Analyzing the packer layers of rogue anti-virus programs

Rachit Mathur, McAfee Labs
Dr. Zheng Zhang, McAfee Labs

Outline

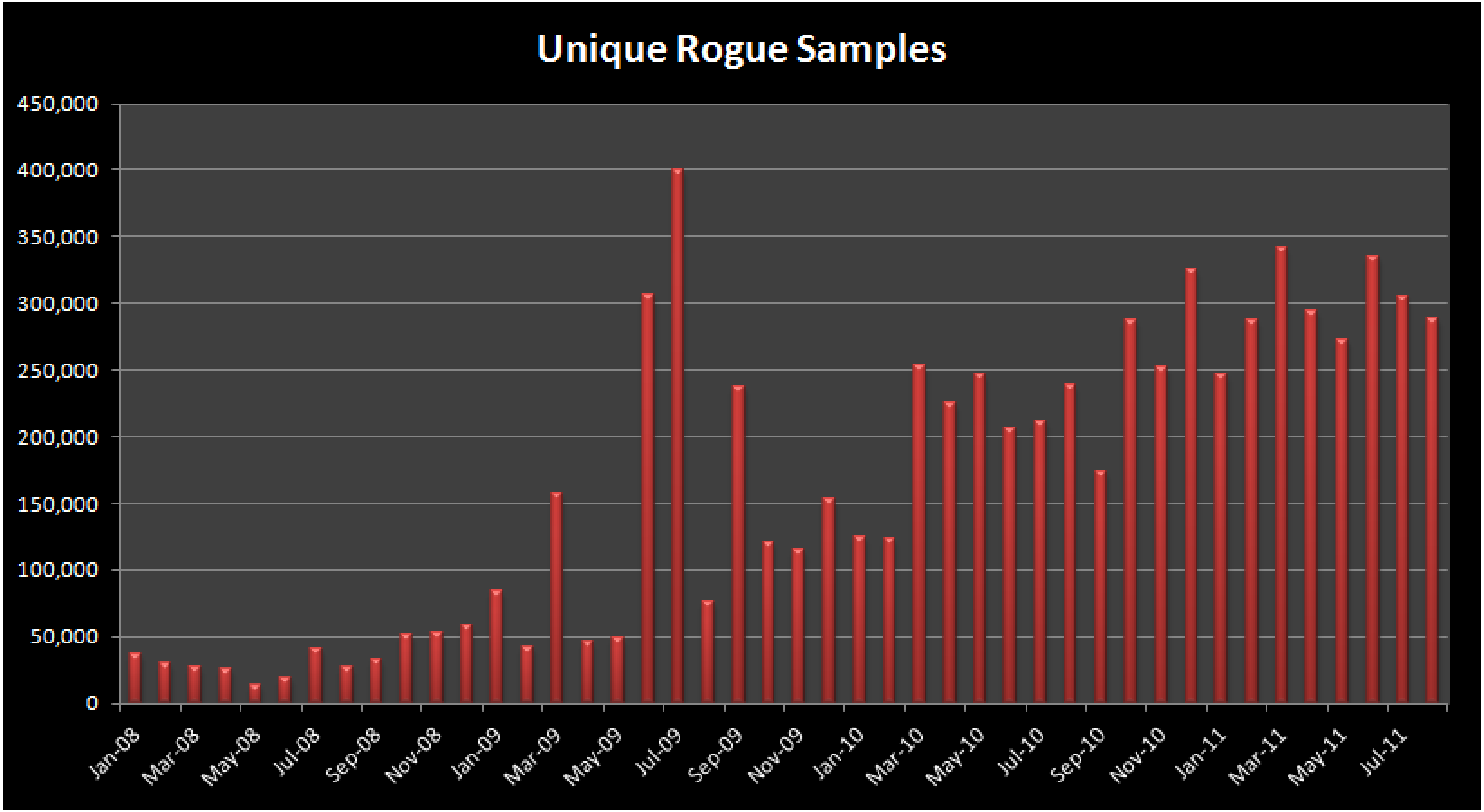
- Introduction
- Junk API Calls
- Exception Context Modifications
- Shared User Data Accesses
- INT 2C
- VM Instructions
- PEB Access
- DLL Image Check
- Conclusions

Introduction

- Rogue Anti-Virus programs a.k.a **Fake AV**
- Display fake notifications
- Make system unusable
- Scare-tactics

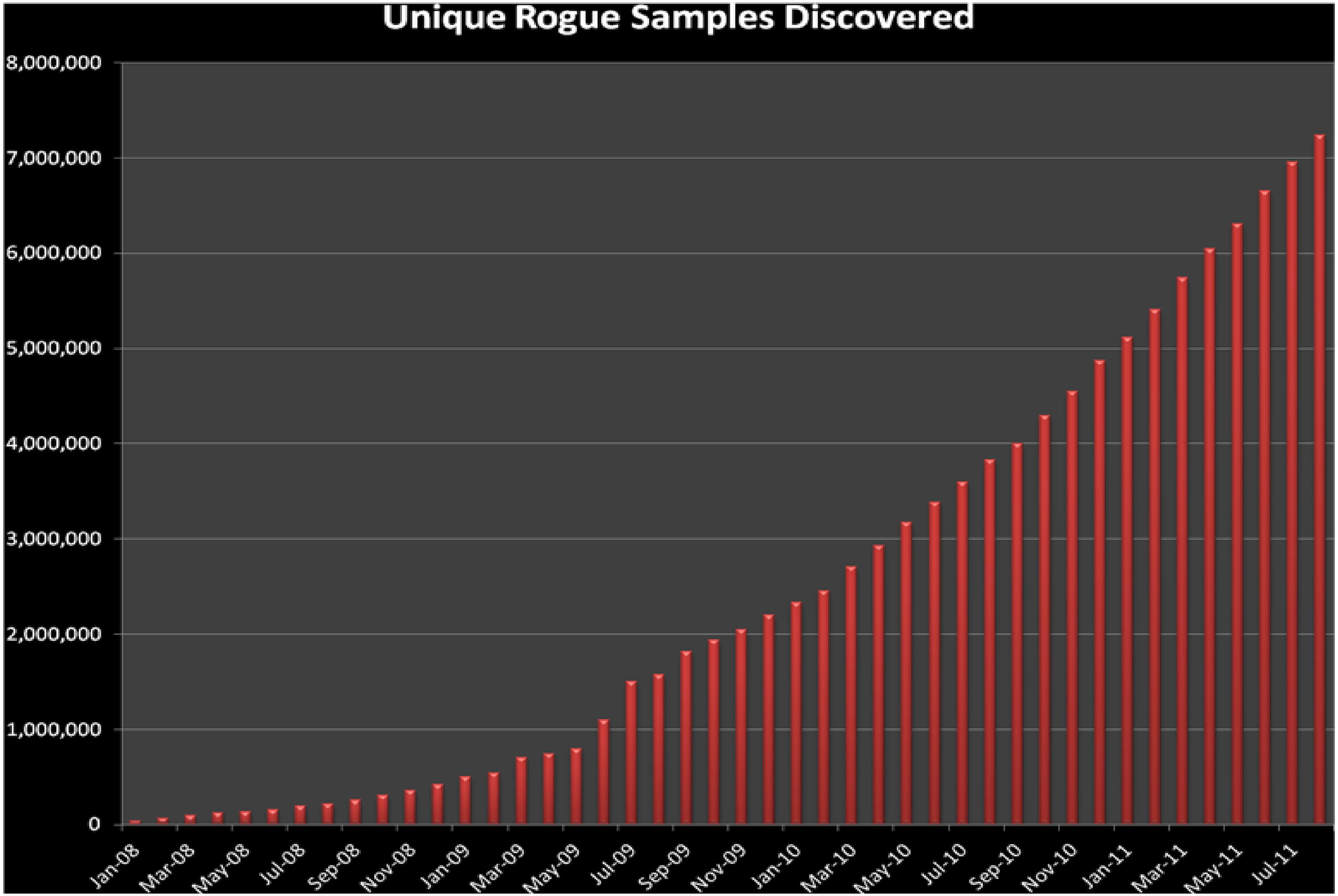
- Been around for years
  - Even a 2009 VB poll found 74% have come across one

Sample volume remains high (by month)



* Special thanks to Craig Schmugar

Sample volume (cumulative)



Packer Layers

Why so many unique samples?

- Server-side code mutations
- Multi-packing: combination of custom and public packers

The both techniques are not unique

- Polymorphism and packing are wide used in other malware families
- Emulation and unpacking generally cope well

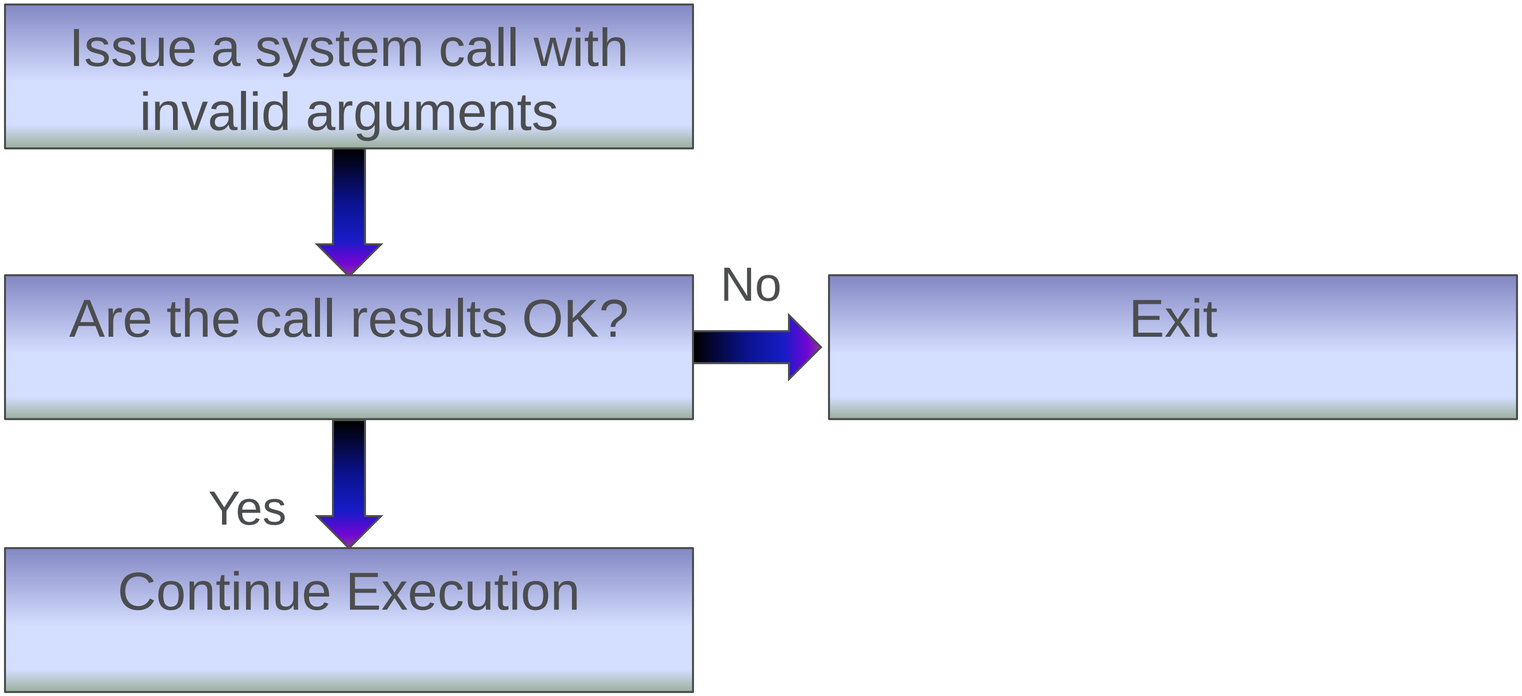
What makes Fake AV binaries different?

- Lots of strains and frequently mutating
- Anti-analysis techniques: creative, constant varying and lots of them

These anti-analysis techniques are the problem

- Let us look at some interesting techniques used by Fake AV

Junk API Calls - Invalid Arguments



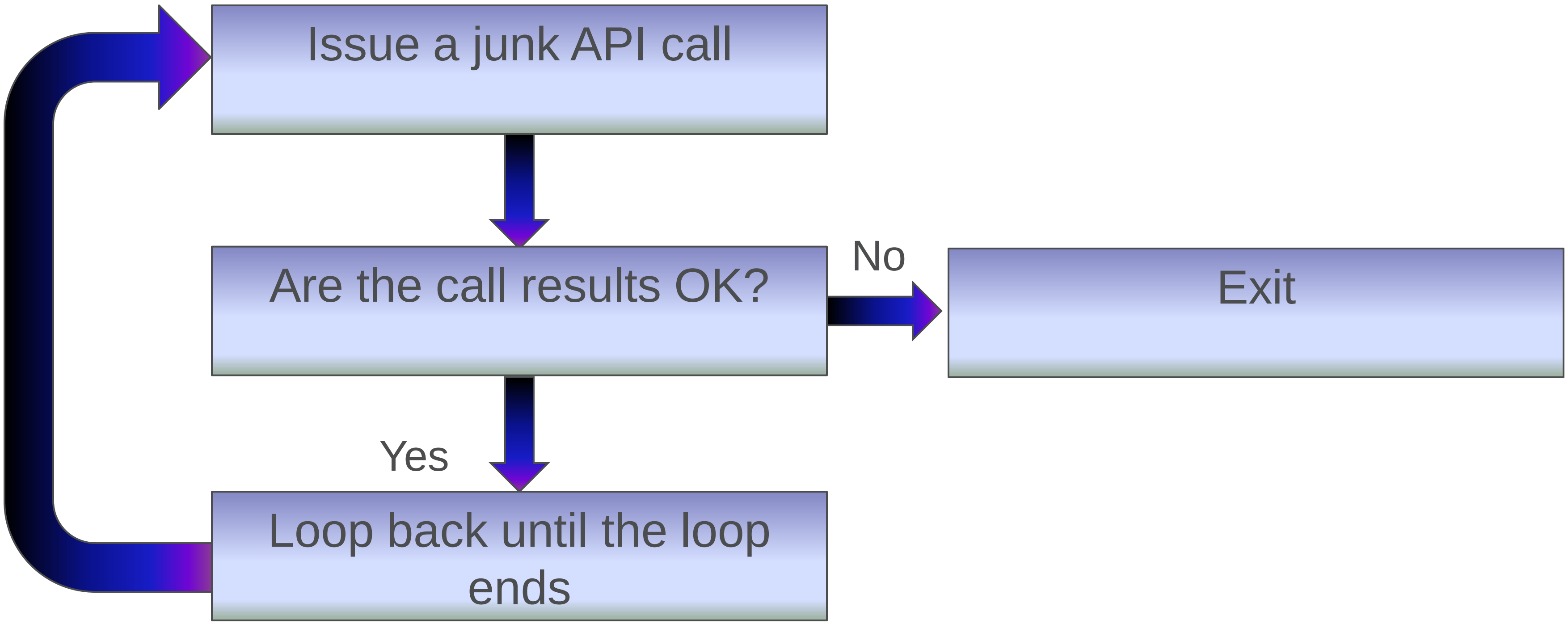
Junk API Calls - Invalid Arguments

```
if (!VirtuaAlloc(esp + 0x28, 0x9d000,
                 MEM_COMMIT|MEM_RESERVE,
                 PAGE_READWRITE))
{
  *ret_addr += (GetLastError() >> 8);
}
return;
```

Junk APIs used by Fake AV:

CreateEvent, LoadLibrary, LoadLibraryEx, CreateFile, VirtualAllocEx, FindActCtxSectionGuid, ZwOpenEvent, …

Junk API Calls – Long Loop



Junk API Calls – Long Loop

```
for (i = 0; i < 0x1000; i++)
    push(CreateMutexA(NULL, TRUE, NULL));
diff = CreateMutexA(NULL, TRUE, NULL) -
        CreateMutexA(NULL, TRUE, NULL);
for (i = 0; i < 0x1000; i++)
    CloseHandle(pop());
*ret_addr += diff + 4;
return;
```

Junk API Calls – Inline Patch

Install an inline hook to a low level system call

↓

Issue a junk system call to trigger hook callback

↓

Continue execution in the inline hook

Junk API Calls – Inline Patch

```
/// install the inline hook
addr = GetProcAddress(ntdll,
“NtQueryInformationFile”);
VirtualProtect(addr, 5,
               PAGE_EXECUTE_READWRITE,
               &old_protect);
memcpy(g_stolen_bytes, addr, 5);
*addr = 0xe9;
*((DWORD*) (addr + 1)) = FAV_Hook - 5 - addr;
/// trigger the hook
GetFileSizeEx(eax, ebp);
… /// wrong branch and eventually crash
```

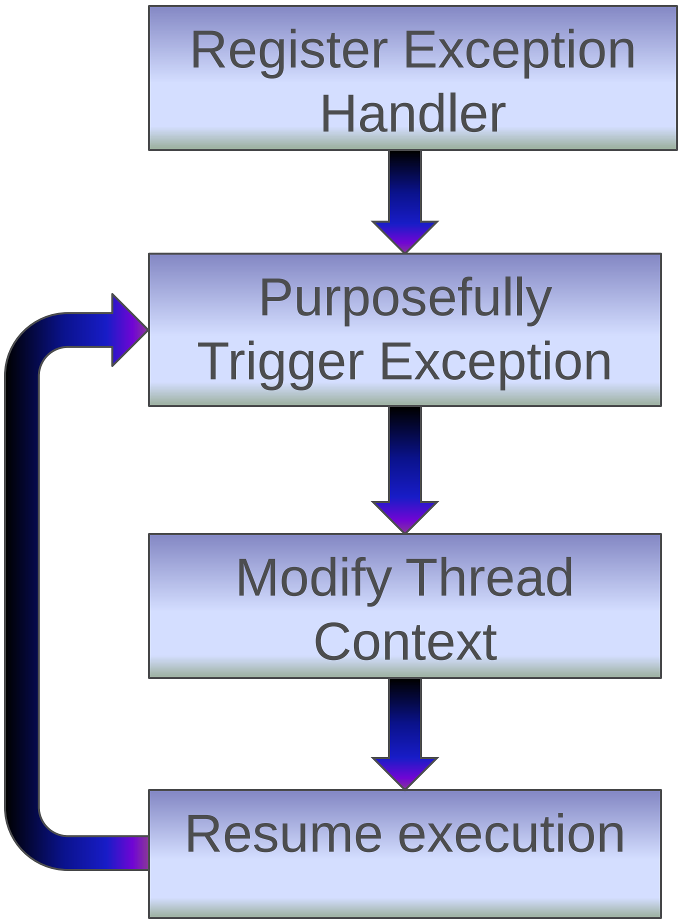
Junk API Calls – Inline Patch

```
FAV_Hook:   /// Get here after triggering inline hook
/// obtain the base pointer
ebp = *(esp + 0x44);
/// Make the first section read/writable
VirtualProtect(sect0_start, sect0_size,
                      PAGE_READWRITE, &old_protect);
/// remove the system call frames
esp += 0x48;
/// remove the inline hook
memcpy(Ntdll!NtQueryInformationFile,
              g_stolen_bytes, 5);
… /// Continue execution
```

Junk API Calls

- Challenges for AV emulation
  - OS emulation
    - Limited API emulation
    - Insufficient parameter validation
    - Emulate the system call chains, call stack frames
    - Recognize the inline hook
    - Mimic the real-OS behaviors
  - CPU emulation
    - Cope with junk loops and long delays
    - Emulation is slow

Context Modification




```
_CONTEXT
   +0x000 ContextFlags     : Uint4B
   +0x004 Dr0              : Uint4B
   +0x008 Dr1              : Uint4B
   +0x00c Dr2              : Uint4B
   +0x010 Dr3              : Uint4B
   +0x014 Dr6              : Uint4B
   +0x018 Dr7              : Uint4B
   +0x01c FloatSave        : _FLOATING_SAVE_AREA
   +0x08c SegGs            : Uint4B
   +0x090 SegFs            : Uint4B
   +0x094 SegEs            : Uint4B
   +0x098 SegDs            : Uint4B
   +0x09c Edi              : Uint4B
   +0x0a0 Esi              : Uint4B
   +0x0a4 Ebx              : Uint4B
   +0x0a8 Edx              : Uint4B
   +0x0ac Ecx              : Uint4B
   +0x0b0 Eax              : Uint4B
   +0x0b4 Ebp              : Uint4B
   +0x0b8 Eip              : Uint4B
   +0x0bc SegCs            : Uint4B
```

Swizzor
Armadillo

Context modification

```
pop  ecx   ; SE handler begin
lea  esp, [esp+4]
pop  eax
pop  edx
inc  dword ptr [edx+0B0h]   ; Context._EAX ; Loop counter
jnz  short loc_401039 ; Continue loop. Jumps to SE handler
                      ; epilog without modifying  EIP.
push eax
xor  eax, eax
xor  eax, [edx+0B8h] ; Context._EIP
add  eax, 58h
xchg eax, [edx+0B8h] ; To terminate loop, modify EIP such
                     ; that no more exceptions are triggered
pop  eax
```

Context modification

- Challenge for AV emulation
  - Long delays caused by both the exception handlings and junk loops
- Variations used for over a year now

Shared User Area

- Vital Windows structure mapped to user-mode processes
- Not well documented
- Typically mapped at fixed address: 0x7FFE0000

```
_KUSER_SHARED_DATA (0x7ffe0000)
   +0x000 TickCountLow      : 0x62aa
   +0x004 TickCountMultiplier : 0xa03afb7
   +0x008 InterruptTime     : _KSYSTEM_TIME
   +0x014 SystemTime        : _KSYSTEM_TIME
   ....
   +0x030 NtSystemRoot      : [260] 0x43
   ...
   +0x300 SystemCall        : 0x7c90eb8b
   +0x304 SystemCallReturn : 0x7c90eb94
   …
   +0x320 TickCountQuad
```

```
GetTickCount:
    mov edx, 7FFE0000h
    mov eax, [edx]
    mul dword ptr [edx+4]
    shrd eax, edx, 18h
    retn
```

Shared User Area Access

- Simple check to see if memory exist and hold correct value

ADD ECX,DWORD PTR [7FFE0304] ntdll.KiFastSystemCallRet

- Use NTSystemRoot string for decryption

- Obfuscate control-flow and thwart analysis

```
cmp     dword ptr ds:7FFE0300h, 0 ; SystemCall pointer points
                              ;           to ntdll.KiFastSyatemCall
jz      short nullsub_1
jmp     dword ptr ds:7FFE0304h ; SystemCallReturn pointer points
endp                          ;          to ntdll.KiFastSyatemCallRet
```

Shared User Area Access

- Verify that “times moves” during decryption

```
push   7FFDFFF8h
LOOP_START:
clc
mov    eax, [esp]      ; eax <== 7ffdfff8
push   ecx
pop    ecx
mov    ecx, [eax+328h] ; [eax+328] = 0x7ffe0320 TickCountQuad
add    ecx, [eax+8] ;[eax+8] = 0x7ffe0000 TickCountLow
shr    ecx, 2
mov    eax, [edi] ; edi points to an encrypted data area
movsx  ecx, cl
xor    eax, ebx
xor    eax, ecx
xor    al, 4Dh
jnz    short LOOP_START
add    esp, 4
```

Shared User Area Access

- Challenges for AV emulation
  - Map and populate the structure
  - Constantly update the dynamic fields
- Used by many Fake AV variants to access various fields
  - One of Fake AV’s all time favorites

INT 2C

- WinNt.h: VOID DbgRaiseAssertionFailure(void) { __asm int 0x2c }
- Changes the EAX and EDX registers
- Detects both debuggers and emulators

```
int     2Ch                       ; Internal routine for MSDOS (IRET)
                                  ; Has a side-effect of setting edx
                                  ; to address of next instruction.
add     eax, 4
inc     esi
lea     ebx, [ebp+arg_40FCFF]     ; load address saved earlier
add     edx, 3                    ; add 3 to address of the instruction
                                  ; just after int 2Ch
mov     eax, 17h
cmp     edx, ebx                  ; Verify that edx was modified properly
                                  ; due to int 2Ch instruction
jz      short loc_1444379
```

VM Instructions

- Virtualization instructions for Intel VT and AMD-V technologies
- Used to trigger exceptions and transfer controls to SEH handler
- Challenges for AV emulation:
  - Needs to support the latest x86 instructions

PEB Access

Process Environment Block

```
+0x000 InheritedAddressSpace : UChar
+0x001 ReadImageFileExecOptions : UChar
+0x002 BeingDebugged    : UChar
+0x003 SpareBool        : UChar
+0x004 Mutant           : Ptr32 Void
+0x008 ImageBaseAddress : Ptr32 Void
+0x00c Ldr              : Ptr32 _PEB_LDR_DATA
......
+0x088 NumberOfHeaps    : Uint4B
+0x08c MaximumNumberOfHeaps : Uint4B
+0x090 ProcessHeaps     : Ptr32 Ptr32 Void
```

_HEAP

```
+0x000 Entry            : _HEAP_ENTRY
+0x008 Signature        : Uint4B
+0x00c Flags            : Uint4B
+0x010 ForceFlags       : Uint4B
```

_PEB_LDR_DATA

```
+0x000 Length           : Uint4B
+0x004 Initialized      : UChar
+0x008 SsHandle         : Ptr32 Void
+0x00c InLoadOrderModuleList : _LIST_ENTRY
+0x014 InMemoryOrderModuleList : _LIST_ENTRY
+0x01c InInitializationOrderModuleList : _LIST_ENTRY
+0x024 EntryInProgress  : Ptr32 Void
```

PEB Access

```
mov    eax, [eax+90h]  ;get ProcessHeaps from PEB
mov    eax, [eax]  ;get default heap
mov    eax, [eax+8]  ;get signature
test   eax, 1  ;check value
Jnz    EMU_DETECTED ; jump, if detected
xor    eax, 0EEFFEEFFh  ;xor with expected
                ;signature value to get zero
xor    eax, ecx  ;assign ecx to eax
jmp    dword ptr [eax+ebp]  ;jumps to correct
                ;location only if signature matched
```

- Challenges for AV emulation:
  - Accurately emulate the PEB and lower-level structures.

DLL Image Check

- Checks the PE headers and API entry points

```
mov  eax, [ebp-4] ; eax <== kernel32 pe header
; Check the kernel32 TimeDateStamp stamp field.
; If matched, go to a wrong branch.
mov  edx, [eax+IMAGE_NT_HEADERS.FileHeader.TimeDateStamp]
sub  edx, 12345678h
jnz  loc_409D98  ; taken if not detected
… ; Get here, if the timestamp is faked
```

- Challenges for AV emulation
  - How to emulate the DLL images that are close enough to the real DLLs.

Conclusions

Fake AV is sophisticated in its anti-analysis techniques

- Many of the techniques are specifically attacking AV emulation.

How to better handle the threats?

- Improve AV emulator
  - Better OS emulation
  - Faster CPU emulation that supports the latest instruction sets
- ‘Unconventional‘ heuristics to detect questionable techniques
  - Can be risky

Hopefully understanding of these challenges will help to improve and to raise-the-bar!

Gracias! - Thank you!

- Questions?

Contact:

Rachit_Mathur@McAfee.com

Zheng_Zhang@McAfee.com

McAfee®