




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

Who am I

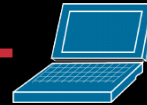
- Ran Menscher
 - Israel
- Independent Software Researcher
 - Reverse Engineering
 - OS internals, Embedded, Applications...
- Past: VP Research, XM Cyber 
 - Vulnerabilities
 - Yes 😊

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

LENGTH	ID	MORE	OFFSET
=3500	=X	=0	=0



LENGTH	ID	MORE	OFFSET
=540	=X	=0	=2960

LENGTH	ID	MORE	OFFSET
=1500	=X	=1	=0

LENGTH	ID	MORE	OFFSET
=1500	=X	=1	=1480



Undeniably Cursed



Undeniably Cursed

Fragmentation Considered Harmful

**Christopher A. Kent
Jeffrey C. Mogul**

December, 1987



Undeniably Cursed

- Reassembly sensitive to resource exhaustion / other DoS



Undeniably Cursed

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

Undeniably

- Reassembly
- Lots of attack

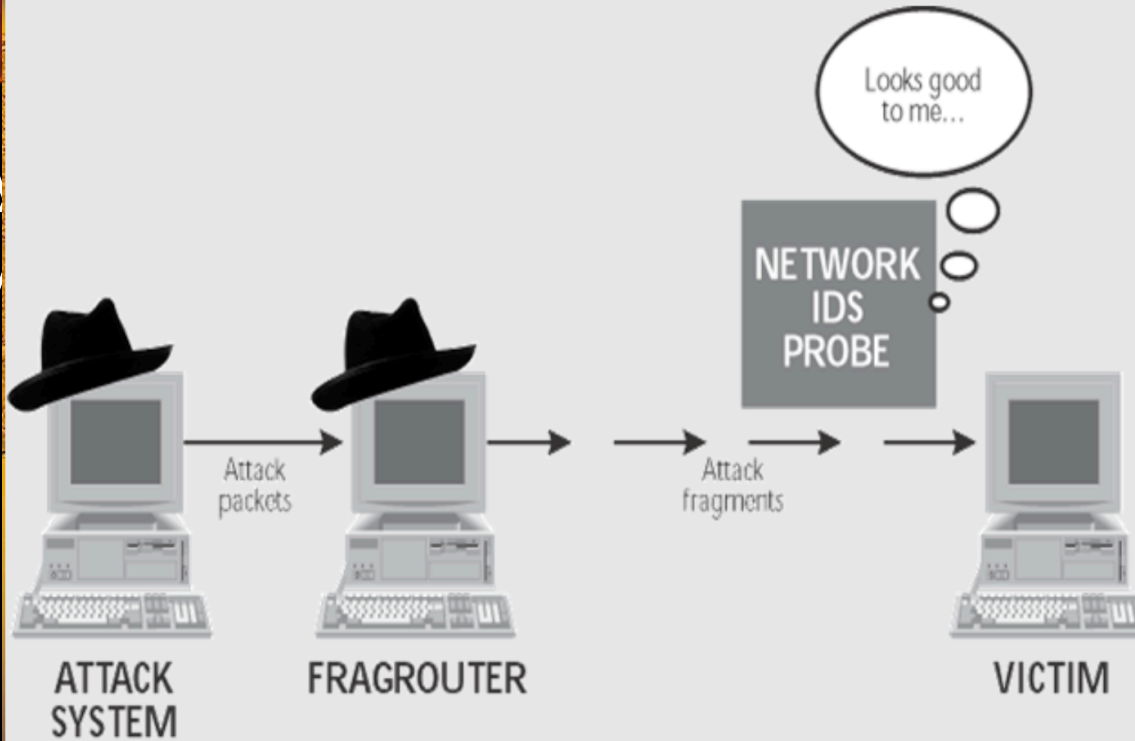


Figure 6.22 Using FragRouter to evade IDS detection

(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



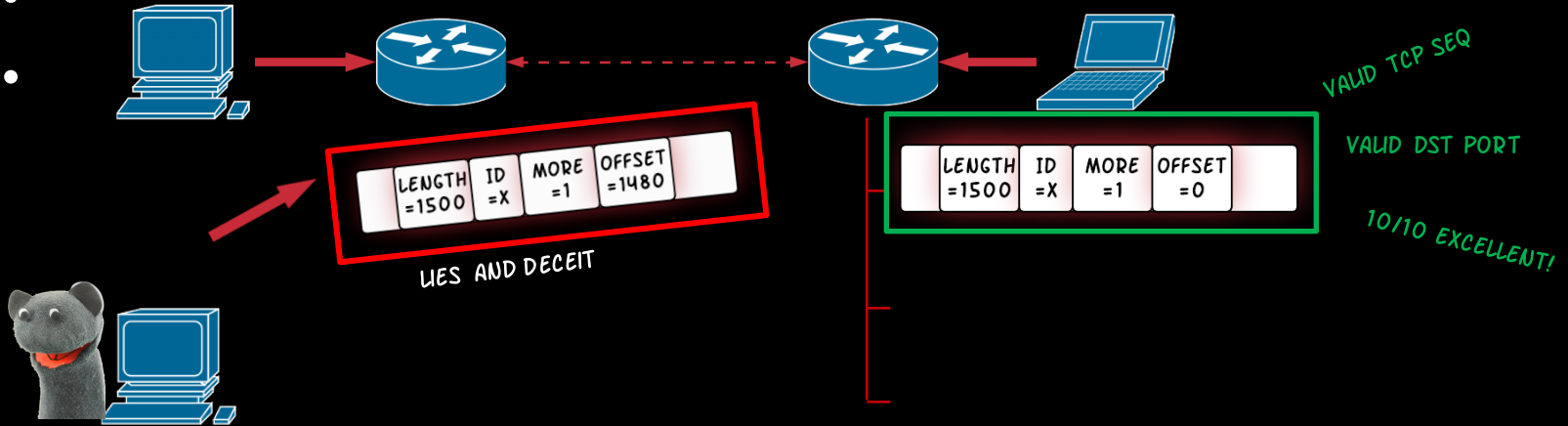
Curse of Global Counter

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

Curse of Global Counter

- DeNATing

-
-



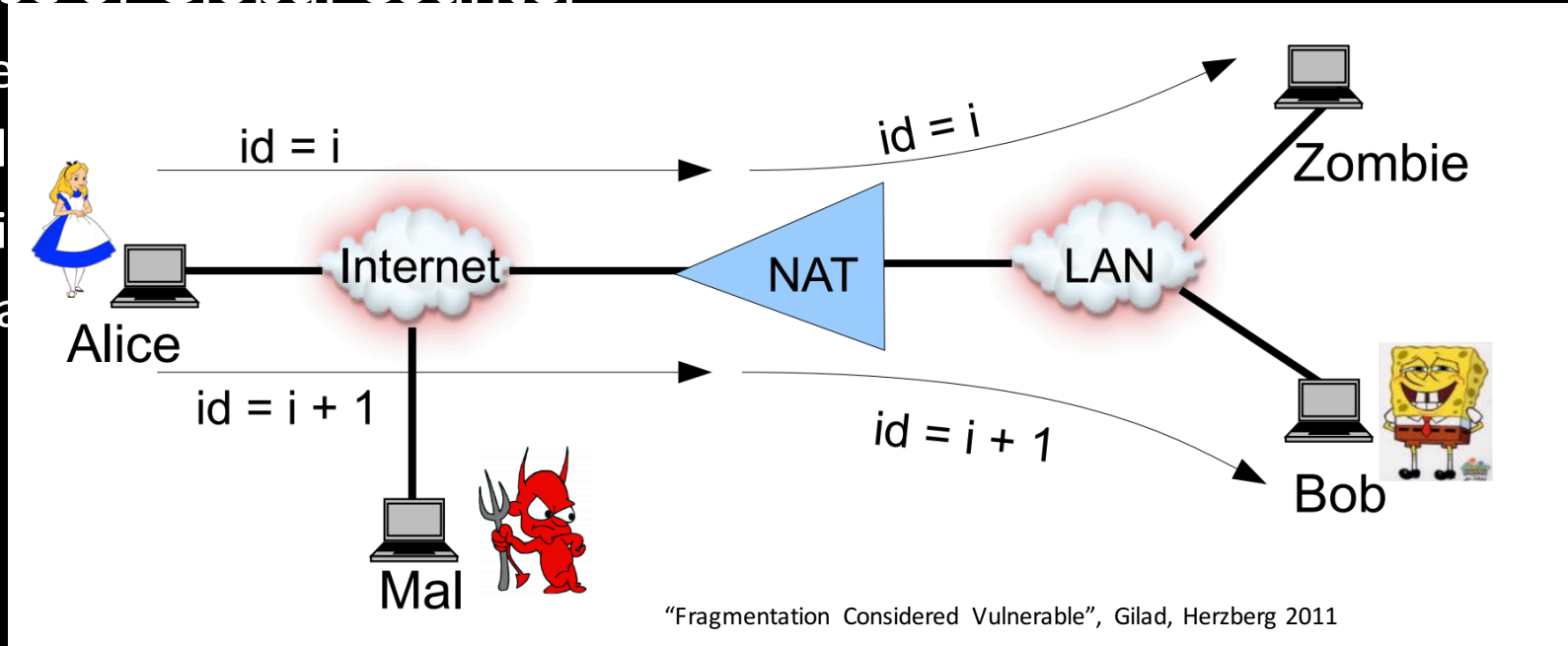


Curse of Global Counter

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

Curse of Global Counter

- De
- Idl
- Bli
- Tra



So the vendors were quick to seal the curse

SLOW AND STEADY



**BUT MOVING A
SENSE OF URGENCY**



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

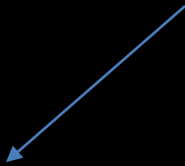
- 
- Is about IP PATH

- 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

73735	45963	78134	63873
02965	58303	90708	20025
98859	23851	27965	62394
33666	62570	64775	78428
81666	26440	20422	05720
15838	47174	76866	14330
89793	34378	08730	56522
78155	22466	81978	57323
16381	66207	11698	99314
75002	80827	53867	37797
99982	27601	62686	44711
84543	87442	50033	14021
77757	54043	46176	42391
80871	32792	87989	72248
30500	28220	12444	71840

Random 4 bytes (init @boot)

\oplus

hash(KEY, IP PATH)

increments[hash(KEY, IP PATH)]

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  
; 33:  if ( !_IpFragmentIdIncrementTable )  
test  rax, rax  
jz    loc_1C00BA7AA
```

```
; 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

```
; 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  
; 33:  ;
```

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

• Allocate 0x8000

- Initialize 8 ... bytes
- `sizeof(int *)`
- Mostly zeros

```
; 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

```
31: _IpFragmentIdIncrementTable = ExAllocatePoolWithTagPriority
; Priority
; NumberOfBytes
; PoolType
; Tag
; ExAllocatePoolWithTagPriority
crementTable = _IpFragmentIdIncrementTable;
IncrementTable, rax
atIdIncrementTable);
```

12345	67800	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
73735	45963	78134	63873
00296	58303	90708	20025
13459	55080	90366	98295
88998	33569	56486	17166
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000
00000	00000	00000	00000

```
lea
mov
lea
xor
cal
mov
tes
js
```

```
IncrementTable, 8i64, 2i64);
```

- Is about IP PATH

~~IDENTIFICATION = BASE + INCREMENT~~

73735	45963	78134	62873
02965	58303	90708	00025
98859	23851	27965	62394
33666	62570	64775	78428
81666	26440	27422	05720
15838	47174	76866	14330
89793	34378	08730	56522
78155	27466	81978	57323
16381	76207	11698	99314
75000	80827	53867	37797
87982	27601	62686	44711
84543	87442	50033	14021
77757	54043	46176	42391
80871	32792	87989	72248
80500	28220	12444	71840

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

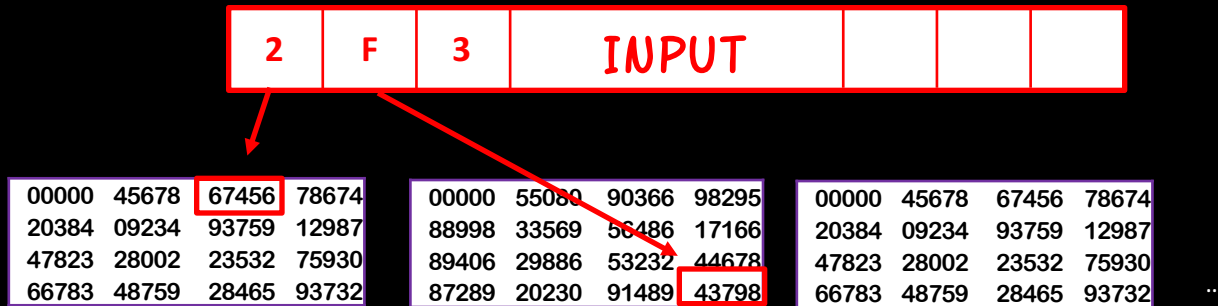
00000	55080	90366	98295
88998	33569	56486	17166
89406	29886	53232	44678
87289	20230	91489	43798

...

KEY is 40 random bytes

hash is a Toeplitz hash

Toeplitz matrices



$$\text{Hash} = \text{tbl}_1[0x2] \oplus \text{tbl}_2[0xF] \oplus \dots$$

KEY is
hash is
Toeplitz

0	$\text{rol}(\text{key}[i],3)$	$\text{rol}(\text{key}[i],2)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],2)$
$\text{rol}(\text{key}[i],1)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],1)$	$\text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],1)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],1)$
$\text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],0)$
$\text{rol}(\text{key}[i],1) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],1) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],1) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],1) \oplus \text{rol}(\text{key}[i],0)$

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

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

...

$$\text{Hash} = \text{tbl}_1[0x2] \oplus \text{tbl}_2[0xF] \oplus \dots$$

KEY is
hash is
Toeplitz

0	$\text{rol}(\text{key}[i],3)$	$\text{rol}(\text{key}[i],2)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],2)$
$\text{rol}(\text{key}[i],1)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],1)$	$\text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],1)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],1)$
$\text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],0)$
$\text{rol}(\text{key}[i],1) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],1) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],1) \oplus \text{rol}(\text{key}[i],0)$	$\text{rol}(\text{key}[i],3) \oplus \text{rol}(\text{key}[i],2) \oplus \text{rol}(\text{key}[i],1) \oplus \text{rol}(\text{key}[i],0)$

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

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

...

$$\text{Hash} = \text{tbl}_1[0x2] \oplus \text{tbl}_2[0xF] \oplus \dots$$



Nibbles of input, that are XOR 8 of each other –
Their hashes are XOR *KEY*[i] of each other!

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!

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

Hash = $\text{tbl}_1[0x2] \oplus \text{tbl}_2[0xF] \oplus \dots$



Nibbles of input, that are XOR 8 of each other –
Their hashes are XOR *KEY*[i] of each other!

$$\text{Hash}(10.0.0.1, 10.0.0.2) = 1234 \oplus 5453 \oplus \dots \oplus 0$$

$$\text{Hash}(0x80 | 10.0.0.1, 10.0.0.2) = 1234 \oplus 5453 \oplus \dots \oplus \text{key}[i]$$

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


$$\text{Hash} = \text{tbl}_1[0x2] \oplus \text{tbl}_2[0xF] \oplus \dots$$

Nibbles of input, that are XOR 8 of each other –
Their hashes are XOR *KEY*[i] of each other!

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

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

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


$$\text{Hash} = \text{tbl}_1[0x2] \oplus \text{tbl}_2[0xF] \oplus \dots$$

Nibbles of input, that are XOR 8 of each other –
Their hashes are XOR *KEY*[i] of each other!

$$\begin{aligned} ID(10.0.0.1, 10.0.0.2) &= \cancel{1234} \oplus \cancel{5453} \oplus \dots \oplus 0 \oplus \cancel{\text{secret}} \\ ID(0x80 | 10.0.0.1, 10.0.0.2) &= \cancel{1234} \oplus \cancel{5453} \oplus \dots \oplus \text{key}[i] \oplus \cancel{\text{secret}} \end{aligned}$$

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


$$\text{Hash} = \text{tbl}_1[0x2] \oplus \text{tbl}_2[0xF] \oplus \dots$$

Nibbles of input, that are XOR 8 of each other –
Their hashes are XOR *KEY*[i] of each other!

$$\begin{aligned} ID(10.0.0.1, 10.0.0.2) &= \cancel{1234} \oplus \cancel{5453} \oplus \dots \oplus 0 && \oplus \cancel{\text{secret}} \\ ID(0x80 | 10.0.0.1, 10.0.0.2) &= \cancel{1234} \oplus \cancel{5453} \oplus \dots \oplus \text{key}[i] && \oplus \cancel{\text{secret}} \end{aligned}$$

$$ID1 \oplus ID2 = \text{KEY}[i]$$



ATTACK (key recovery):

- Get two samples of IP IDs
- For IP PATHs that differ by a nibble. XOR 8 of each other.
- $\text{Key}[0] = \text{ID}_1 \wedge \text{ID}_2$ (if we hit increment=0)
- Repeat until confident of key[0]
- Repeat for other key parts

The background features a dark red and orange gradient at the top, overlaid with faint, stylized Egyptian hieroglyphs. Below this, the silhouettes of two pyramids are visible against a lighter orange glow. The rest of the background is solid black.
$$\text{IDENTIFICATION}_1 = \text{KEY}[i] \oplus \text{IDENTIFICATION}_2$$



If increment₁ == 0 If increment₂ == 0

$$\text{IDENTIFICATION}_1 = \text{KEY}[i] \oplus \text{IDENTIFICATION}_2$$

- But if increment₁ ≠ 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

The background features a dark red and orange gradient. At the top, there are faint, stylized Egyptian hieroglyphs. Below the hieroglyphs, two black silhouettes of pyramids are visible against the gradient. The word "DEMO" is written in white, bold, sans-serif capital letters on the left side of the image.

DEMO

The background of the slide features a dark red and orange gradient. At the top, there are faint, stylized Egyptian hieroglyphs. Below the hieroglyphs, two black silhouettes of pyramids are visible against the gradient background.

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