Behind the Black Mirror

Simulating attacks with mock C2 servers

Scott Knight
Threat Researcher / VMware SBU

September 2020

vmware

Who am I?

Scott Knight

- Threat Researcher on the Threat Analysis Unit (TAU) team at VMware
- Reverse engineer malware
- Track threat actors
- Share information back with the security community

Website: https://knight.sc

Github: github.com/knightsc

Twitter: @sdotknight

Agenda

What and Why

Mock C2 How To

C2 Protocol Characteristics

Simulating Attacks

Demo

Closing Thoughts

What and Why

What?

In programming, a mock object is one that simulates the behavior of a real object in order to facilitate software testing.

A mock C2 server can be thought of as a server that simulates the behavior of a real malware sample's C2 server in order to test and analyze the behavior of the malware.

Why?

Dynamic analysis of a sample can be challenging and misleading without a C2 responding. Only a small subset of the malware's capabilities are observed.

Reverse engineering a network protocol is hard without real network traffic.

Improved visibility into what the malware does in a real world attack.

Mock C2 How To

Step 1 – Pick a language

Scripting

- Python, Ruby, Node.js, etc.
- Pros
  - Quick to get something up and running
  - Easy to use on multiple platforms
- Cons
  - Working with binary data can be cumbersome

Compiled

- Go, Swift, Rust, C/C++, etc.
- Pros
  - Often better performance
  - Better concurrency support
- Cons
  - Building projects can be slower and error prone

Step 2 – Set up your network

Hardware

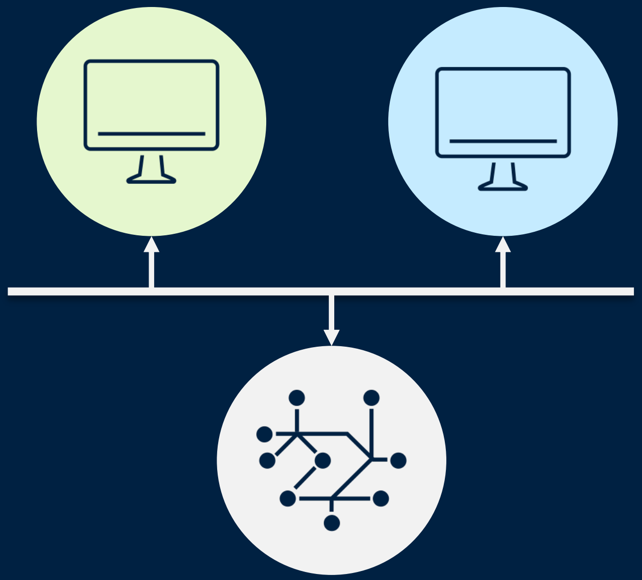
- One mock C2 server
- One detonation host
- Both machines on the same network
- Virtual machines and virtual networks make this easy

DNS

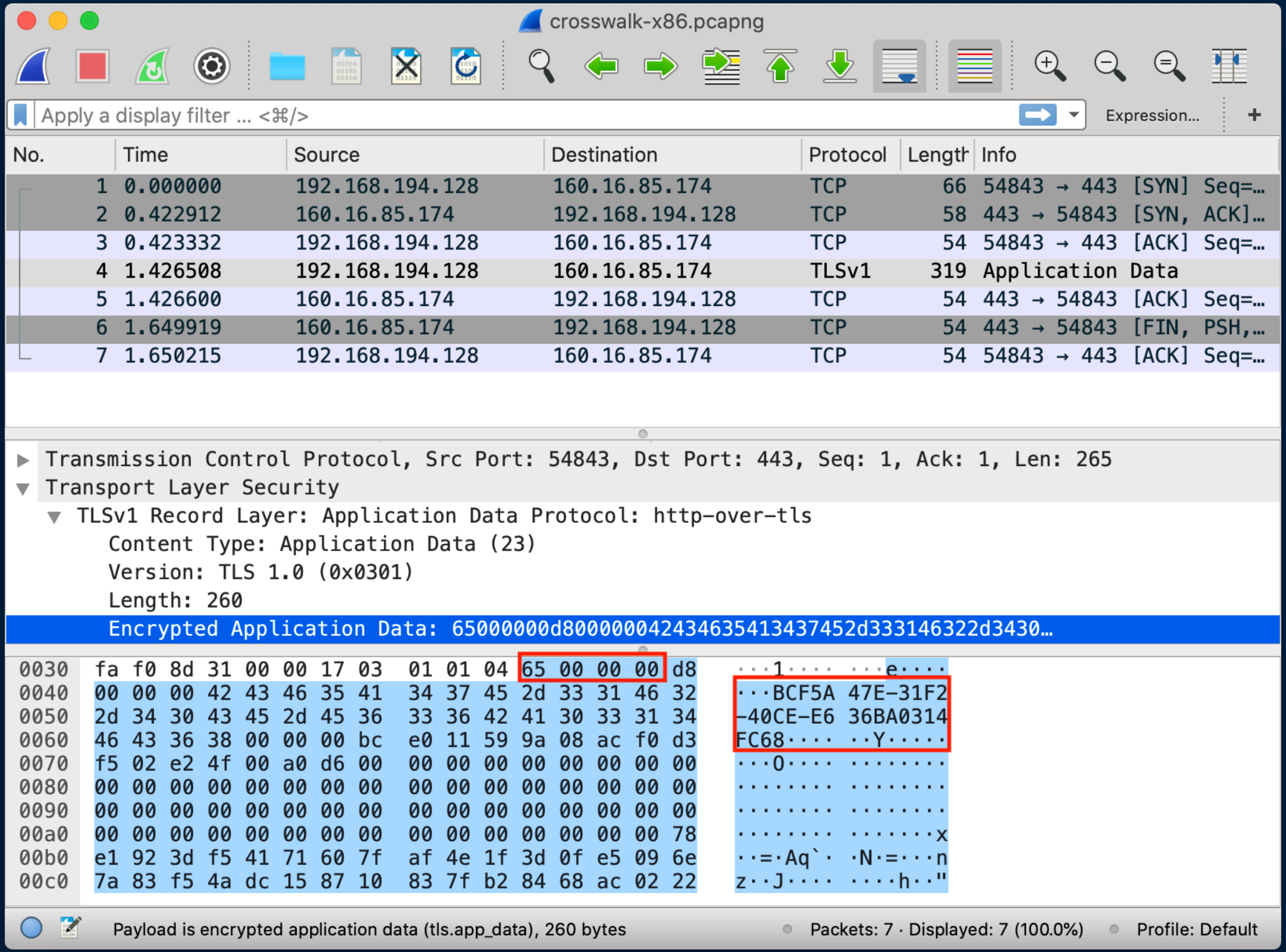
- Modify the hosts file on the detonation machine

IP

- Set mock C2 server to have an IP address of a real C2
- Set detonation host to have an IP address on the same subnet



Step 3 - Just listen



Start a server listening on the C2 port

Run Wireshark to capture traffic

See what shows up

Step 4 – Iterate!

Start dynamic analysis

- What format are commands sent in?
- Does the malware encrypt the data?
- Does the malware send a beacon?
- Does it expect a response?

Mock out responses to the malware

Test sending commands

Repeat

C2 Protocol Characteristics

Network and Encryption

First hurdle in understanding how malware communicates

What transport mechanism does it use?

- TCP
- UDP
- TLS
- Fake TLS
- HTTP

Is the network data encrypted?

- Encryption often obscures the real payloads being sent and received
- It's necessary to reverse in order to properly mock a C2 server
- There are some common patterns that show up often

Simple XOR

```
{
  v10 = 0;
  if ( len <= 0 )
    return v9;
  if ( (unsigned int)len >= 0x20 )
  {
    v11 = data + 16;
    do
    {
      v12 = *((__m128i *)v11 - 1);
      v10 += 32;
      v11 += 32;
      *((__m128i *)v11 - 3) = _mm_xor_si128(v12, (__m128i)xmmword_4CE0D0);
      *((__m128i *)v11 - 2) = _mm_xor_si128(*((__m128i *)v11 - 2), (__m128i)xmmword_4CE0D0);
    }
    while ( v10 < len - len % 32 );
  }
  for ( ; v10 < len; ++v10 )
    data[v10] ^= 0x77u;
  v13 = *v6;
  while ( 1 )
  {
    v9 = send(v13, data_1, v5, 0);
    v13 = *v6;
    if ( *v6 == -1 || v9 < 0 )
      break;
    v5 -= v9;
    data_1 += v9;
    if ( v5 <= 0 )
      return v9;
  }
```

Characteristics

- Very common
- Easy to spot in packet captures
- Easy to reverse

Malware Examples

- OSX.Yort
- BISTROMATH

Complex XOR

```
unsigned int __cdecl DecryptData(char *input, int length)
{
  int i; // esi
  int key2_lb; // edx
  char *data; // edi
  char key1; // cl
  unsigned int key3; // eax
  char v7; // bl
  unsigned int key2; // [esp+8h] [ebp-4h]

  i = 0;
  LOBYTE(key2_lb) = 0x8B;
  data = input;
  key1 = 0x17;
  key2 = 0xB8D68B;
  key3 = 0x2497029;
  if ( length > 0 )
  {
    while ( 1 )
    {
      data[i] ^= key1 ^ key3 ^ key2_lb;
      v7 = key2_lb & (key1 ^ key3);
      key2_lb = ((((unsigned __int16)key2 ^ (unsigned __int16)(8 * key2)) & 0x7F8) << 20) | (key2 >> 8);
      key1 = v7 ^ key3 & key1;
      ++i;
      key3 = (((key3 << 7) ^ (key3 ^ (16 * (key3 ^ (2 * key3)))) & 0xFFFFFF80) << 17) | (key3 >> 8);
      key2 = key2_lb;
      if ( i >= length )
        break;
      data = input;
    }
  }
  return key3;
}
```

Characteristics

- “Custom” encryption
- Key derivation can be harder to reverse
- Obscures network traffic more

Malware Examples

- HOTCROISSANT
- Rifdoor
- SLICKSHOES

RC4

```
rc4_state *__fastcall CMataNet_rc4_init(mata_net *mataNet, rc4_state *rc4state, __int64 key, int key_length)
{
  rc4_state *result; // rax
  unsigned __int8 v7; // [rsp+2Bh] [rbp-5h]
  int i; // [rsp+2Ch] [rbp-4h]
  int v9; // [rsp+2Ch] [rbp-4h]

  for ( i = 0; i <= 255; ++i )
    rc4state->sbox[i] = i;
  rc4state->i1 = 0;
  result = rc4state;
  rc4state->j1 = 0;
  v9 = 0;
  v7 = 0;
  while ( v9 <= 255 )
  {
    v7 += rc4state->sbox[v9] + *(_BYTE *)(v9 % key_length + key);
    result = (rc4_state *)CMataNet_swap_bytes(mataNet, &rc4state->sbox[v9++], &rc4state->sbox[v7]);
  }
  return result;
}
```

Characteristics

- Still common in malware
- Easy to spot when reversing

Malware Examples

- Dacls/MATA

AES

```
BOOL __fastcall DeriveKey(global_struct *global, int sessionKey, _DWORD *phHash, int data, int len, int a6)
{
  BOOL result; // eax

  if ( *phHash )
  {
    ((void (__stdcall *)(_DWORD))global->CryptDestroyHash)(*phHash);
    *phHash = 0;
  }
  if ( ((int (__stdcall *)(int, int, _DWORD, _DWORD, _DWORD *))global->CryptCreateHash)(
         global->cryptProvider,
         32771,                                // MD5
         0,
         0,
         phHash)
    && ((int (__stdcall *)(_DWORD, int, int, _DWORD))global->CryptHashData)(*phHash, data, len, 0) )
  {
    // Hash the data passed in with MD5 and then derive a AES-128 key
    result = ((int (__stdcall *)(int, int, _DWORD, int, int))global->CryptDeriveKey)(
               global->cryptProvider,
               26126,                          // CALG_AES_128
               *phHash,
               0x800000,
               sessionKey) != 0;
  }
  else
  {
    result = 0;
  }
  return result;
}
```

Characteristics

- Easy to spot when reversing
- Tends to use standard library/APIs
- Often comes along with more complex key derivation

Malware Examples

- CROSSWALK

Handshakes and Key Negotiation

```
{
  v3 = 0;
  if ( a2 )
  {
    if ( a3 )
    {
      v3 = 0;
      *a1 = socket(2, 1, 0);
      v8.sa_family = 2;
      *(_DWORD *)&v8.sa_data[2] = inet_addr(a2);
      *(_WORD *)v8.sa_data = __ROL2__(a3, 8);
      if ( !connect(*a1, &v8, 0x10u) )
      {
        if ( !(unsigned int)CMataNet_SSLHandshake((unsigned int *)a1) )
          return 0;
        v7 = 0x20000;
        if ( !(unsigned int)CMataNet_SendBlock((__int64)a1, &v7, 4, 1) )
          return 0;
        v7 = 0;
        v3 = 0;
        if ( (unsigned int)CMataNet_RecvBlock((__int64)a1, &v7, 4, 1, 0x12Cu) && v7 == 0x20100 )
        {
          v3 = 0x20200;
          v3 = (unsigned int)CMataNet_SendBlock((__int64)a1, &v7, 4, 1) != 0;
        }
      }
    }
  }
  return v3;
}
```

Characteristics

- Usually more complex to reverse and mock
- Can make network detection easier

Malware Examples

- Dacls/MATA
- CROSSWALK

Command Structure

Things to look for

- ”Small” numbers
  - 32-bit or 64-bit
- Little Endian or Big Endian
- ASCII/Unicode Characters

```
00000000  e2 09 00 00 85 aa 0c ac 00 00 00 00 41 00 00 00
00000010  d6 e6 2c ac 2a 68 77 79 ed cf 68 bc 3d 81 14 14
00000020  84 a0 00 d4 d8 f4 08 70 31 6d 48 e2 a9 50 e3 4a
00000030  1d 39 06 01 b6 cf 43 60 8f 0a e8 a5 fd f4 56 16
00000040  a0 ed 7d aa 1c 21 78 d8 ae 22 da 74 3e 1c cc 45
00000050  57
```

Common Fields

- ID/Opcode
- Length
- Payload

Payload

- You can get a rough idea of format from static analysis
- Test sending the command and debug the malware if necessary

Type of Commands

Typical

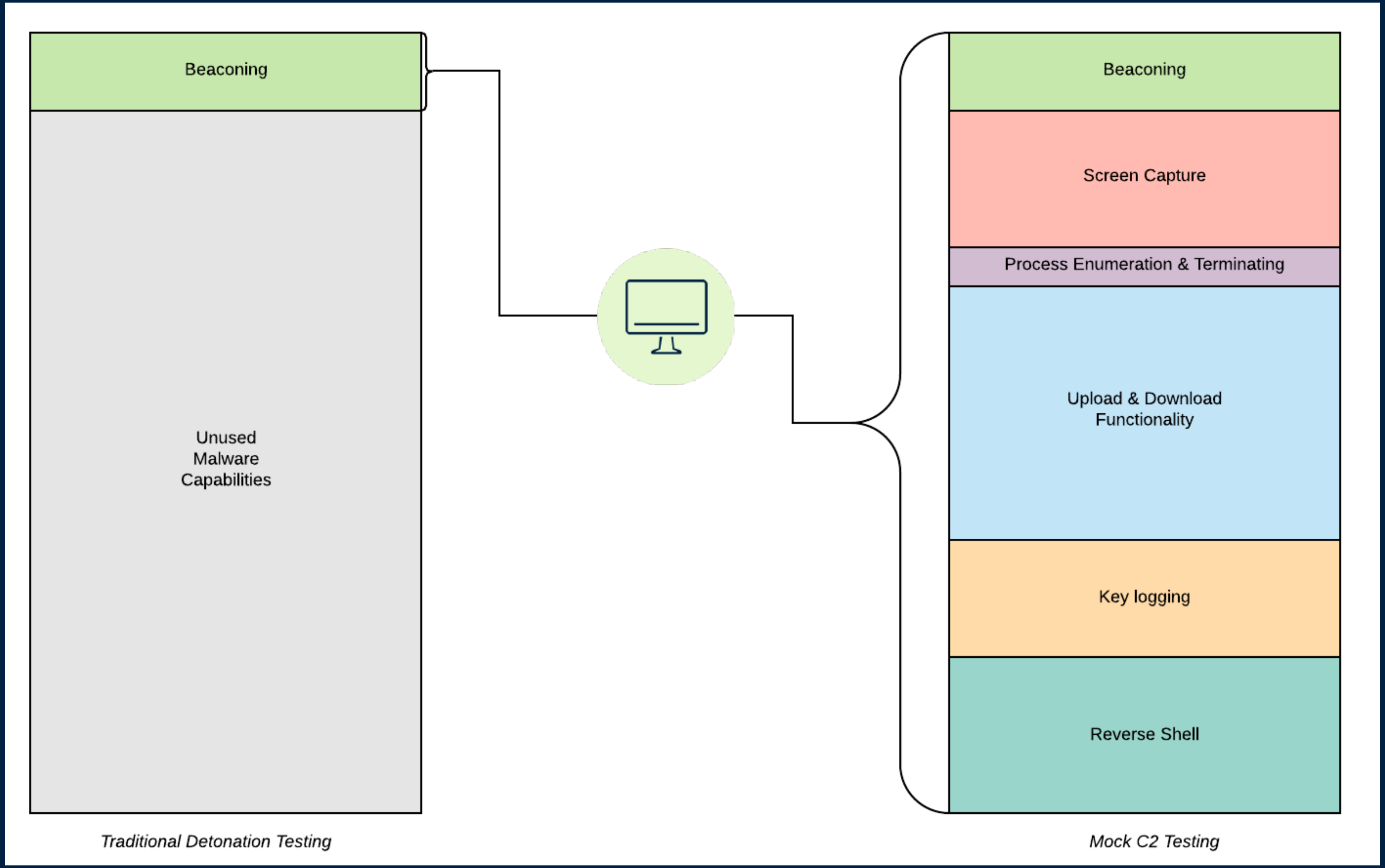
- Beacon
- Process Enumeration
- Listing files
- File copy/move/delete
- Upload/Download files
- Command execution

Less common

- Screenshot
- Live screen viewing
- Microphone recording
- Key logging

Simulating Attacks

Our approach



Consolidate our mock C2 servers into a centralized tool

Make it easy for researchers to reverse and implement new protocols

Make it easy for anyone to simulate an attack

Provide a user interface red teams are already used to

Demo

Closing Thoughts

Closing Thoughts

Mocking C2 servers can have a huge benefit

It’s easier than you might think to get a mock C2 server working

Contribute to the project!

- https://github.com/carbonblack/mockc2

Thank You