KEYPATCH: binary patcher for IDA Pro

http://keystone-engine.org/keypatch

NGUYEN Anh Quynh <aquynh -at- gmail.com>

Trada hacking - 16/9/2016

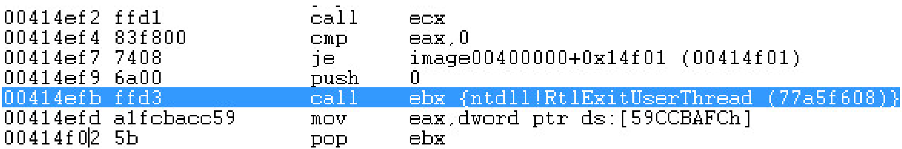
Who am I

- Nguyen Anh Quynh, aquynh -at- gmail.com
  - Nanyang Technological University, Singapore
  - PhD in Computer Science
  - Operating System, Virtual Machine, Binary analysis, etc
  - Capstone disassembler: http://capstone-engine.org
  - Unicorn emulator: http://unicorn-engine.org
  - Keystone assembler: http://keystone-engine.org

Binary patching

- CrackMe, CTF challenges
- Malware analysis
- Modify binary without source code :-)

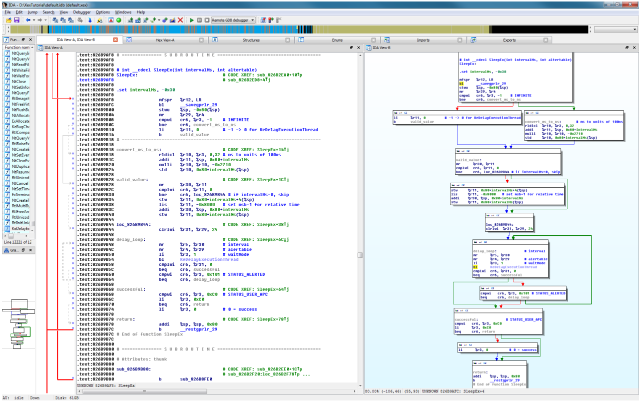
URLZone Banking Trojan



```
00414ef2 ffd1                call     ecx
00414ef4 83f800              cmp      eax,0
00414ef7 7408                je       image00400000+0x14f01 (00414f01)
00414ef9 6a00                push     0
00414efb ffd3                call     ebx {ntdll!RtlExitUserThread (77a5f608)}
00414efd a1fcbacc59          mov      eax,dword ptr ds:[59CCBAFCh]
00414f02 5b                  pop      ebx
```

IDA Pro

- https://www.hex-rays.com
- De-facto binary analysis tool
- Extendable with plugin SDK (C, Python)

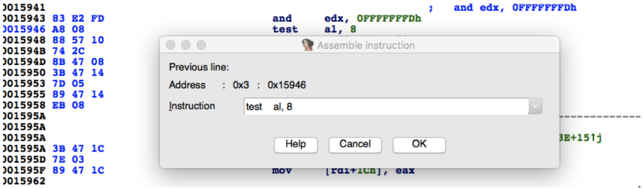


```
.text:826B9AF8 # =============== S U B R O U T I N E =======================================
.text:826B9AF8
.text:826B9AF8
.text:826B9AF8 # int __cdecl SleepEx(int intervalMs, int altertable)
.text:826B9AF8 SleepEx:                                # CODE XREF: sub_826B2EA0+10↑p
.text:826B9AF8                                         # sub_826B2ED8+4↑j
.text:826B9AF8
.text:826B9AF8 .set intervalNs, -0x30
.text:826B9AF8
.text:826B9AF8                 mfspr   %r12, LR
.text:826B9AFC                 bl      __savegprlr_29
.text:826B9B00                 stwu    %sp, -0x80(%sp)
.text:826B9B04                 mr      %r29, %r4
.text:826B9B08                 cmpwi   cr6, %r3, -1    # INFINITE
.text:826B9B0C                 bne     cr6, convert_ms_to_ns
.text:826B9B10                 li      %r11, 0         # -1 -> 0 for KeDelayExecutionThread
.text:826B9B14                 b       valid_value
.text:826B9B18 # ---------------------------------------------------------------------------
.text:826B9B18
.text:826B9B18 convert_ms_to_ns:                       # CODE XREF: SleepEx+14↑j
.text:826B9B18                 rldicl  %r10, %r3, 0,32 # ms to units of 100ns
.text:826B9B1C                 addi    %r11, %sp, 0x80+intervalNs
.text:826B9B20                 mulli   %r10, %r10, -0x2710
.text:826B9B24                 std     %r10, 0x80+intervalNs(%sp)
.text:826B9B28
.text:826B9B28 valid_value:                            # CODE XREF: SleepEx+1C↑j
.text:826B9B28                 mr      %r30, %r11
.text:826B9B2C                 cmplwi  cr6, %r11, 0
.text:826B9B30                 bne     cr6, loc_826B9B44 # if intervalMs=0, skip
.text:826B9B34                 stw     %r11, 0x80+intervalNs+4(%sp)
.text:826B9B38                 lis     %r11, -0x8000   # set msb=1 for relative time
.text:826B9B3C                 addi    %r30, %sp, 0x80+intervalNs
.text:826B9B40                 stw     %r11, 0x80+intervalNs(%sp)
.text:826B9B44
.text:826B9B44 loc_826B9B44:                           # CODE XREF: SleepEx+38↑j
.text:826B9B44                 clrlwi  %r31, %r29, 24
.text:826B9B48
.text:826B9B48 delay_loop:                             # CODE XREF: SleepEx+6C↓j
.text:826B9B48                 mr      %r5, %r30       # interval
.text:826B9B4C                 mr      %r4, %r29       # alertable
.text:826B9B50                 li      %r3, 1          # waitMode
.text:826B9B54                 bl      KeDelayExecutionThread
.text:826B9B58                 cmplwi  cr6, %r31, 0
.text:826B9B5C                 beq     cr6, successful
.text:826B9B60                 cmpwi   cr6, %r3, 0x101 # STATUS_ALERTED
.text:826B9B64                 beq     cr6, delay_loop
.text:826B9B68
.text:826B9B68 successful:                             # CODE XREF: SleepEx+64↑j
.text:826B9B68                 cmpwi   cr6, %r3, 0xC0  # STATUS_USER_APC
.text:826B9B6C                 li      %r3, 0xC0
.text:826B9B70                 beq     cr6, return
.text:826B9B74                 li      %r3, 0          # 0 = success
.text:826B9B78
.text:826B9B78 return:                                 # CODE XREF: SleepEx+78↑j
.text:826B9B78                 addi    %sp, %sp, 0x80
.text:826B9B7C                 b       __restgprlr_29
.text:826B9B7C # End of function SleepEx
.text:826B9B7C
.text:826B9B80
.text:826B9B80 # =============== S U B R O U T I N E =======================================
.text:826B9B80
.text:826B9B80 # Attributes: thunk
.text:826B9B80
.text:826B9B80 sub_826B9B80:                           # CODE XREF: sub_826B2EE0+1C↑p
.text:826B9B80                                         # sub_826B2F20:loc_826B2F78↑p ...
.text:826B9B80                 b       sub_826B8FE0
UNKNOWN 826B9AF8: SleepEx
```

```
80.00% (-106,46) (55,93) UNKNOWN 826B9AFC: SleepEx+4
```

Built-in binary patcher of IDA

- Modify binary code with menu "Edit | Patch program | Assemble..."



```
)015941                                                   ;   and edx, 0FFFFFFFDh
)015943 83 E2 FD                  and     edx, 0FFFFFFFDh
)015946 A8 08                     test    al, 8
)015948 88 57 10
)01594B 74 2C
)01594D 8B 47 08
)015950 3B 47 14
)015953 7D 05
)015955 89 47 14
)015958 EB 08
)01595A
)01595A
)01595A                                                                 3E+15↑j
)01595A 3B 47 1C
)01595D 7E 03
)01595F 89 47 1C                  mov     [rdi+1Ch], eax
)015962
```

Assemble instruction

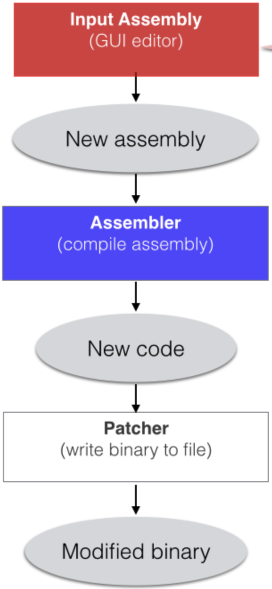
Previous line:

Address : 0x3 : 0x15946

Instruction: test al, 8

- Save changes permanently to binary file
  - Menu "Edit | Patch program | Apply patches to input file..."

How it work?



Input Assembly
(GUI editor)

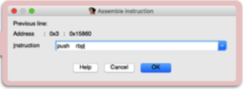
New assembly

Assembler
(compile assembly)

New code

Patcher
(write binary to file)

Modified binary



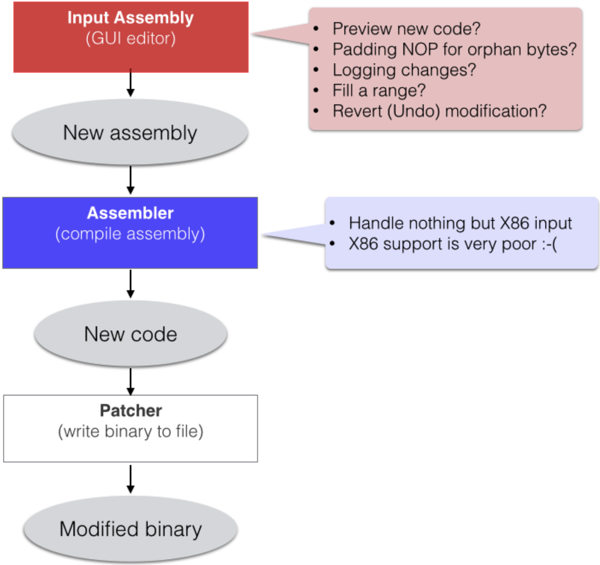
Assemble instruction

Previous line:

Address : 0x3 : 0x15860

Instruction push rbp

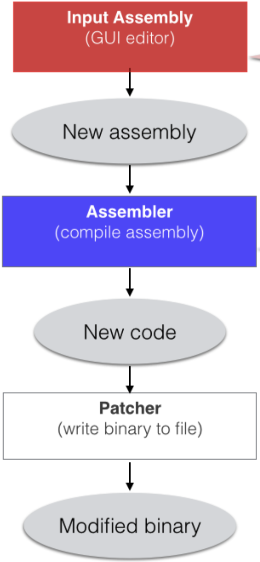
Problems of IDA built-in binary patcher



- Preview new code?
- Padding NOP for orphan bytes?
- Logging changes?
- Fill a range?
- Revert (Undo) modification?

- Handle nothing but X86 input
- X86 support is very poor :-(

Keypatch Solution



Input Assembly
(GUI editor)

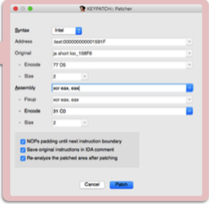
New assembly

Assembler
(compile assembly)

New code

Patcher
(write binary to file)

Modified binary



KEYPATCH:: Patcher

Syntax: Intel

Address: .text:000000000001591F

Original: ja short loc_158F6

- Encode: 77 D5
- Size: 2

Assembly: xor eax, eax

- Fixup: xor eax, eax
- Encode: 31 C0
- Size: 2

☑ NOPs padding until next instruction boundary
☑ Save original instructions in IDA comment
☑ Re-analyze the patched area after patching

Keystone == Next Generation Assembler Framework

Assembler framework

Definition

- Compile assembly instructions & returns encoding as sequence of bytes
  - Ex: inc EAX → 40
- May support high-level concepts such as macro, function, etc
- Framework to build apps on top of it

Applications

- Dynamic machine code generation
  - Binary rewrite
  - Binary searching

Good assembler framework?

- True framework
    - Embedded into tool without resorting to external process
- Multi-arch
    - X86, Arm, Arm64, Mips, PowerPC, Sparc, etc
- Updated
    - Keep up with latest CPU extensions
- Multi-platform
    - *nix, Windows, Android, iOS, etc
- Bindings
    - Python, Ruby, Go, NodeJS, etc

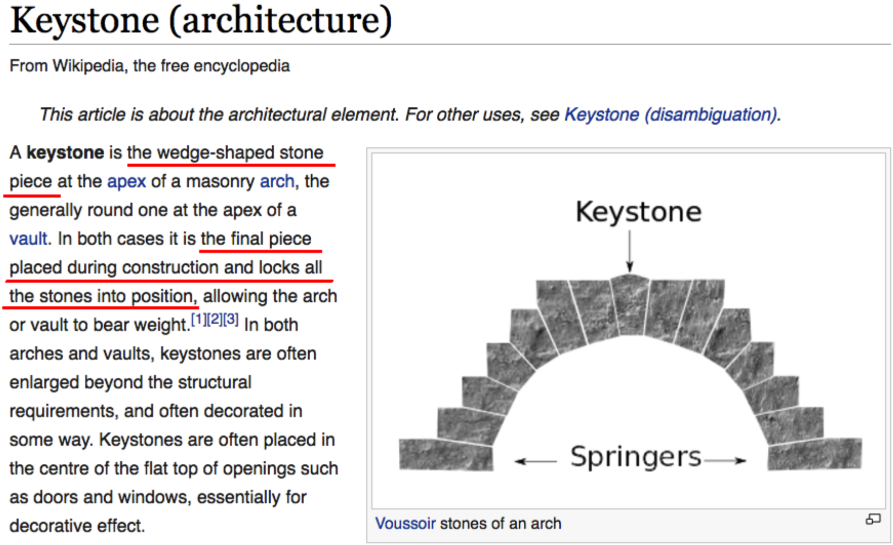
Existing assembler frameworks

- Nothing is up to our standard, even in 2016!
  - Yasm: X86 only, no longer updated
  - Intel XED: X86 only, miss many instructions & closed-source
  - Other important archs: Arm, Arm64, Mips, PPC, Sparc, etc?

Life without assembler frameworks?

- People are very much struggling for years!
    - Use existing assembler tool to compile assembly from file
    - Call linker to link generated object file
    - Use executable parser (ELF) to parse resulted file for final encoding
- Ugly and inefficient
- Little control on the internal process & output
- Cross-platform support is very poor

"If not now, then when? If not you, then who?" - Kailash Satyarthi



Keystone (architecture)

From Wikipedia, the free encyclopedia

This article is about the architectural element. For other uses, see Keystone (disambiguation).

A **keystone** is the wedge-shaped stone piece at the apex of a masonry arch, the generally round one at the apex of a vault. In both cases it is the final piece placed during construction and locks all the stones into position, allowing the arch or vault to bear weight.[1][2][3] In both arches and vaults, keystones are often enlarged beyond the structural requirements, and often decorated in some way. Keystones are often placed in the centre of the flat top of openings such as doors and windows, essentially for decorative effect.

Voussoir stones of an arch

Timeline

- Indiegogo campaign started on March 17th, 2016 (for 3 weeks)
  - 99 contributors, 4 project sponsors
- Beta code released to beta testers on April, 2016
  - Only Python binding available at this time
- Version 0.9 released on May, 2016: http://keystone-engine.org
  - More bindings by beta testers: NodeJS, Ruby, Go & Rust
- Version 0.9.1 released on July 27th, 2016
  - 2 more bindings: Haskell & OCaml

Keystone engine

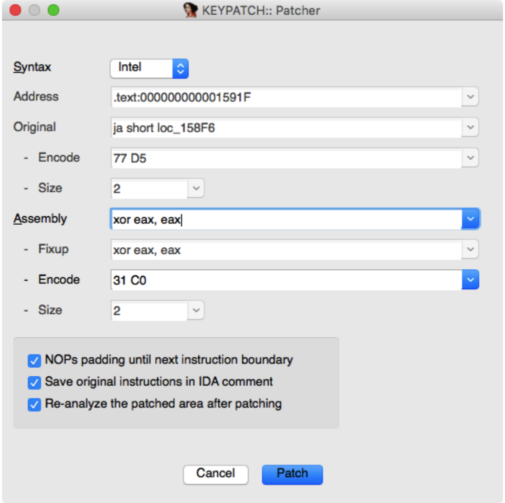
- True framework
  - Embedded into tool without resorting to external process
- Multi-arch
  - X86, Arm, Arm64, Mips, PowerPC, Sparc, Hexagon, SystemZ
- Updated
  - Keep up with latest CPU extensions
- Multi-platform
  - *nix, Windows, Android, iOS, etc
- C++ core & multi-bindings
  - Python, Ruby, Go, NodeJS, OCaml, Rust, Haskell
- Support various X86 undocumented instructions
- Compact & lightweight: 10x smaller than LLVM

Keypatch binary patcher for IDA

Keypatch

- Co-developed with Thanh Nguyen (VNSecurity.net)
- Open source IDA plugin `http://keystone-engine.org/keypatch`
- Tool for assembling & patching in IDA
- Built on top of Keystone assembler framework
  - Version 1.0 released at BlackHat USA 2016, August 4th, 2016
  - Version 2.0 released on September 14th, 2016
  - Version 2.0.1 released on September 15th, 2016

Keypatch - Patcher



KEYPATCH:: Patcher

Syntax: Intel

Address: .text:000000000001591F

Original: ja short loc_158F6

- Encode: 77 D5
- Size: 2

Assembly: xor eax, eax

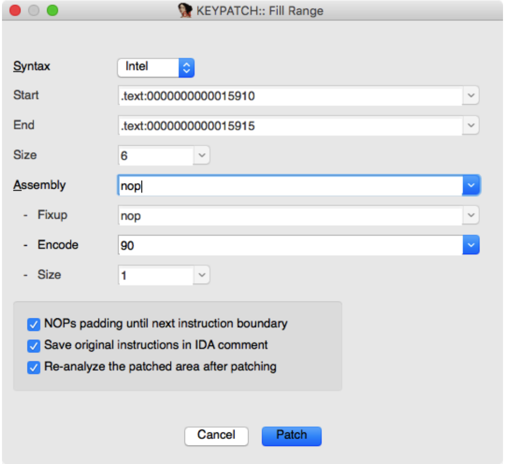
- Fixup: xor eax, eax
- Encode: 31 C0
- Size: 2

☑ NOPs padding until next instruction boundary

☑ Save original instructions in IDA comment

☑ Re-analyze the patched area after patching

Keypatch - Fill Range



KEYPATCH:: Fill Range

Syntax: Intel

Start: .text:0000000000015910

End: .text:0000000000015915

Size: 6

Assembly: nop

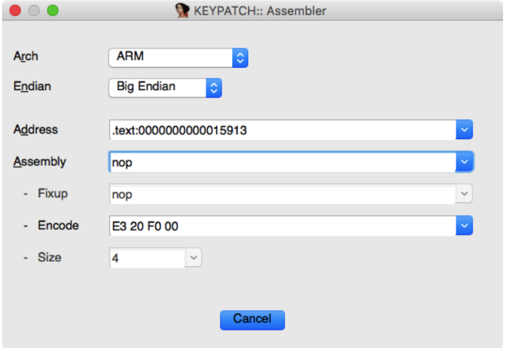
- Fixup: nop
- Encode: 90
- Size: 1

☑ NOPs padding until next instruction boundary

☑ Save original instructions in IDA comment

☑ Re-analyze the patched area after patching

Keypatch - Assembler



KEYPATCH:: Assembler

Arch: ARM

Endian: Big Endian

Address: .text:0000000000015913

Assembly: nop

- Fixup: nop
- Encode: E3 20 F0 00
- Size: 4

Keypatch vs IDA's built-in patcher

- More friendly
  - Code preview
  - Padding NOPs automatically
  - Logging modifications
  - Fill a range of selected code
  - Assembler (do not modify)
  - Revert (undo)
- Support 8 architectures
  - Arm, Arm64, Hexagon, Mips, PowerPC, Sparc, SystemZ, X86
  - X86 support is fantastic
- Open source

Conclusions

- Keypatch is a superior binary patcher for IDA
  - Multi-arch + multi-platform
  - Feature-rich & friendly
  - Open source
- Looking for new contributors for our open source projects
  - Keypatch + Keystone engine
  - Capstone engine + Unicorn engine

KEYSTONE ENGINE
UNICORN ENGINE
CAPSTONE ENGINE

References

- Keypatch: http://keystone-engine.org/keypatch
- Keystone assembler
  - Homepage: http://keystone-engine.org
  - Twitter: @keystone_engine
  - Github: http://github.com/keystone-engine/keystone
  - Mailing list: http://freelists.org/list/keystone-engine

Questions and answers

KEYPATCH: binary patcher for IDA Pro

http://keystone-engine.org/keypatch

NGUYEN Anh Quynh <aquynh -at- gmail.com>