrace{IDENTIFICATION_1}^{\text{If increment}_1 == 0} = KEY[i] \oplus \overbrace{IDENTIFICATION_2}^{\text{If increment}_2 == 0}$$

- But if increment≠0
- We can deduce content from the table (=uninitialized mem)

ATTACK (reading kernel mem)

- Choose IP ID for IP PATHs known to have increment=0
- Use recovered key to initialize Toeplitz matrix values
- Get IP IDs for IP PATHs differing by a nibble from chosen IP PATH
- Calculate expected IP IDs according to matrix
- *SAMPLE - EXPECTED* = Table content = uninitialized mem

DEMO

Predicting IP IDs

- When increment=0, prediction is practical
- Works similarly to the memory read
- Problem reduced to assessing # of packets sent

Take Aways

- DontFragment (DF) is not just an IP flag. it’s good advice.
- Yes, Coders who refactor working code are grave robbers.
- If you mix performance and security, a simple bug will bring you down.

Questions?

Ran Menscher
ran@menschers.com
Twitter: @menscherr