Bypassing patchguard on Windows 8.1 and Windows 10

Mark Ermolov, Artem Shishkin

Positive Technologies
What is patchguard?

— “Please don’t patch our kernels” call from MS

— Even if your kernel patch is correct, you’ll catch a BSOD
  • 0x109 CRITICAL_STRUCTURE_CORRUPTION

— Protected structures
  • System images: ntoskrnl.exe, win32k.sys, hal.dll etc.
  • System structures: IDT, GDT, Syscall tables etc.

— Periodic checksums validation for protected stuff

— Doesn’t work on Windows 9
What if we really need to?

— Go for it!

— But...
  - Patchguard developers are prepared for reverse engineers
  - Hyper-inlined obfuscation © Alex Ionescu
  - Anti-debugging tricks
  - Several ways of checks invocation
Code obfuscation

— Symbol stripping

```
sub_140F3CFC2 proc near

var_1B18 = dword ptr -1B18h
BugCheckParameter4 = qword ptr -1AF8h
var_1AF0 = qword ptr -1AF0h
var_1AE8 = qword ptr -1AE8h
var_1AD8 = dword ptr -1AD8h
var_1AD4 = dword ptr -1AD4h
Src = qword ptr -1A00h
var_1AC8 = qword ptr -1AC8h
var_1AC0 = qword ptr -1AC0h
var_1AB8 = qword ptr -1AB8h
Size = qword ptr -1A00h
var_1AA8 = qword ptr -1A08h
var_1AA0 = qword ptr -1A00h
anonymous_13 = qword ptr -1A98h
anonymous_12 = qword ptr -1A90h
anonymous_24 = qword ptr -1A88h
anonymous_23 = qword ptr -1A70h
anonymous_40 = qword ptr -1A60h
anonymous_39 = qword ptr -1A40h
anonymous_2 = qword ptr -196Ch
anonymous_19 = qword ptr -1940h
anonymous_22 = qword ptr -1910h
anonymous_25 = dword ptr -1718h
anonymous_21 = qword ptr -1130h

; CODE XREF: KiFilterFiberContext+117'h
; KiFilterFiberContext+1C2'h ...
```
Code obfuscation

— Misleading names

<table>
<thead>
<tr>
<th>CmpAppendD11Section proc near</th>
<th>db</th>
<th>2Eh</th>
</tr>
</thead>
<tbody>
<tr>
<td>2E 48 31 11</td>
<td>xor</td>
<td>[rcx], rdx</td>
</tr>
<tr>
<td>48 31 51 08</td>
<td>xor</td>
<td>[rcx+8], rdx</td>
</tr>
<tr>
<td>48 31 51 10</td>
<td>xor</td>
<td>[rcx+10h], rdx</td>
</tr>
<tr>
<td>48 31 51 18</td>
<td>xor</td>
<td>[rcx+18h], rdx</td>
</tr>
<tr>
<td>48 31 51 20</td>
<td>xor</td>
<td>[rcx+20h], rdx</td>
</tr>
<tr>
<td>48 31 51 28</td>
<td>xor</td>
<td>[rcx+28h], rdx</td>
</tr>
<tr>
<td>48 31 51 30</td>
<td>xor</td>
<td>[rcx+30h], rdx</td>
</tr>
<tr>
<td>48 31 51 38</td>
<td>xor</td>
<td>[rcx+38h], rdx</td>
</tr>
<tr>
<td>48 31 51 40</td>
<td>xor</td>
<td>[rcx+40h], rdx</td>
</tr>
<tr>
<td>48 31 51 48</td>
<td>xor</td>
<td>[rcx+48h], rdx</td>
</tr>
<tr>
<td>48 31 51 50</td>
<td>xor</td>
<td>[rcx+50h], rdx</td>
</tr>
<tr>
<td>48 31 51 58</td>
<td>xor</td>
<td>[rcx+58h], rdx</td>
</tr>
<tr>
<td>48 31 51 60</td>
<td>xor</td>
<td>[rcx+60h], rdx</td>
</tr>
<tr>
<td>48 31 51 68</td>
<td>xor</td>
<td>[rcx+68h], rdx</td>
</tr>
<tr>
<td>48 31 51 70</td>
<td>xor</td>
<td>[rcx+70h], rdx</td>
</tr>
<tr>
<td>48 31 51 78</td>
<td>xor</td>
<td>[rcx+78h], rdx</td>
</tr>
</tbody>
</table>
Code obfuscation

- Code junk generation
  - Loop unrolling
  - Dead code insertion
  - Indirect calls and variable accesses
Anti-debugging

— Works only on free builds without kernel debugger!

```c
__int64 KeInitAmd64SpecificState()
{
    signed int v0; // edx@2
    __int64 result; // rax@2

    if ( !InitSafeBootMode )
    {
        v0 = __ROR4__(KdPitchDebugger | KdDebuggerNotPresent, 1);
        result = (v0 / ((KdPitchDebugger | KdDebuggerNotPresent) != 0 ? -1 : 17));
    }
    return result;
}
```
Anti-debugging

— Randomly inserted checks for debugger presence
Anti-debugging

— If you use breakpoints, they will be included to a patchguard checksum, leading to a 0x109 bugcheck
— If you use hardware breakpoints, well...

```
cli       
sidt       fword ptr [rbp+320h]
lidt       fword ptr [rbp+228h]
mov        dr7, r13
lidt       fword ptr [rbp+320h]
sti        
```
Non-linear code flow

— Active usage of Vectored Exception Handling

```c
int64 KeInitAmd64SpecificState()
{
    signed int v0; // edx@2
    __int64 result; // rax@2

    if ( !InitSafeBootMode )
    {
        v0 = __ROR4__(KdPitchDebugger | KdDebuggerNotPresent, 1);
        result = (v0 / ((KdPitchDebugger | KdDebuggerNotPresent) != 0 ? -1 : 17));
    }
    return result;
}
```
Reverse-engineering

— For dynamic analysis with KD (with windbg f.e.)
  • Remove all kd presence checks manually
    — Look them up with IDA scripting
    — Apply patches in KD with pykd
    — Do it before “Phase1InitializationDiscard“

— For static analysis with IDA
  • Try not to give up waiting for patchguard initialization function decompilation
    — Use something else, like hypervisor-based debugger ;)

POSITIVE TECHNOLOGIES
Reverse-engineering

— Since patchguard is developed incrementally, the key functions in reversing it are
  • KiFilterFiberContext – chooses the way for invoking patchguard checks
  • Unnamed sub inside KiFilterFiberContext – creates a structure aka patchguard context and schedules it’s verification
  • Other functions (like context checkers) can be misleadingly named, but you can look them up around KiFilterFiberContext since they are located in a single compilation unit
Bypassing patchguard

— There are different approaches
  • patch kernel image so that patchguard will just not start
  • hook KeBugCheckEx and restore the state of a system
  • modify checkers so that they would be always valid

  • de-schedule contexts verification
    — This is what we’ve implemented
Contexts verification scheduling

— Context verification might be launched with
  
  • KeSetCoalescableTimer
    — A timer that periodically launches context verification
  
  • Prcb.AcpiReserved
    — A certain ACPI event (f.e. Idle transition)
  
  • Prcb.HalReserved
    — A hal timer clock
  
  • PsCreateSystemThread
    — A queued system thread that sleeps a random amount of time
  
  • KeInsertQueueApc
    — A queued regular kernel APC
  
  • KiBalanceSetManagerPeriodicDpc
    — A periodic event which happens every "KiBalanceSetManagerPeriod" ticks
Contexts verification descheduling

— So we’ve got to deschedule context verification once and for all
  • KeSetCoalescableTimer
    — Timer? Disable!
  • Prcb.AcpiReserved
    — Zero out this field
  • Prcb.HalReserved
    — Same here
  • PsCreateSystemThread
    — Scan sleeping worker threads and set wait time to infinite for suitable
  • KeInsertQueueApc
    — Same here
  • KiBalanceSetManagerPeriodicDpc
    — Revert to KiBalanceSetManagerDeferredRoutine
Thank you!

- ashiskin [at] ptsecurity [dot] com
- mermolov [at] ptsecurity [dot] com
- www.ptsecurity.com
- blog.ptsecurity.com