# Ucieczka z Matrixa: (nie)bezpieczna analiza malware

Mateusz “j00ru” Jurczyk

SECURE 2014, Warszawa

# Mateusz Jurczyk

- ISE @ Project Zero, Google
- Dragon Sector Team Vice-Captain
- @j00ru
- http://j00ru.vexillium.org/

# O czym będzie?

- **O malware.**
- **O narzędziach do ich analizy.**
- **O podatnościach bezpieczeństwa w tychże.**
- **O ich *aktywnym* wykorzystaniu w celu wykonania złośliwego kodu na maszynie ofiary.**

Krótko mówiąc: jakie zagrożenia czyhają na analityków złośliwego oprogramowania na realnych przykładach.

# O czym NIE będzie?

O pasywnym utrudnianiu analizy

(obfuskacji, trikach anti-debugging, steganografii itp.)

# A rok temu na SECURE…

*“Rzecz o zwiększaniu (nie)bezpieczeństwa. Food for thought.”*

Gynvael Coldwind

# A rok temu na SECURE…

Gynvael pokazał, że aplikacje antywirusowe oprócz wykrywania zagrożeń mogą poważnie zwiększać *attack surface* systemu komputerowego.

# A rok temu na SECURE…

- Programy AV to w końcu świetny target dla bughunterów i atakujących:
  - Najczęściej napisane w językach natywnych (wydajność)…
  - Parsują skomplikowane formaty danych: pliki wykonywalne, archiwa, dokumenty, …
  - Z prawami admina lub jądra…
  - Przyjmując jako wejście każdy plik pojawiający się na maszynie.

# Teza się potwierdza

**Przez ostatni rok pojawiło się jeszcze więcej dowodów:**

- Joxean Koret, Breaking Anti-Virus Software, SyScan 2014 [1].
  - dziesiątki krytycznych błędów w większości produktów AV.
- Tavis Ormandy, Denial of Service in Microsoft Malware Protection Engine, June 2014 [2].

# Teza się potwierdza

**Joxean Koret** @matalaz · Oct 17

There are some antivirus that I wonder if they happen to know how to code at all. I'm not even talking about security aware coding.




1

# No dobra, …

… antywirusy mogą nie być do końca bezpieczne.

A jeśli uogólnimy nieco problem, i zastanowimy się nad całym zbiorem software, które “styka się” ze złośliwym oprogramowaniem?

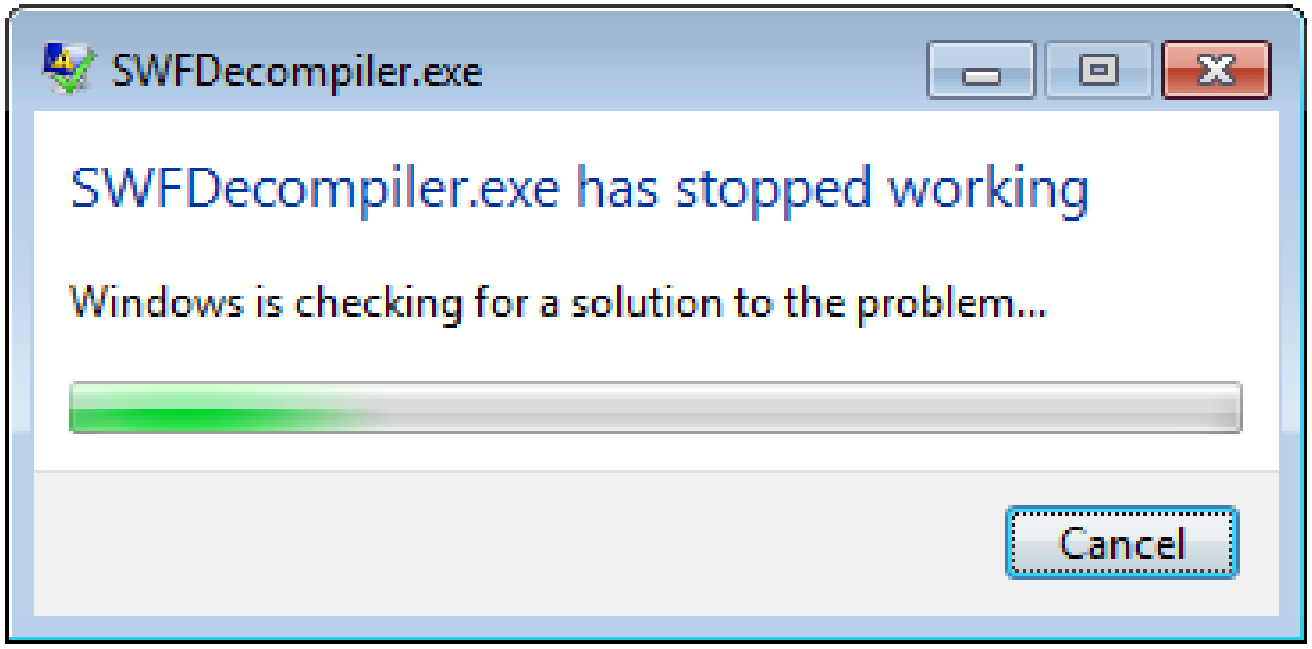
Na przykład nad narzędziami do analizy malware?

# Krótka dygresja

## Jak powstał temat na tę prezentację?

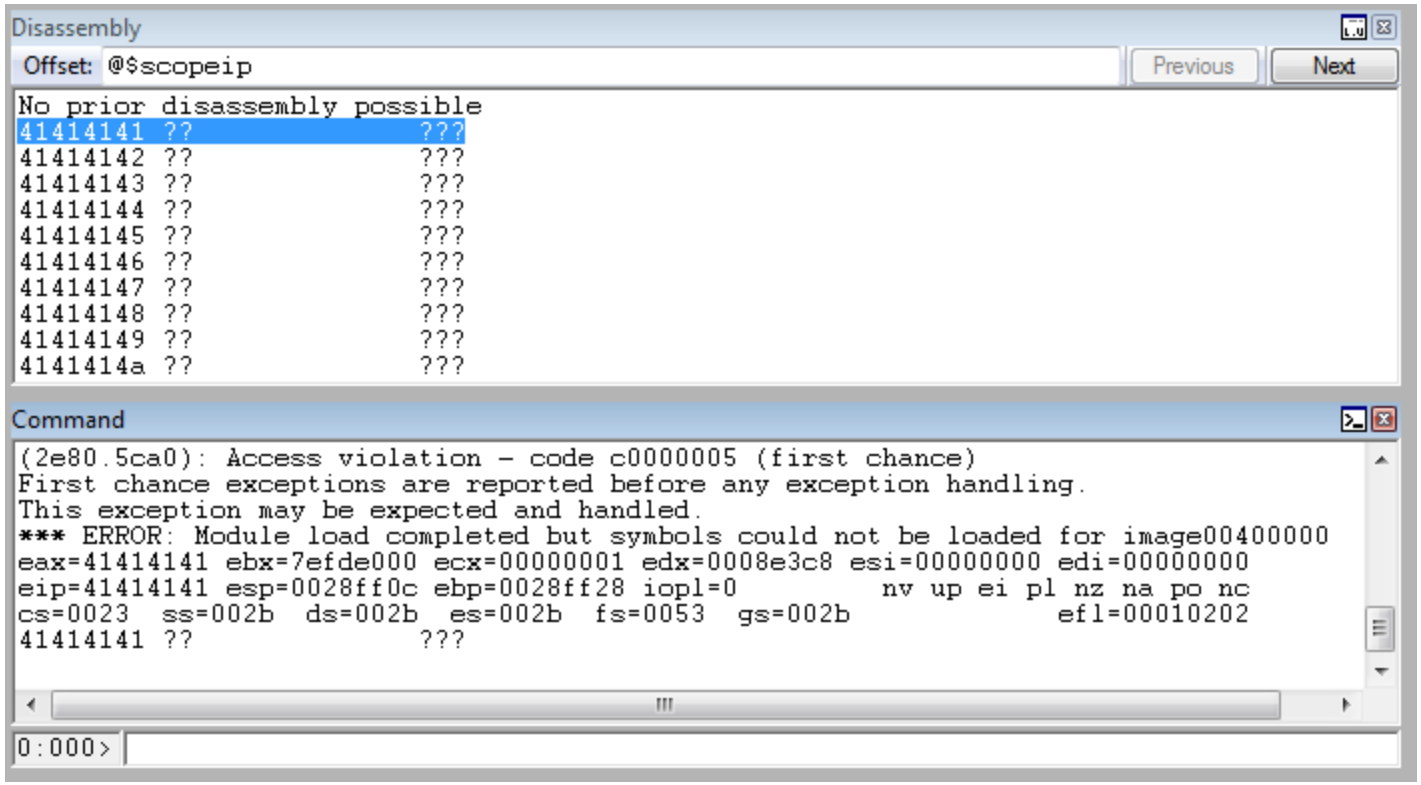
- Rok 2012. Na pewnej popularnej stronie z końcówką .cn (Chiny) znajduję plik SWF o interesującej mnie funkcjonalności.
- Radośnie ładuję ów plik do *Sothink SWF Decompiler*, jednego z lepszych dekompilatorów plików Flash.
- Odczekuję kilka sekund, po czym…

# Krótka dygresja



# Krótka dygresja

Podpinam debugger, a tam:



# OK, OK, I get the message!

# Wracając do tematu…

Czy możemy zaufać narzędziom, których na codzień używają ludzie w firmach AV, innych organizacjach, a właściwie i my sami?

- Kto nie uruchomił kiedyś *ProcessExplorera*, *ProcessMonitora* lub *Wiresharka*?

**Prosta odpowiedź: niestety nie. ☹**

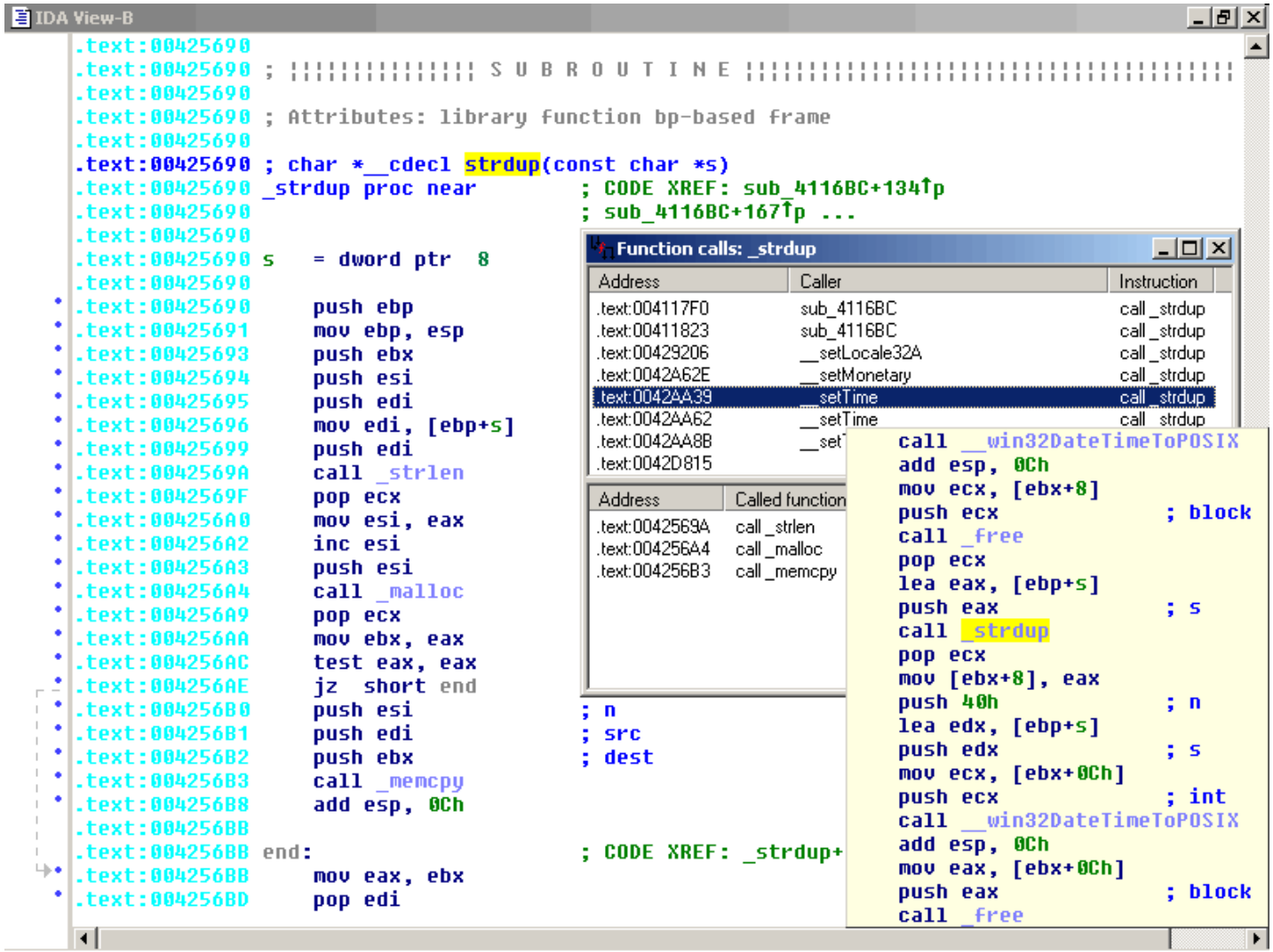
Kod przeznaczony do pracy *z definicji* w niebezpiecznym środowisku często nie reprezentuje wcale wyższego standardu.

# PRZYKŁADY

# Hex-Rays IDA Pro

- **IDA Pro** to obecnie najlepszy dostępny na rynku deasembler plików wykonywalnych.
  - Działa na platformach Windows, GNU/Linux, OS X.
  - Ogromna baza użytkowników: niezbędne narzędzie każdego *reverse-engineera*.
  - Możliwość dokupienia dekompilatorów dla platform IA-32, IA-64 oraz ARM 32.

# Znajomy widok



# Hex-Rays inwestuje w bezpieczeństwo

- ASLR, DEP, /GS i inne zabezpieczenia domyślnie włączone w pliku wykonywalnym i bibliotekach.
- Własna, bezpieczniejsza implementacja standardowych funkcji C takich jak `strncat` czy `strncpy`.
- Bug bounty program: za każdy krytyczny błąd w deasemblerze lub dekompilatorze firma płaci **3000 USD**.

# Bug bounty program

Hex-Rays Home

Hex-Rays

Bug Bounty

Home | Products | Support | Forum | Blog | News | About us | Contact | Site Map

## Hex-Rays Security Bug Bounty Program

We have been toying with the idea since 2008 and finally decided to go ahead with it. In short, if you find a security bug in IDA or the Decompiler and report it to us, you may receive a cash award.

The purpose of our Security Bug Bounty Program to make our tools more secure and reward those who help us in this endeavor.

Please check the reported vulnerabilties below.

### Bounty Guidelines

1. Hex-Rays will pay a 3000 USD bounty for certain security bugs.
2. All IDA or Decompiler license holders can participate (with or without active support plan), except Hex-Rays employees and their families.
3. What security bugs will be considered:
    1. Only bugs in Hex-Rays products (**IDA and the Decompiler**) are eligible.
    2. Security bugs must be in Hex-Rays code (not in third party/contributed code). In some cases we may take responsibility for third-party code as well.
    3. Security bugs must be original and previously unreported and not fixed yet.
    4. Security bugs with high or critical impact are eligible (remote code execution, privilege escalation, etc).
    5. Security bugs must be present in the latest public release of IDA/Decompiler.
    6. Security bugs must work on a clean, unmodified installation of IDA/Decompiler with all publicly released patches applied.
    7. In some cases we may accept bugs which require modification of the default settings of IDA (but not any binary patching, registry editing etc.).
    8. Security bugs have to be triggered without user's interaction, or with interaction which happens naturally during user's work.
4. Bugs which are NOT eligible for the bounty:
    1. Issues with our web site
    2. Bugs which occur when the user explicitly starts a debugging session, executes a script, or any other action which may lead to execution of external code as part of its normal functionality.
    3. Anti-debugging and similar tricks.
    4. Simple crashes and denial-of-service bugs, although we'll still be interested to get the reports of these :)
5. How to apply: send your report to bugbounty@hex-rays.com. The report should include the POC code and a small description of the bug and its impact.
6. We reserve the right to refuse a bounty payment if we believe the actions of the reporter have endangered the security of Hex-Rays' end users.
7. The duration of the bounty program: undetermined. We reserve the right to close the program at any moment.
8. What will be asked from the reporters: a proper and legal picture identification and bank account information within 30 days of the bug acknowledgement.
9. Collective entries are allowed. The bounty will be paid to the person designated by the group.

https://hex-rays.com/bugbounty.shtml

# Z drugiej strony...

## ... IDA Pro jest świetnym targetem dla bughuntera.

- Napisany w całości w C/C++.
- Część kodu źródłowego dostępna w IDA SDK.
- Częściowa lista wspieranych formatów plików:

MS DOS, EXE File, MS DOS COM File, MS DOS Driver, New Executable (NE), Linear Executable (LX), Linear Executable (LE), Portable Executable (PE) (x86, x64, ARM), Windows CE PE (ARM, SH-3, SH-4, MIPS), Mach-O for OS X and iOS (x86, x64, ARM and PPC), Dalvik Executable (DEX), EPOC (Symbian OS executable), Windows Crash Dump (DMP), XBOX Executable (XBE), Intel Hex Object File, MOS Technology Hex Object File, Netware Loadable Module (NLM), Common Object File Format (COFF), Binary File, Object Module Format (OMF), OMF library, S-record format, ZIP archive, JAR archive, Executable and Linkable Format (ELF), Watcom DOS32 Extender (W32RUN), Linux a.out (AOUT), PalmPilot program file, AIX ar library (AIAFF), PEF (Mac OS or Be OS executable), QNX 16 and 32-bits, Nintendo (N64), SNES ROM file (SMC), Motorola DSP56000 .LOD, Sony Playstation PSX executable files,, object (psyq) files, library (psyq) files

# Z drugiej strony…

## … IDA Pro jest świetnym targetem dla bughuntera.

- Jeszcze więcej obsługiwanych formatów assembly (ponad 60 rodzin procesorów):

AMD K6-2 3D-Now! extensions, ARM Architecture versions from v3 to v8 including Thumb, Thumb-2, DSP instructions and NEON Advanced SIMD instructions., ARMv4/ARMv4T: ARM7 cores (ARM7TDMI/ARM710T/ARM720T/ARM740T), ARM9 cores (ARM9TDMI/ARM920T/ARM922T/ARM940T), ARMv5/ARMv5TE/ARMv5TEJ: ARM9 cores (ARM946E-S/ARM966E-S/ARM968E-S/ARM926EJ-S/ARM996HS), ARM10E (ARM1020E/ARM1022E/ARM1026EJ-S), ARMv6/ARMv6T2/ARMv6Z/ARMv6K: ARM11 cores (ARM1136J(F)-S/ARM1156T2(F)-S/ARM1176JZ(F)-S/ARM11 MPCore), ARMv6-M: Cortex-M0/Cortex-M0+/Cortex-M1 (e.g. NXP LPC800/LPC1xxx, Freescale Kinetis L and M series, STM32 F0 series etc.), ARMv7-M: Cortex-M3 (e.g. NXP LPC17xx/18xx/13xx, STM32 F1/F2/L1 series, TI Stellaris, Toshiba TX03/TMPM3xx etc.), ARMv7E-M: Cortex-M4 (e.g. NXP LPC43xx, STM32 F3/F4 series, TI Stellaris LM4F, Freescale Kinetis K series and W series, Atmel AT91SAM4 etc.), ARMv7-R: Cortex-R4(F)/Cortex-R5/Cortex-R7 (e.g. TI TMS570LS etc.), ARMv7-A: Cortex-A5/Cortex-A7/Cortex-A8/Cortex-A9/Cortex-A12/Cortex-A15 (e.g. TI Sitara, TI OMAP series, Samsung S5PC100 and Exynos, Nvidia Tegra, Freescale i.MX, Allwinner A-Series and many others), ARMv7 (custom): Apple A4/A5/A5X/A6/A6X (Swift microarchitecture, used in Apple's iPhone/iPod/iPad/AppleTV), Qualcomm Snapdragon, Note: this list is incomplete; code for any ARM-compliant core can be disassembled , ATMEL AVR, DEC PDP-11, Fujitsu FR, GameBoy, Hitachi/Renesas H8/300, H8/300L, H8/300H, H8S/2000, H8S/2600, H8SX, Hitachi H8/500, Hitachi HD 6301, HD 6303, Hitachi HD 64180, INTEL 8080, INTEL 8085, INTEL 80196, INTEL 8051, INTEL 860XR, INTEL 960, INTEL 80x86 and 80x87, INTEL Pentium family, including SSE, SSE2, SSE3, SSE4, Java Virtual Machine, KR1878, Microsoft .NET (Common Language Infrastructure bytecode), Mitsubishi MELPS740 or Renesas 740, Hitachi/Renesas M16C, MN102 (comes only with source code), MOS Technologies 6502, Motorola/Freescale MC680xx, CPU32 (68330), MC6301, MC6303, MC6800, MC6801, MC6803, MC6805, MC6808, HCS08, MC6809, MC6811, M68H12C, ColdFire, Motorola MC6812/MC68HC12/CPU12, Freescale HCS12, HCS12X (including XGATE coprocessor), NSC CR16 (comes only with source code), NEC V850 and V850E1 (V850ES), EFI Byte Code (EBC), SPU (Synergistic Processing Unit of the Cell BE), MSP430, MSP430X, PIC 12XX, PIC 14XX, PIC 18XX, PIC 16XXX, Rockwell C39 (comes only with source code), SAM8, SGS Thomson ST-7, and ST-20, TLCS900 (comes only with source code), unSP from SunPlus, Philips XA series (51XA G3), Intel xScale, Z80, Zilog Z8, Zilog Z180, Zilog Z380, x64 architecture (Intel x64 and AMD64), ARM64 Architecture (aka AArch64), ARMv8-A: Cortex-A50/Cortex-A53/Cortex-A57, ARMv8 (custom): Apple A7 (Cyclone microarchitecture, used in iPhone 5s), Dalvik (Android bytecode, DEX), DEC Alpha, DSP563xx, DSP566xx, DSP561XX, TI TMS320C2X, TMS320C5X, TMS320C6X, TMS320C64X, TMS 320C54xx, TMS320C55xx, TMS320C3, TI TMS320C27x/TMS320C28x, Hewlett-Packard HP-PA, Hitachi SH1, SH2, SH3, Hitachi SH4 - Dreamcast, SH-4a, IBM/Motorola PowerPC/POWER architecture, including Power ISA extensions:, Book E (Embedded Controller Instructions), Freescale ISA extentions (isel etc.), SPE (Signal Processing Engine) instructions, AltiVec (SIMD) instructions, Hypervisor and virtualization instructions, All instructions from the Power ISA 2.06 specification (Vector, Decimal Floating Point, Integer Multiply-Accumulate, VSX etc.), Cell BE (Broadband Engine) instructions (used in PlayStation 3), VLE (Variable Length Encoding) compressed instruction set, Xenon (Xbox 360) instructions, including VMX128 extension, Paired Single SIMD instructions (PowerPC 750CL/Gekko/Broadway/Espresso, used in Nintendo Wii and WiiU), Motorola/Freescale PowerPC-based cores and processors, including (but not limited to):, MPC5xx series: MPC533/MPC535/MPC555/MPC556/MPC561/MPC562/MPC563/MPC564/MPC566, Note: code compression features of MPC534/MPC564/MPC556/MPC566 (Burst Buffer Controller) are currently not supported, MPC8xx series (PowerQUICC): MPC821/MPC850/MPC860, MPC8xxx series (PowerQUICC II, PowerQUICC II Pro, PowerQUICC III): MPC82xx/MPC83xx/MPC85xx/MPC87xx, MPC5xxx series (Qorivva): MPC55xx, MPC56xx, MPC57xx, Power PC 4xx, 6xx, 74xx, e200 (including e200z0 with VLE), e500 (including e500v1, e500v2 and e500mc), e600, e700, e5500, e6500 cores, QorIQ series: P1, P2, P3, P4, P5 and T1, T2, T4 families, Infineon Tricore architecture (up to architecture v1.6), Intel IA-64 Architecture - Itanium., Motorola DSP 56K, Motorola MC6816, MIPS, MIPS Mark I (R2000), MIPS Mark II (R3000), MIPS Mark III: (R4000, R4200, R4300, R4400, and R4600), MIPS Mark IV: R8000, R10000, R5900 (Playstation 2), MIPS32, MIPS32r2, MIPS32r3 and MIPS64, MIPS64r2, MIPS64r3, Allegrex CPU (Playstation Portable), including VFPU instructions, Cavium Octeon ISA extensions, MIPS16 (MIPS16e) Application Specific Extension, MIPS-MT, MIPS-3D, smartMIPS Application Specific Extensions, Tohiba TX19/TX19A Family Application Specific Extension (MIPS16e+ aka MIPS16e-TX), Mitsubishi M32R, Mitsubishi M7700, Mitsubishi M7900, Nec 78K0 and Nec 78K0S, STMicroelectronics ST9+, ST-10, SPARCII, ULTRASPARC, Siemens C166 (flow), Fujitsu F2MC-16L, Fujitsu F2MC-LC

# Wiele błędów zgłoszonych i naprawionych od 2011

**Reported vulnerabilities**

| Date | Reporter | Products | Description |
|---|---|---|---|
| 2011-02-08 19:21 | Stefan Esser | IDA 5.7 and 6.0 | Vulnerability in Macho-O loader |
| 2011-02-10 10:37 | Alin Rad Pop | IDA 5.7 and 6.0 | Vulnerability in the conversion of string encodings |
| 2011-02-11... | Masaaki Chida | IDA 5.7 and 6.0 | Multiple vulnerabilities |
| 2011-02-20... | Masaaki Chida | IDA 5.7 and 6.0 | Multiple vulnerabilities |
| 2011-03-18... | undisclosed | IDA 5.7 and 6.0 | Plugin autorun vulnerability |
| 2011-04-10... | undisclosed | IDA 5.7 and 6.0 and early copies of 6.1 | WinDbg autorun vulnerability |
| 2012-03-19 19:50 | Greg MacManus | IDA versions up to 6.2 | Python autorun script vulnerability |
| 2013-07-07 01:33 | Masaaki Chida | IDA versions 6.3 and 6.4 | Vulnerability in .net processor module |
| 2013-07-15 at 19:14 | Masaaki Chida | IDA versions up to 6.4 | Windbg autorun vulnerability |
| 2013-07-21 11:13 | Masaaki Chida | IDA versions up to 6.4 | Vulnerability in hint calculation |
| 2014-01-05 at 01:07 | George Hotz | IDA versions up to 6.5 | Vulnerability in Mach-O loader |
| 2014-06-09 17:52 | Tadashi Kobayashi | IDA versions up to 6.6 | Vulnerability in til file loading |
| 2014-09-06 12:54 | Mateusz Jurczyk | IDA versions up to 6.6 | Multiple vulnerabilities |

https://www.hex-rays.com/bugbounty.shtml

# Czy opłaca się szukać dalej?

- Pytanie zadane w połowie sierpnia 2014.
- Zdecydowałem się dać sobie szansę.
  - Na 100% nie wszystkie obsługiwane formaty zostały dokładnie przeaudytowane.
  - Wysoka nagroda to dobry motywator.
- Przez około 3 tygodnie audytowałem wieczorami otwarty i zamknięty kod IDA.

# Rezultaty pobieżnego audytu

- **12** różnych **błędów** klasy “memory corruption” znalezionych w parserach formatów:
  - QNX, COFF, DBG, EPOC, DEX, PEF
- **2 błędy** uznane przez Hex-Rays za NOFIX ze względu na nierealistyczny scenariusz ataku.
- Pozostałe **10** podatności sklasyfikowane jako **6** odrębnych problemów.
- Poprawione przez producenta w mniej niż dwa tygodnie.
  - Data zgłoszenia: **6 września 2014**
  - Data wypuszczenia łatki: **15 września 2014**

10. qrealloc() could manage to allocate 0xDEADBEEF bytes on Linux64. This value was used to force a std:bad_alloc() exception, and a successful memory allocation was not what other parts of IDA were expecting. The bug was reported by Mateusz Jurczyk on 2014-09-06 at 12:54. We provide a fix for v6.5 and v6.6.
11. COFF: maliciously truncated symbol table could lead to a memory corruption. The bug was reported by Mateusz Jurczyk on 2014-09-06 at 12:54. We provide a fix for v6.5 and v6.6.
12. EPOC: a specially crafted input file could lead to a memory corruption. The bug was reported by Mateusz Jurczyk on 2014-09-06 at 12:54. We provide a fix for v6.5 and v6.6.
13. DEX: a specially crafted input file could lead to a memory corruption. The bug was reported by Mateusz Jurczyk on 2014-09-06 at 12:54. We provide a fix for v6.5 and v6.6.
14. PEF: a specially crafted input file could lead to a memory corruption. The bug was reported by Mateusz Jurczyk on 2014-09-06 at 12:54. We provide a fix for v6.5 and v6.6.
15. Also the archive includes fixes for other bugs (not necessarily security bugs) discovered and fixed so far.

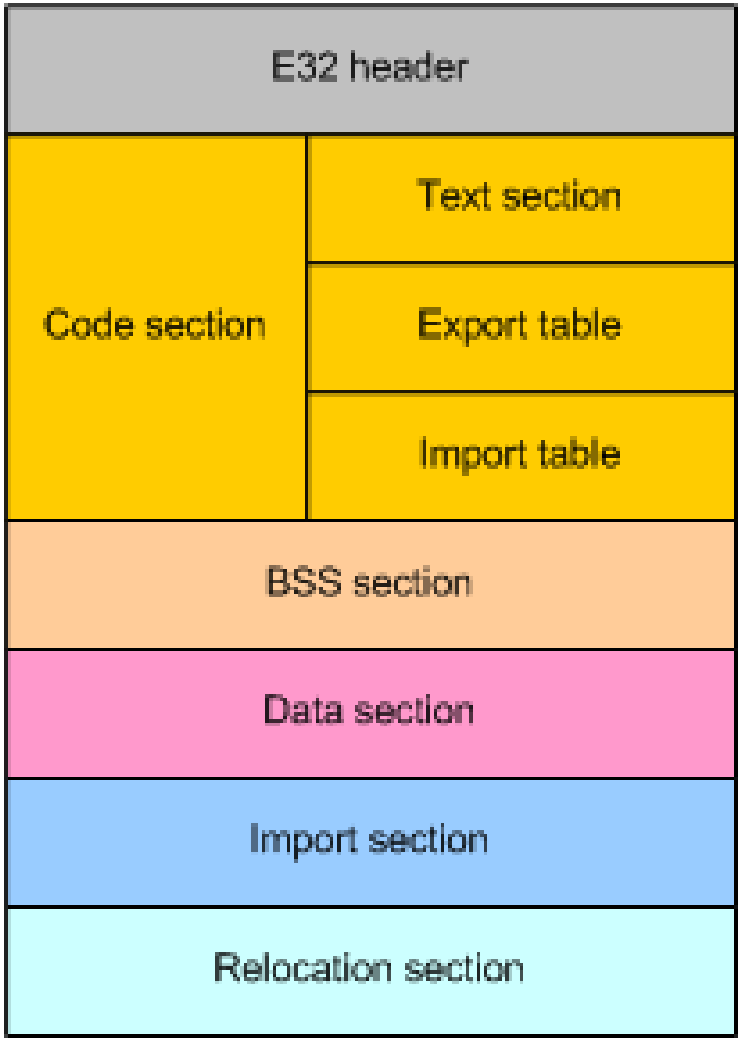
# Rodzaje znalezionych błędów

- **COFF, DBG**: Heap Buffer Overflow due to an Integer Underflow.
- **EPOC**: 4-byte Heap Buffer Overflow due to an Off-By-One in bounds checking.
- **DEX**: Heap Buffer Overflow due to Integer Overflow.
- **PEF:** Multiple Heap Buffer Overflows due to Integer Handling Problems.
- **PEF**: Heap Memory Corruption due to logical errors in memory management.
- **Many**: Heap Buffer Overflows due to logical errors in memory management.

# EPOC 4-byte Heap Overflow due to Off-By-One

- EPOC to prosty format plików wykonywalnych używany na systemie operacyjnym o tej samej nazwie.
  - Szerzej znany jako Symbian.
- Składa się z nagłówka, po którym następują sekcje pamięci.
  - Dane sekcji w pliku mogą być zapisane “czystym tekstem” lub skompresowane algorytmem DEFLATE lub BYTEPAIR.

# EPOC file format overview



# Nie tak szybko…

- Brak kodu źródłowego dla formatu EPOC w IDA SDK.
  - Pozostaje tylko inżynieria wsteczna pliku `epoc.ldw`.
- Bardzo uboga dokumentacja formatu, w szczególności algorytmów (de)kompresji.
- Jedyne sensowne źródła wiedzy to projekty GnuPoc [3] (SymbianOS SDK) i symbian-dump (publicznie dostępny kod Symbiana) [4]

# Dalsza analiza

- Po krótkiej analizie okazało się, że Hex-Rays wykorzystał kod dekompresji DEFLATE z paczki GnuPoc.
  - Jednak możemy czytać kod C! ☺
- Jak na dekompresję, kod jest napisany wyjątkowo dobrze.

# Krótkie ćwiczenie: znajdź błąd

```
void Huffman::InternalizeL(TBitInput& aInput,TUint32 aHuffman[],TInt aNumCodes)
...
{
...
  TUint32* p=aHuffman;
  const TUint32* end=aHuffman+aNumCodes;
...
      while (rl>0)
        {
        if (p>end)
          {
          Panic(EHuffmanCorruptFile);
          }
        *p++=last;
        --rl;
        }
...
```

# Krótkie ćwiczenie: znajdź błąd

```
void Huffman::InternalizeL(TBitInput& aInput,TUint32 aHuffman[],TInt aNumCodes)
...
{
...
  TUint32* p=aHuffman;
  const TUint32* end=aHuffman+aNumCodes;
...
      while (rl>0)
        {
        if (p>end)
          {
          Panic(EHuffmanCorruptFile);
          }
        *p++=last;
        --rl;
        }
...
```

# Off-By-One Error

- Warunek powinien brzmieć:

```
if (p >= end) {
```

- Błąd pozwala na 4-bajtowe (`sizeof(Tuint32)`) nadpisanie bufora o stałej długości 1316 bajtów.
- W pobliskim kodzie znajdowało się więcej pomyłek tego typu.

# Off-By-One Error

## 4 bajty wystarczą, by uszkodzić wewnętrzne struktury sterty Linuksa:

```
*** glibc detected *** ./idaq: free(): invalid pointer: 0x09dcecf8 ***
======= Backtrace: =========
/lib/i386-linux-gnu/libc.so.6(+0x75b12)[0xf5f41b12]
/opt/ida-6.6/loaders/epoc.llx(+0xce3c)[0xf55bde3c]
/opt/ida-6.6/loaders/epoc.llx(+0xd406)[0xf55be406]
/opt/ida-6.6/libida.so(+0x16c815)[0xf749b815]
/opt/ida-6.6/libida.so(load_nonbinary_file+0xde)[0xf749bb8e]
./idaq[0x80b7ff5]
./idaq[0x8134d06]
/opt/ida-6.6/libida.so(init_database+0x10d6)[0xf73d1c06]
./idaq[0x8094a1e]
./idaq[0x8284848]
./idaq[0x828497f]
./idaq[0x809605b]
./idaq[0x8096133]
./idaq[0x80963f7]
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3)[0xf5ee54d3]
./idaq[0x807ed11]
```

# PEF Multiple Heap Buffer Overflows due to Integer Handling Problems.

- *Preferred Executable Format*
  - Format plików wykonywalnych używany wiele lat temu w systemie Mac OS.
  - Obecnie spotykany jedynie na platformie PowerPC pod kontrolą systemu BeOS.
- Kod źródłowy dostępny w IDA SDK.

# Funkcja process_loader_data

- Funkcja przetwarza dane adresowane przesunięciami zdefiniowanymi w nagłówku pliku.

```
//----------------------------------------------------------------------
static void process_loader_data(bytevec_t &ldrdata, const
qvector<pef_section_t> &sec)
{
  if ( ldrdata.size() < sizeof(pef_loader_t) )
    bad_loader_data();
  pef_loader_t &pl = *(pef_loader_t *)ldrdata.begin();
  swap_pef_loader(pl);
  pef_library_t *pil = (pef_library_t *)(&pl + 1);
  uint32 *impsym = (uint32 *)(pil + pl.importLibraryCount);
  pef_reloc_header_t *prh =
                (pef_reloc_header_t *)(impsym + pl.totalImportedSymbolCount);
  char *stable = (char *)(ldrdata.begin() + pl.loaderStringsOffset);
  uint16 *relptr = (uint16 *)(ldrdata.begin() + pl.relocInstrOffset);
  uint32 *hash = (uint32 *)(ldrdata.begin() + pl.exportHashOffset);
  uint32 hashsize = (1 << pl.exportHashTablePower);
  uint32 *keytable = hash + hashsize;
  pef_export_t *pe = (pef_export_t *)(keytable + pl.exportedSymbolCount);
```

# Struktura pef_loader_t

- Wszystkie *offsety* zdefiniowane są jako 32-bitowe, w pełni kontrolowane pola.

```
struct pef_loader_t
{
  sint32 mainSection;
  uint32 mainOffset;
  sint32 initSection;
  uint32 initOffset;
  sint32 termSection;
  uint32 termOffset;
  uint32 importLibraryCount;
  uint32 totalImportedSymbolCount;
  uint32 relocSectionCount;
  uint32 relocInstrOffset;
  uint32 loaderStringsOffset;
  uint32 exportHashOffset;
  uint32 exportHashTablePower;
  uint32 exportedSymbolCount;
};
```

- IDA Pro 6.6 dostępna jest wyłącznie w wersji 32-bitowej, co sprzyja powstawaniu błędów typu Integer Overflow podczas wykonywania operacji na wskaźnikach i offsetach.

# Co prawda weryfikacja jest...

## ... ale wykonywana źle.

```
static bool inside_or_end(const bytevec_t &vec, const void *end)
{
  const uchar *p = (const uchar *)end;
  return vec.begin() <= p && p <= vec.end();
}

...

  if ( !inside_or_end(ldrdata, impsym)
//     || !inside_or_end(ldrdata, impsym+pl.totalImportedSymbolCount)
     || !inside_or_end(ldrdata, prh)
     || !inside_or_end(ldrdata, prh+pl.relocSectionCount)
     || !inside_or_end(ldrdata, stable)
     || !inside_or_end(ldrdata, relptr)
     || !inside_or_end(ldrdata, keytable)
     || !inside_or_end(ldrdata, pe)
     || !inside_or_end(ldrdata, pe+pl.exportedSymbolCount)
     || pl.termSection != -1 && pl.termSection >= sec.size()
     || pl.initSection != -1 && pl.initSection >= sec.size()
     || pl.mainSection != -1 && pl.mainSection >= sec.size() )
  {
    bad_loader_data();
  }
```

Sprawdzane są wskaźniki wynikowe, wobec których mógł wcześniej wystąpić Integer Overflow.

# W konsekwencji bufor się przepełnia

## A aplikacja wysypuje:

```
Program received signal SIGSEGV, Segmentation fault.
0xf5f3a75a in ?? () from /opt/ida-6.6/loaders/pef.llx
(gdb) where
#0  0xf5f3a75a in ?? () from /opt/ida-6.6/loaders/pef.llx
#1  0xf7d7c815 in ?? () from /opt/ida-6.6/libida.so
#2  0xf7d7cb8e in load_nonbinary_file () from /opt/ida-6.6/libida.so
[...]
(gdb) x/10i $eip
=> 0xf5f3a75a:    mov    0x8(%eax),%ecx
   0xf5f3a75d:    mov    %edx,0x4(%eax)
[...]
(gdb) info reg
eax            0x8a6aff8    145141752
ecx            0x0    0
edx            0x0    0
[...]
```

# Więcej błędów

- W `process_loader_data` znajdowały się 4 *Integer Overflowy* prowadzące do *memory corruption*.
- Dla spostrzegawczych: w pokazanym wcześniej listingu znajduje się jeszcze jeden błąd logiczny.
  - Wskaźniki na semantycznie odrębne struktury w pliku wyliczane są na podstawie jednego bufora wejściowego.
  - Struktury mogą (a nie powinny) nachodzić na siebie w pamięci.
  - Modyfikacja jednej może prowadzić do nieoczekiwanej zmiany innej.

# BADMEMSIZE

- IDA używa wewnętrznie własnych kontenerów pamięci dla list i buforów:
  - `qvector`
  - `bytevec_t` (dziedziczące po `qvector`)
- Klasy te udostępniają pewne standardowe metody, takie jak:
  - `::growfill`
  - `::append`
  - `::reserve`
  - `::resize`

# Ochrona przed Integer Overflows

- Metody te wykrywają sytuacje przepełnienia zmiennych całkowitoliczbowych.
- Obsługiwały je jednak w dość niecodzienny sposób.
  - Zamiast zwrócić kod błędu lub rzucić wyjątek, ustawiały rozmiar wymaganego bufora na stałą `BADMEMSIZE`.
  - **Założenie: `malloc(BADMEMSIZE)` nigdy się nie powiedzie.**

# Przykład

```
bytevec_t &append(const void *buf, size_t sz)
  {
    if ( sz > 0 )
    {
      size_t cur_sz = size();
      size_t new_sz = cur_sz + sz;
      if ( new_sz < cur_sz )
        new_sz = BADMEMSIZE
      resize(new_sz);
      memcpy(begin() + cur_sz, buf, sz);
    }
    return *this;
  }
  bytevec_t &growfill(size_t sz, uchar filler=0)
  {
    if ( sz > 0 )
    {
      size_t cur_sz = size();
      size_t new_sz = cur_sz + sz;
      if ( new_sz < cur_sz )
        new_sz = BADMEMSIZE
      resize(new_sz, filler);
    }
    return *this;
  }
```

idasdk66/include/pro.h

# Więęęc… czym jest BADMEMSIZE?

```
#ifdef __X64__
#define BADMEMSIZE 0xDEADBEEFDEADBEEF
#else
#define BADMEMSIZE 0xDEADBEEF
#endif
```

idasdk66/include/pro.h

# Hmm…

- IDA jest dostępna tylko w wersji 32-bitowej, więc rozważamy stałą `0xDEADBEEF`.
- 0xDEADBEEF = 3 735 928 559
  - *Prawie* cała 32-bitowa przestrzeń adresowa.
  - No właśnie, “prawie”. Sprawdźmy, czy faktycznie alokacja o tym rozmiarze nigdy się nie powiedzie.

# Windows

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 04260000 | Private Data | 64 K | | | | | | |
| 04270000 | Private Data | 64 K | | | | | | |
| 04280000 | Private Data | 64 K | | | | | | |
| 04290000 | Private Data | 4,096 K | 3,344 K | 3,344 K | 2,988 K | 2,988 K | | |
| 04690000 | Private Data | 1,024 K | 8 K | 8 K | 8 K | 8 K | | |
| 04790000 | Mapped File | 4 K | 4 K | 4 K | 4 K | | 4 K | 4 K |
| 047A0000 | Shareable | 4 K | 4 K | | 4 K | | 4 K | 4 K |
| 047B0000 | Shareable | 4 K | 4 K | | 4 K | | 4 K | 4 K |
| 048C0000 | Private Data | 128 K | 4 K | 4 K | 4 K | 4 K | | |
| 04910000 | Private Data | 64 K | 12 K | 12 K | 12 K | 12 K | | |
| 04990000 | Thread Stack | 256 K | 28 K | 28 K | 8 K | 8 K | | |
| 049D0000 | Thread Stack | 256 K | 28 K | 28 K | 4 K | 4 K | | |
| 04B00000 | Thread Stack | 1,024 K | 24 K | 24 K | 4 K | 4 K | | |
| 04CB0000 | Thread Stack | 1,024 K | 12 K | 12 K | | | | |
| 54480000 | Image (ASLR) | 9,544 K | 9,544 K | 120 K | 9,544 K | 64 K | 9,480 K | |
| 608A0000 | Image (ASLR) | 884 K | 884 K | 16 K | 884 K | 8 K | 876 K | |
| 6FA70000 | Image (ASLR) | 1,656 K | 1,656 K | 24 K | 364 K | 12 K | 352 K | 96 K |
| 6FCD0000 | Image (ASLR) | 588 K | 588 K | 28 K | 588 K | 12 K | 576 K | |
| 6FD70000 | Image (ASLR) | 528 K | 528 K | 20 K | 528 K | 12 K | 516 K | |
| 6FE00000 | Image (ASLR) | 2,904 K | 2,904 K | 36 K | 2,904 K | 20 K | 2,884 K | |
| 700E0000 | Image (ASLR) | 420 K | 420 K | 12 K | 420 K | 8 K | 412 K | |
| 70150000 | Image (ASLR) | 1,788 K | 1,788 K | 128 K | 1,788 K | 64 K | 1,724 K | |

# Mac OS X

```
                                        …
CG image                0ad39000-0ad45000 [   48K] rw-/rwx SM=PRV
OpenGL GLSL             0aec8000-0aeca000 [    8K] rw-/rwx SM=ZER
MALLOC_LARGE (freed)    0aeca000-0af0b000 [  260K] rw-/rwx SM=PRV
MALLOC_LARGE (freed)    0af8c000-0afcd000 [  260K] rw-/rwx SM=PRV
CG image                0afed000-0b000000 [   76K] rw-/rwx SM=PRV
MALLOC_SMALL            0b000000-0b800000 [ 8192K] rw-/rwx SM=COW
__DATA                  0bfbc000-0bffe000 [  264K] rw-/rwx SM=COW
__DATA                  0bffe000-0c023000 [  148K] rw-/rwx SM=PRV
__DATA                  0c12f000-0c145000 [   88K] rw-/rwx SM=ZER
MALLOC_LARGE (freed)    0c246000-0c287000 [  260K] rw-/rwx SM=PRV
CG image                0c367000-0c3cb000 [  400K] rw-/rwx SM=PRV
VM_ALLOCATE             0c3cb000-0c3ce000 [   12K] rw-/rwx SM=PRV
Memory Tag 241          0c3cf000-0c3e5000 [   88K] rw-/rwx SM=COW
MALLOC_TINY             0c400000-0c500000 [ 1024K] rw-/rwx SM=PRV
MALLOC_SMALL (freed)    0c800000-0d000000 [ 8192K] rw-/rwx SM=COW
__DATA                  488c5000-49599000 [ 12.8M] rw-/rwx SM=COW
__DATA                  49599000-495a8000 [   60K] rw-/rwx SM=PRV
__DATA                  8ff08000-8ff0a000 [    8K] rw-/rwx SM=PRV
__DATA                  8ff0a000-8ff32000 [  160K] rw-/rwx SM=COW
__DATA                  a032f000-a0330000 [    4K] rw-/rwx SM=COW
__DATA                  a0330000-a0331000 [    4K] rw-/rwx SM=COW
                                        …
```

# Linux

```
...
08048000-083de000 r-xp 00000000 fc:01 5250367
083de000-083e1000 r--p 00395000 fc:01 5250367
083e1000-083e6000 rw-p 00398000 fc:01 5250367
083e6000-083f8000 rw-p 00000000 00:00 0
08e80000-092e6000 rw-p 00000000 00:00 0 [heap]
f5649000-f56c9000 rw-s 00000000 00:04 12582916
f56c9000-f56d4000 r-xp 00000000 fc:01 6817133
f56d4000-f56d5000 r--p 0000a000 fc:01 6817133
f56d5000-f56d6000 rw-p 0000b000 fc:01 6817133
f56eb000-f56ec000 rw-p 00000000 00:00 0
...
```

# Linux

## BINGO!

- W przestrzeni adresowej 32-bitowego procesu na 64-bitowym systemie Linux istnieje “luka” o rozmiarze ~0xEC000000 bajtów!
- W konsekwencji `malloc(BADMEMSIZE)` powiedzie się, przypisując rozmiar `0xDEADBEEF` alokacji, której tak naprawdę przysługuje rozmiar `≥ 0x100000000`.

# Konsekwencje

- Błąd znajduje się w wielu klasach odpowiedzialnych za zarządzanie pamięcią, używanych przez główny silnik IDA oraz obsługę ponad połowy formatów wejściowych.
- W raporcie dla Hex-Rays przedstawiony został crash sprowokowany przy pomocy formatu Mach-O.

# BADMEMSIZE: crash w Mach-O

```
Program received signal SIGSEGV, Segmentation fault.
0xf4d98168 in ?? () from /opt/ida-6.6/loaders/macho.llx
(gdb) where
#0  0xf4d98168 in ?? () from /opt/ida-6.6/loaders/macho.llx
#1  0xf4db4aaf in ?? () from /opt/ida-6.6/loaders/macho.llx
#2  0xf4db6fec in ?? () from /opt/ida-6.6/loaders/macho.llx
#3  0xf7d7c815 in ?? () from /opt/ida-6.6/libida.so
#4  0xf7d7cb8e in load_nonbinary_file () from /opt/ida-
   6.6/libida.so
...
(gdb) x/10i $eip
=> 0xf4d98168:    mov    (%edi,%edx,4),%ecx
   0xf4d9816b:    mov    %ecx,%eax
   0xf4d9816d:    mov    %ecx,%ebx
...
```

# Hex-Rays IDA Pro: podsumowanie

Krótko mówiąc: **są błędy i łatwo je znaleźć.**

Z perspektywy researchera: **warto się po nie schylić i zgłosić. ☺**

Z perspektywy użytkownika: **nie do końca bezpieczny software w przypadku analizy plików z niezaufanych źródeł (prawdopodobnie 90% use cases).**

# Czego jeszcze używamy do analizy?

**Maszyn wirtualnych!**

# Bezpieczeństwo VM

- Maszyny wirtualne używane w celu separacji potencjalnie szkodliwego kodu od właściwego środowiska pracy.
  - Założenie o całkowitym oddzieleniu środowisk wykonywalnych.

**Ale czy na pewno?**

# Przepływ danych

- Pomiędzy *guestem* a *hostem* wymieniana jest znaczna ilość informacji:
  - urządzenia wirtualne implementowane programowo przez VM (ekran, dysk twardy, karta sieciowa itp.)
  - komendy karty graficznej, akceleracja 3D.
  - tzw. *additions* – współdzielone foldery, drag-n-drop plików, współdzielony schowek, …

# Wektory ataku

- Każdy kanał komunikacji to potencjalny wektor ataku dla odseparowanej aplikacji.
  - Możliwość ataku sterownika pomocniczego VM, w celu wykonania kodu ring-0 (w *gueście*).
  - Możliwość ataku monitora VM, w celu wykonania kodu ring-3 (w *hoście*).
  - W przypadku wybitnych błędów (np. w CPU), możliwość ataku jądra hosta.
    - Przykład: luki w jądrach i hypervisorach związane z obsługą instrukcji SYSRET (Rafał Wojtczuk, 2012)

# Łatwość testowania

- Wiele maszyn wirtualnych jest całkowicie open-source, umożliwiając audyt (skomplikowanego) kodu źródłowego.
- Ponadto, kanały komunikacji są łatwym celem fuzzingu (np. VGA).

# Fuzzing VGA

DEMO

Losowe przykłady

# A JAK JEST W PRAKTYCE?

# Bezpieczeństwo VM: VirtualBox

## User-mode → kernel-mode EoP

- Tarjei Mandt, Oracle VirtualBox Integer Overflow Vulnerabilities, 2011 [5]
- Mateusz Jurczyk, Oracle VirtualBox Integer Overflow Vulnerabilities, 2012 [6]
- Matt Bergin, Oracle VirtualBox Guest Additions Arbitrary Write Privilege Escalation, 2014 [7]

# Bezpieczeństwo VM: VirtualBox

## Guest → host EoP

- Francisco Falcon, Breaking Out of VirtualBox through 3D Acceleration, Recon, 2014 [8]
- Florian Ledoux, Advanced Exploitation of VirtualBox 3D Acceleration VM Escape Vulnerability, VUPEN blog, 2014 [9]

# Bezpieczeństwo VM: VMWare

- Derek Soeder
  - dziesiątki błędów: CVE-2008-4279, CVE-2012-1516, CVE-2012-1517, CVE-2013-1406, CVE-2013-3519 i inne.
- Kostya Kortchinsky, CLOUDBURST: A VMware Guest to Host Escape Story, 2009 [10]
- Piotr Bania, VMware CloudBurst - VMware Guest to Host Escape Exploit, 2009 [11]
- Piotr Bania, Old vmware cloudburst exploit, 2012 [12]

# Bezpieczeństwo VM: Xen

## 74 poprawione błędy od początku 2013

| Advisory | Public release | Updated | Version | CVE(s) |
|---|---|---|---|---|
| XSA-108 | 2014-10-01 12:00 | 2014-10-01 12:02 | 4 | CVE-2014-7188 |
| XSA-107 | 2014-09-09 12:30 | 2014-09-11 10:07 | 2 | CVE-2014-6268 |
| XSA-106 | 2014-09-23 12:00 | 2014-( | | |
| XSA-105 | 2014-09-23 12:00 | 2014-( | | |
| XSA-104 | 2014-09-23 12:00 | 2014-( | | |
| XSA-103 | 2014-08-12 12:00 | 2014-( | | |
| XSA-102 | 2014-08-12 12:00 | 2014-( | | |
| XSA-101 | 2014-06-25 12:00 | 2014-( | | |
| XSA-100 | 2014-06-17 11:44 | 2014-( | | |
| XSA-99 | 2014-06-17 11:44 | 2014-( | | |
| XSA-98 | 2014-06-04 12:00 | 2014-( | | |
| XSA-97 | 2014-08-12 12:00 | 2014-( | | |
| XSA-96 | 2014-06-03 12:00 | 2014-( | | |
| XSA-95 | 2014-05-14 10:44 | 2014-( | | |
| XSA-94 | 2014-04-23 13:05 | 2014-( | | |
| XSA-93 | 2014-04-22 15:05 | 2014-( | | |
| XSA-92 | 2014-04-29 08:50 | 2014-( | | |
| XSA-91 | 2014-04-30 09:52 | 2014-( | | |
| XSA-90 | 2014-03-24 13:00 | 2014-( | | |
| XSA-89 | 2014-03-25 12:00 | 2014-( | | |
| XSA-88 | 2014-02-12 12:00 | 2014-( | | |
| XSA-87 | 2014-01-23 17:38 | 2014-( | | |
| XSA-86 | 2014-02-06 12:00 | 2014-( | | |
| XSA-85 | 2014-02-06 12:00 | 2014-( | | |
| XSA-84 | 2014-02-06 12:00 | 2014-( | | |
| XSA-83 | 2014-01-23 12:00 | 2014-( | | |
| XSA-82 | 2013-12-02 17:13 | 2014-02-19 16:54 | 4 | CVE-2013-6885 |
| XSA-81 | 2013-11-27 13:21 | | - | - |
| XSA-80 | 2013-12-10 12:00 | 2013-12-10 12:58 | 3 | CVE-2013-6400 |
| XSA-79 | 2013-11-27 13:20 | | - | - |
| XSA-78 | 2013-11-20 17:08 | 2013-11-21 11:32 | 2 | CVE-2013-6375 |
| XSA-77 | 2013-12-10 12:00 | 2013-12-10 12:58 | 3 | none (yet) assigned |
| XSA-76 | 2013-11-26 12:00 | 2013-11-26 17:02 | 3 | CVE-2013-4554 |
| XSA-75 | 2013-11-08 16:20 | 2013-11-11 11:42 | 2 | CVE-2013-4551 |
| XSA-74 | 2013-11-26 12:00 | 2013-11-26 17:02 | 3 | CVE-2013-4553 |
| XSA-73 | 2013-11-01 15:07 | 2013-11-04 13:15 | 3 | CVE-2013-4494 |
| XSA-72 | 2013-10-29 12:00 | 2013-10-29 15:39 | 3 | CVE-2013-4416 |
| XSA-71 | 2013-10-10 12:00 | 2013-10-10 12:28 | 2 | CVE-2013-4375 |
| XSA-70 | 2013-10-10 12:00 | 2013-10-10 12:22 | 2 | CVE-2013-4371 |
| XSA-69 | 2013-10-10 12:00 | 2013-10-10 12:22 | 2 | CVE-2013-4370 |
| XSA-68 | 2013-10-10 12:00 | 2013-10-10 12:22 | 2 | CVE-2013-4369 |
| XSA-67 | 2013-10-10 12:00 | 2013-10-10 12:22 | 2 | CVE-2013-4368 |
| XSA-66 | 2013-09-30 10:04 | 2013-09-30 10:04 | 3 | CVE-2013-4361 |
| XSA-65 | 2013-10-02 15:00 | 2013-10-02 16:23 | 2 | CVE-2013-4344 |
| XSA-64 | 2013-09-30 10:04 | 2013-09-30 10:04 | 3 | CVE-2013-4356 |
| XSA-63 | 2013-09-30 10:04 | 2013-09-30 10:04 | 3 | CVE-2013-4355 |
| XSA-62 | 2013-09-24 12:00 | 2013-09-25 08:23 | 2 | CVE-2013-1442 |
| XSA-61 | 2013-09-10 10:56 | 2013-09-11 12:13 | 2 | CVE-2013-4329 |
| XSA-60 | 2013-07-19 12:00 | 2014-02-19 16:54 | 6 | CVE-2013-2212 |
| XSA-59 | 2013-08-20 12:00 | 2013-08-20 12:07 | 4 | CVE-2013-3495 |
| XSA-58 | 2013-06-26 12:00 | 2013-06-26 13:18 | 2 | CVE-2013-1432 |
| XSA-57 | 2013-06-20 12:00 | 2013-06-26 10:37 | 4 | CVE-2013-2211 |
| XSA-56 | 2013-05-17 12:00 | 2013-05-17 15:44 | 2 | CVE-2013-2072 |
| XSA-55 | 2013-06-03 16:18 | 2013-06-20 10:26 | 5 | CVE-2013-2194 CVE-2013-2195 CVE-2013-2196 |
| XSA-54 | 2013-06-03 12:00 | 2014-06-03 12:23 | 4 | CVE-2013-2078 |
| XSA-53 | 2013-06-03 12:00 | 2013-06-03 16:18 | 3 | CVE-2013-2077 |
| XSA-52 | 2013-06-03 12:00 | 2013-06-03 16:18 | 3 | CVE-2013-2076 |
| XSA-51 | 2013-05-06 15:00 | 2013-05-06 21:18 | 2 | CVE-2013-2007 |
| XSA-50 | 2013-04-18 15:16 | 2013-04-18 15:16 | 1 | CVE-2013-1964 |
| XSA-49 | 2013-05-02 12:00 | 2013-05-02 14:27 | 2 | CVE-2013-1952 |
| XSA-48 | 2013-04-15 15:00 | 2013-04-15 15:00 | 2 | CVE-2013-1922 |
| XSA-47 | 2013-04-04 17:54 | 2013-04-04 17:54 | 1 | CVE-2013-1920 |
| XSA-46 | 2013-04-18 12:00 | 2013-04-18 13:35 | 3 | CVE-2013-1919 |
| XSA-45 | 2013-05-02 12:00 | 2013-05-02 13:54 | 2 | CVE-2013-1918 |
| XSA-44 | 2013-04-18 12:00 | 2013-04-18 13:50 | 3 | CVE-2013-1917 |
| XSA-43 | 2013-02-05 12:00 | 2013-02-05 12:59 | 2 | CVE-2013-0231 |
| XSA-42 | 2013-02-12 12:00 | 2013-02-13 16:49 | 2 | CVE-2013-0228 |
| XSA-41 | 2013-01-16 14:50 | 2013-01-17 12:17 | 2 | CVE-2012-6075 |
| XSA-40 | 2013-01-16 14:50 | 2013-01-16 14:50 | 1 | CVE-2013-0190 |
| XSA-39 | 2013-02-05 12:00 | 2013-02-05 12:59 | 2 | CVE-2013-0216 CVE-2013-0217 |
| XSA-38 | 2013-02-05 12:00 | 2013-02-15 11:40 | 3 | CVE-2013-0215 |
| XSA-37 | 2013-01-04 16:00 | 2013-01-04 16:00 | 1 | CVE-2013-0154 |
| XSA-36 | 2013-02-05 12:00 | 2013-02-21 11:05 | 4 | CVE-2013-0153 |
| XSA-35 | 2013-01-22 11:49 | 2013-01-23 18:28 | 4 | CVE-2013-0152 |
| XSA-34 | 2013-01-22 11:49 | 2013-01-22 11:49 | 2 | CVE-2013-0151 |
| XSA-33 | 2013-01-08 12:00 | 2013-01-11 17:10 | 3 | CVE-2012-5634 |
| XSA-32 | 2012-12-03 17:51 | 2012-12-03 17:51 | 4 | CVE-2012-5525 |
| XSA-31 | 2012-12-03 17:51 | 2012-12-03 17:51 | 3 | CVE-2012-5515 |
| XSA-30 | 2012-12-03 17:51 | 2012-12-03 17:51 | 4 | CVE-2012-5514 |
| XSA-29 | 2012-12-03 17:51 | 2012-12-03 17:51 | 3 | CVE-2012-5513 |
| XSA-28 | 2012-12-03 17:51 | 2012-12-03 17:51 | 3 | CVE-2012-5512 |
| XSA-27 | 2012-12-03 17:51 | 2013-01-17 12:17 | 5 | CVE-2012-5511 CVE-2012-6333 |
| XSA-26 | 2012-12-03 17:51 | 2012-12-03 17:51 | 3 | CVE-2012-5510 |

# Bezpieczeństwo VM: qemu

- Nelson Elhage, Virtunoid: A KVM Guest → Host privilege escalation exploit, 2011 [13]
- Dziesiątki błędów, głównie w implementacji emulowanych urządzeń.

# Bezpieczeństwo VM: qemu

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | CVE-2014-5263 | 119 | DoS Overflow +Priv Mem. Corr. | 2014-08-26 | 2014-08-27 | 6.8 |

vmstate_xhci_event in hw/usb/hcd-xhci.c in QEMU 1.6.0 does not terminate the list with the VMSTATE_END_O loop, and memory corruption) and possibly gain privileges via unspecified vectors.

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | CVE-2014-2894 | 189 | Exec Code Mem. Corr. | 2014-04-23 | 2014-05-10 | 7.2 |

Off-by-one error in the cmd_smart function in the smart self test in hw/ide/core.c in QEMU before 2.0 allows lo underflow and memory corruption.

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | CVE-2014-0150 | 189 | Exec Code Overflow | 2014-04-18 | 2014-05-10 | 4.9 |

Integer overflow in the virtio_net_handle_mac function in hw/net/virtio-net.c in QEMU 2.0 and earlier allows lo heap-based buffer overflow.

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | CVE-2013-4544 | 20 | DoS Exec Code | 2014-05-08 | 2014-05-09 | 4.9 |

hw/net/vmxnet3.c in QEMU 2.0.0-rc0, 1.7.1, and earlier allows local guest users to cause a denial of service o interrupt indices. NOTE: some of these details are obtained from third party information.

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | CVE-2013-4377 | 399 | DoS | 2013-10-11 | 2014-03-05 | 2.3 |

Use-after-free vulnerability in the virtio-pci implementation in Qemu 1.4.0 through 1.6.0 allows local users to c

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | CVE-2013-4375 | 399 | DoS | 2014-01-19 | 2014-03-05 | 2.7 |

The qdisk PV disk backend in qemu-xen in Xen 4.2.x and 4.3.x before 4.3.1, and qemu 1.1 and other versions unspecified vectors.

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | CVE-2013-4344 | 119 | Overflow +Priv | 2013-10-04 | 2014-03-05 | 6.0 |

Buffer overflow in the SCSI implementation in QEMU, as used in Xen, when a SCSI controller has more than 2 LUNS command.

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | CVE-2013-2007 | 264 | | 2013-05-21 | 2013-08-22 | 6.9 |

The qemu guest agent in Qemu 1.4.1 and earlier, as used by Xen, when started in daemon mode, uses weak

| | | | | | | |
|---|---|---|---|---|---|---|
| 9 | CVE-2012-6075 | 119 | DoS Exec Code Overflow | 2013-02-12 | 2014-04-19 | 9.3 |

Buffer overflow in the e1000_receive function in the e1000 device driver (hw/e1000.c) in QEMU 1.3.0-rc2 and of service (guest OS crash) and possibly execute arbitrary guest code via a large packet.

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | CVE-2012-3515 | 20 | +Priv | 2012-11-23 | 2014-03-05 | 7.2 |

Qemu, as used in Xen 4.0, 4.1 and possibly other products, when emulating certain devices with a virtual cons triggers the overwrite of a "device model's address space."

# Błędy po stronie użytkownika VM

- Współdzielone foldery
  - czy ktoś kiedykolwiek odpalał programy znajdujące się w *shared folder* VMki wykonującej niezaufany kod?
    - DLL hijacking lub po prostu zainfekowanie EXEka znajdującego się w katalogu.
- Brak separacji sieciowej, dostęp do systemu plików hosta przez network shares.
- Inne zapomniane lub nieoczywiste kanały komunikacji.

# A SKORO JESTEŚMY JUŻ PRZY MASZYNACH WIRTUALNYCH…

# Inne drogi ucieczki

- Jeśli w celu analizy malware używamy opcji zdalnego debuggowania jądra, tworzymy kolejny kanał komunikacji, który może stać się celem ataku.
- Bezpośrednim celem jest w tym przypadku WinDbg.

# Protokół KDCOM

- WinDbg rozmawia z jądrem guesta przy pomocy protokołu KDCOM.
  - *Kernel Debugging Communication*
  - Prosty format pakietowy o nagłówku:

```
typedef struct _KD_PACKET_HEADER {
  DWORD Signature;
  WORD PacketType;
  WORD DataSize;
  DWORD PacketID;
  DWORD Checksum;
  BYTE PacketBody[1];
} KD_PACKET_HEADER, *PKD_PACKET_HEADER;
```

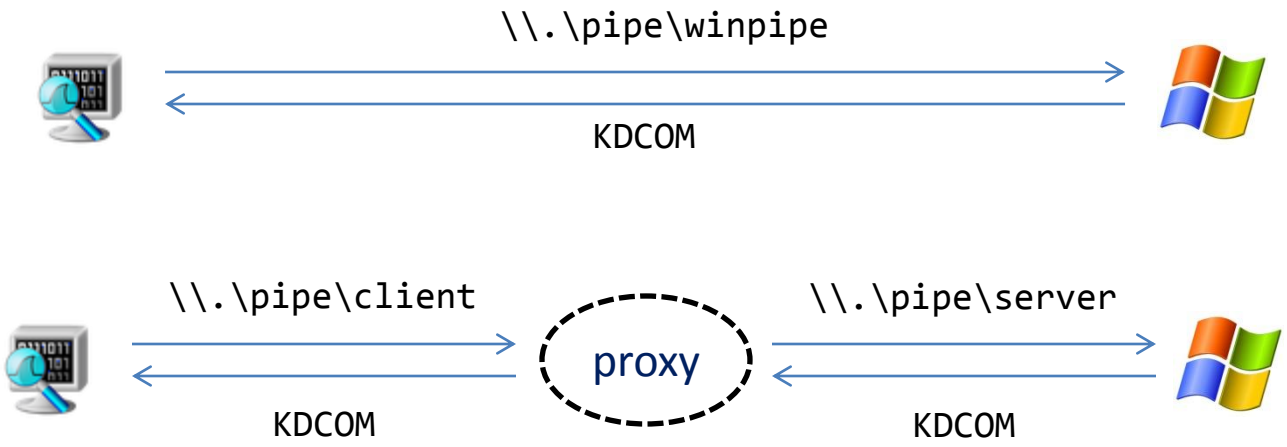
  - Ponad 50 wspieranych typów wiadomości.

# Protokół KDCOM

- Dogłębnie opisany w kilku źródłach:
  - Kernel and remote debuggers [14]
  - KD extension DLLs & KDCOM protocol [15]
- Doczekał się również niezależnych implementacji.
  - SYSPROGS VirtualKD – aplikacja przyspieszająca sesje debugowania po porcie COM dla VirtualBox i VMWare
  - SecureWorks Wind Pill – perlowa implementacja protokołu.

# Fuzzing KDCOM

- Automatyzacja testowania poprzez stworzenie *proxy* modyfikującego poprawną komunikację między WinDbg a kernelem.



# Fuzzing KDCOM

- Po wielu próbach nie udało się sprowokować żadnego błędu w WinDbg, związanego bezpośrednio z obsługą protokołu.
- Crashe pojawiły się natomiast podczas mutacji *ciała* wiadomości przesyłanych z wirtualnej maszyny do debuggera.

# Cóż takiego parsuje WinDbg?

Pliki PE, oczywiście!

# Przetwarzanie plików PE

- WinDbg obsługuje symbole.
  - publiczne, w tabeli eksportów EXEków.
  - prywatne, znajdujące się w odpowiadających plikach PDB.
- W celu uzyskania potrzebnych informacji, trzeba przetworzyć pliki wykonywalne z *guesta*.
  - WinDbg zczytuje z pamięci gościa pamięć modułów, a następnie przekazuje je do DbgHelp API odpowiedzialnych za obsługę symboli.

# Kilka słów o DbgHelp

- Pomocnicza biblioteka dostarczana przez Microsoft.
- Oferuje wysokopoziomowe funkcjonalności przydatne w debuggerach.
  - Obsługa symboli.
  - Tworzenie dumpów procesu.
  - Odczytywanie stosu wywołań.

# Kilka słów o DbgHelp

- Parsuje skomplikowane formaty plików takie jak PE lub PDB.
- Jakość kodu jest FATALNA, praktycznie brak jakiejkolwiek walidacji danych wejściowych.

} "ŚWIETNIE"!

# Przykładowe podatności DbgHelp

- Wiele *out-of-bounds reads* przez brak walidacji pól struktur PE, np. `IMAGE_FILE_HEADER.NumberOfSections`.
- *Arbitrary read* podczas obsługi *Export Table*.
- *Out-of-bounds read* przez brak walidacji numerów porządkowych w *Import Table*.
- *Arbitrary read* podczas obsługi *COFF Symbol Table*.
- *Out-of-bounds write* przez brak walidacji numerów porządkowych w *Export Table*.
- *Integer Overflow* podczas dynamicznej alokacji tablicy wewnętrznej do obsługi *Export Table*.

Wykonując kod z poziomu jądra w wirtualnej maszynie możemy doprowadzić do naruszenia integralności pamięci WinDbg, i potencjalnie uruchomić kontrolowany kod.

Research opisany dokładnie w 2010 na blogu [16]

# WinDbg i błędy logiczne

- Dokładnie w tym samym czasie Alex Ionescu odkrył, że protokół KDCOM pozwala na wykonanie dowolnej komendy WinDbg z poziomu guesta.
- Komenda `.shell` umożliwia uruchomienie dowolnego programu z kontrolowanymi parametrami.
- W efekcie uzyskujemy w 100% stabilne wykonanie kontrolowanego kodu na hoście z poziomu guesta.

# WinDbg i błędy logiczne

Rozwiązanie: uruchamiać windbg.exe z flagą **–secure**.

*Secure Mode* wyłącza w debuggerze wszelkie opcje umożliwiające mu ingerowanie w system, na którym został uruchomiony.

# A TO NIE KONIEC!

# Wireshark

## Istna skarbnica błędów bezpieczeństwa w dissectorach formatów.

| Rok | Liczba biuletynów |
|---|---|
| 2014 | 19 |
| 2013 | 68 |
| 2012 | 40 |
| 2011 | 19 |
| 2010 | 14 |
| 2009 | 9 |
| 2008 | 7 |
| 2007 | 3 |
| 2006 | 3 |

https://www.wireshark.org/security/

# Pozostałe narzędzia

Wszelkiego rodzaju dissectory, mniej znane  deasemblery etc. są najczęściej równie podatne.

Po prostu poświęca się im mniej uwagi, przez co są najczęściej przetestowane gorzej niż szeroko znane aplikacje.

Przykład: wspomniany na początku Sothink SWF Decompiler.

# JAKI Z TEGO MORAŁ?

# Czy obecnie malware aktywnie atakuje analityków lub systemy detekcji?

Chyba* nie.

* a przynajmniej ja o tym nie wiem.

# A czy mogłyby?

Zdecydowanie tak.

# Wnioski?

Oczywiście nie w tym rzecz, żeby przestać używać wspomnianych narzędzi. ☺

Dobrze jednak stosować podstawowe praktyki bezpieczeństwa – choć wszystko zależy od sytuacji i modelu zagrożenia.

# Propozycje

- Używać nowych systemów operacyjnych
  - każdy program jest znacznie bezpieczniejszy na Windows 8.1 niż na Windows Vista.
  - co jest swoją drogą "winą" Microsoftu, ale to temat na osobną dyskusję.
- Zmniejszać *attack surface*
  - włączanie jedynie niezbędnie potrzebnych urządzeń w VM.
  - korzystanie z *additions* i innych dodatkowych technologii (np. akceleracji 3D) wyłącznie w razie potrzeby.
  - w ogólności: redukowanie do minimum liczby linii kodu, które przetwarzają niezaufane dane.

# Propozycje

## Zwiększać ilość poziomów separacji.

Czy prawdopodobne jest, że ktoś stworzy stabilny exploit na naszą wersję IDA Pro?

**Jasne.**

# Propozycje

## Zwiększać ilość poziomów separacji.

A czy prawdopodobne jest, że ktoś zaatakuje host exploitem pod naszą wersję IDA Pro, odpaloną w sandboksie działającym w okrojonej wersji maszyny wirtualnej?

**Możliwe, ale nieprawdopodobne.**

Do przemyślenia.

# The end.

j00ru.vx@gmail.com

http://j00ru.vexillium.org/

@j00ru

# Materiały


[1] http://mincore.c9x.org/breaking_av_software.pdf
[2] https://technet.microsoft.com/en-us/library/security/2974294.aspx
[3] https://github.com/mstorsjo/gnupoc-package
[4] http://sourceforge.net/projects/symbiandump/
[5] http://mista.nu/blog/2011/07/19/oracle-virtualbox-integer-overflow-vulnerabilities/
[6] http://www.oracle.com/technetwork/topics/security/cpujan2012-366304.html
[7] https://www.korelogic.com/Resources/Advisories/KL-001-2014-001.txt
[8] http://recon.cx/2014/slides/Breaking_Out_of_VirtualBox_through_3D_Acceleration-Francisco_Falcon.pdf
[9] http://www.vupen.com/blog/20140725.Advanced_Exploitation_VirtualBox_VM_Escape.php
[10] http://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloudburst-SLIDES.pdf
[11] http://blog.piotrbania.com/2009/09/vmware-cloudburst-vmware-guest-to-host.html
[12] http://blog.piotrbania.com/2012/07/old-vmware-cloudburst-exploit.html
[13] https://media.blackhat.com/bh-us-11/Elhage/BH_US_11_Elhage_Virtunoid_WP.pdf
[14] http://www.developerfusion.com/article/84367/kernel-and-remote-debuggers/
[15] http://articles.sysprogs.org/kdvmware/kdcom.shtml
[16] http://j00ru.vexillium.org/?p=405