Fakty, mity i przemyślenia na temat offensive security

Mateusz "j00ru" Jurczyk

Sekurak Hacking Party, Kraków 2015

O prelegencie

- **Zawodowo:** *bughunter*
- **Hobbystycznie:** *bughunter + …*

O czym będzie

- Anegdoty.
- Filozoficzne przemyślenia.
- Doświadczenia nabyte podczas pracy w branży.
- Pewnie trochę narzekania.

Wszystkie przedstawione opinie są moje i tylko moje.

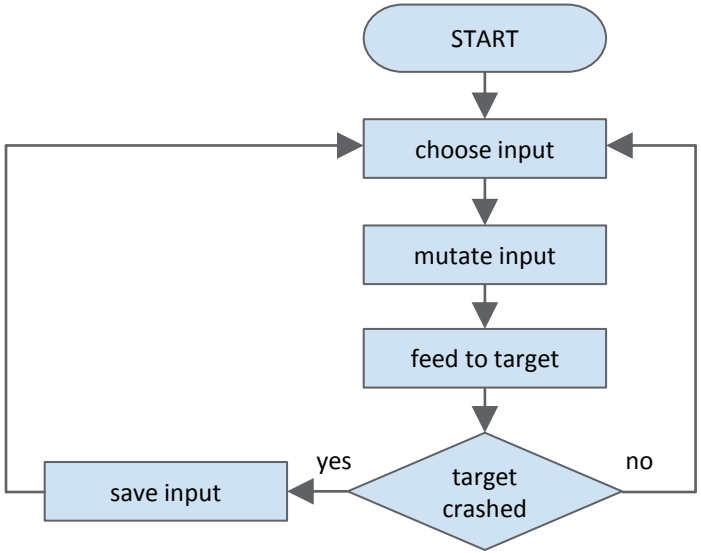
Offensive security, czyli... ?

Fuzzing

Fuzz testing *or* ***fuzzing*** *is a software testing technique, often automated or semi-automated, that involves providing invalid, unexpected, or random data to the inputs of a computer program.*

http://en.wikipedia.org/wiki/Fuzz_testing

Na schemacie



Easy to learn, hard to master.

Hard to master?

- Jak generowane są pliki wejściowe?
- Jaki jest bazowy zestaw plików wejściowych?
- W jaki sposób modyfikujemy wejście?
- Z jaką dokładnością wykrywamy błędy?
- Czy podejmujemy decyzje na podstawie zachowania programu dla poprzednich testów?
- W jaki sposób minimalizujemy interesujące zestawy testowe?
- W jaki sposób rozpoznajemy unikalne błędy?
- A co, jeśli aplikacja wymaga interakcji?
- A co, jeśli aplikacja *crashuje* się cały czas w jednym miejscu?
- A co, jeśli w fuzzowanym formacie znajdują się sumy kontrolne?

...

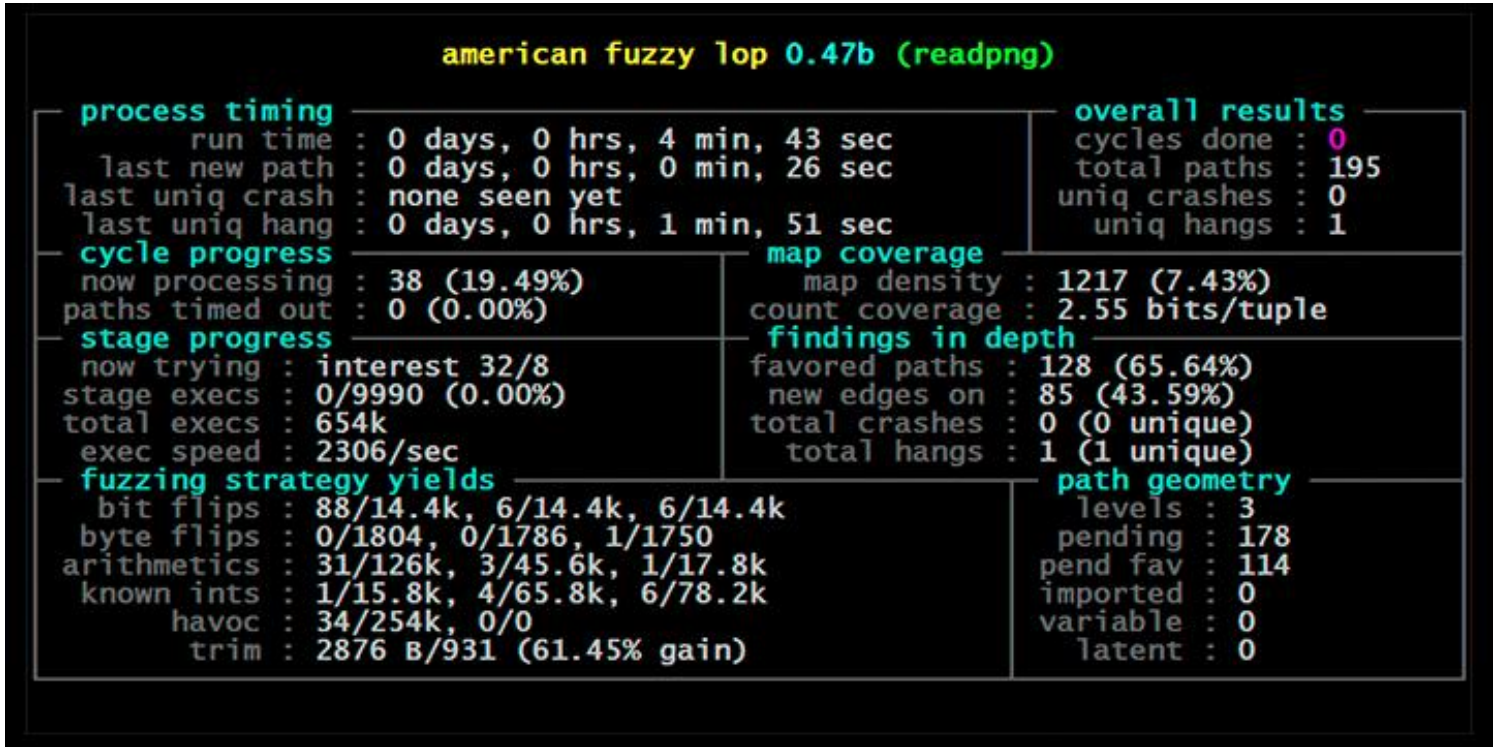
Niemniej, *easy to learn*

- Nie trzeba rozwiązać wszystkich powyższych problemów, żeby zacząć fuzzować.
- Właściwie wystarczy napisać na szybko programik na <1000 linii w dowolnym języku programowania.
- Czasem nawet i to zbyt wiele.

Istnieją gotowce

- American Fuzzy Lop
- Honggfuzz
- Peach
- Radamsa
- cross_fuzz, ref_fuzz
- SDL MiniFuzz File Fuzzer
- …

Efekt pralki



Wszyscy fuzzują.

Łatwo się rozleniwić

1. Napisz trywialny fuzzer w kilka dni (ew. ściągnij gotowy)
2. Skompiluj / skonfiguruj target
3. Uruchom fuzzer
4. Błędy same się znajdują
5. …
6. PROFIT!

SECURITY IS EASY

LET'S GO SHOPPING

Cele fuzzowania

1. Odkrycie podatności, żeby je wykorzystać / pobawić się / sprzedać.
2. Odkrycie podatności, żeby czegoś dowieść.
   - ustalanie polityki bezpieczeństwa w domu, firmie itp.

Mentalne skróty

1. **Leci fuzzing → znajdują się crashe → są błędy → software nie jest bezpieczny.**

2. Leci fuzzing → nie ma crashy → … ?

Mentalne skróty

1. Leci fuzzing → znajdują się crashe → są błędy → software nie jest bezpieczny.

2. **Leci fuzzing → nie ma crashy → … ?**

Mentalne skróty

1. Leci fuzzing → znajdują się crashe → są błędy → software nie jest bezpieczny.

2. **Leci fuzzing → nie ma crashy → nie ma błędów → software jest bezpieczny?**

NOPE

Fuzzer nie jest wyrocznią.

Dlaczego?

Fuzzery są (najczęściej) losowe.

Dlaczego?

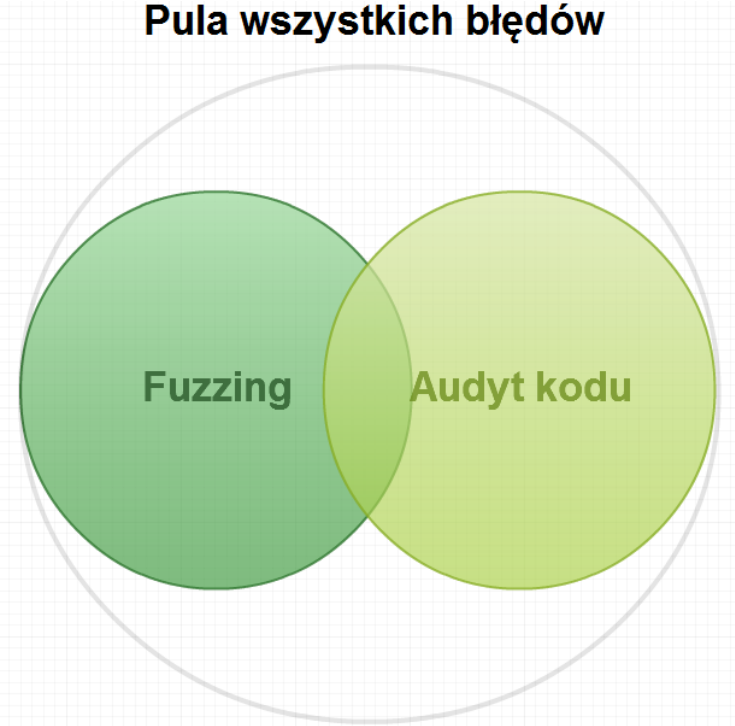
Ludzie nie przykładają się / popełniają błędy.

Dlaczego?

Formaty plików i ich implementacje drastycznie się różnią.

a fuzzery są zazwyczaj uniwersalne

Dlaczego?



Dlaczego?

```
char buffer[8];
if (input.magic == 0xbaadc0de) {
   strcpy(buffer, input.data);
}
```

Dlaczego?

“Fuzz clean” ≠ bug clean

Ale… ale…

- A co jeśli:
  - wiele różnych osób fuzzowało dany software,
  - z motywacją znalezienia błędów (żeby się pobawić / sprzedać itd.),
  - na prawdopodobnie różne sposoby,
  - przez długi okres czasu?

Core software

OK	acl	One day of afl/asan fuzzing turned up nothing.
OK	attr	One day of afl/asan fuzzing (2.4.47) turned up nothing
WIP	binutils	Multiple issues found in executable parsers by various people, upstream is actively working on fixing them. Has different independent exec parsers (libbfd, readelf). bug #17512 bug #17531
OK	bzip2	Received some fuzzing in the past. Requires checksum disabling [patch]
Stale	cpio	Open (likely RCE) bugs, last release quite old, unclear state [1]
Stale	dc/bc	Issues reported in November 2014, no release for many years, but still active developers.
OK	diffutils	Long afl-fuzzing turned up nothing
OK	expat	Seems robust with preliminary fuzzing.
OK	file/libmagic	Multiple issues were fixed in 5.21, e.g. [1], more issues found, now fixed in 5.22 [2] [3]
OK	freetype	Preliminary afl/asan fuzzing turned up nothing. It only ships very limited command line tools, not really fuzzing-friendly.
OK	gettext	Latest release (0.19.4) fixes several known issues, preliminary afl/asan fuzzing turned up no new issues [1]
WIP	giflib	giflib itself is pretty solid, however the tools shipped with it already expose memory bugs without any fuzzed input. [1] ([2] [3]) [4]
OK	gnupg	Various issues were found through fuzzing, latest versions 2.1.2, 2.0.27 and 1.4.19 should fix all of them TFPA 001/2014 TFPA 001/2015
OK	gzip	Has likely seen many reviews over the years. Fuzzing requires disabling CRC checks.
OK	ImageMagick	In the past it was pretty easy to fuzz bugs in imagemagick, but after some review by Google most of them have been fixed and these days there are at least no more trivial to find fuzzing issues. GraphicsMagick is a fork of ImageMagick, therefore issues often apply to both.
Stale	Info-ZIP	Multiple issues not fixed in a release [1] [2]
OK	less	No known issues, has been fuzzed in the past, communication with developers was not optional, see [CVE-2014-9488]

Limitation

Please be aware that this data has certain limitations and is not suitable as an overall assesment of software security.

- The very nature of such data is that it's rapidly changing. The information may be outdated.
- There is an infinite number of strategies for fuzzing. An "OK" does not mean that further fuzzing with different inputs, different tools or different strategies won't turn up further issues.
- Fuzzing only finds a certain subset of security issues. There is a vast number of security issues that can never be exposed by fuzzing. It can therefore only be a small part of a wider security strategy.

Patrz poprzednie punkty.

Zasadniczy problem

- *Fuzzing* nie motywuje do naukowego podejścia do tematu.
- Fundamentalne pytania, na które prawie nikt nie zna odpowiedzi:
  - jaki % wszystkich wykonywalnych ścieżek obejmuje nasz fuzzing?
  - jaki % istotnie różnych stanów programów obejmuje nasz fuzzing?
  - jaki jest przyrost pokrycia?
  - czy używana konfiguracja / algorytmy mutacji są efektywne / optymalne?
  - …

Przemyślenie

Fuzzing NIE powinien być używany w dyskusjach jako argument za bezpieczeństwem oprogramowania.

Przemyślenie

Jak najbardziej w celu zilustrowania braku bezpieczeństwa w aplikacji.

(np. żeby przekonać developerów do sandboxingu)

Przemyślenie

Fuzzing to wciąż bardzo rozwojowa dziedzina.

Póki co (prawie) wszyscy robią to samo.

Wracając do szukania błędów...

Co robi większość?

- Uruchamiają ogólnodostępne fuzzery.
- Piszą i uruchamiają swoje:
  - Na losowych, kiepsko wyselekcjonowanych samplach z Internetu.
  - Prosty bitflipping.
  - Nieinstrumentowany software.
  - Bez informacji o pokryciu kodu.
  - Na jednej maszynie.
  - …

Podstawowe pytanie:

czy można to zrobić lepiej?

Bardziej realistycznie:

czy "ja" mogę to zrobić lepiej?


BE NICE TO FAT PEOPLE
ONE DAY THEY MIGHT SAVE YOUR LIFE

A jest co poprawiać...

Hard to master?

- Jak generowane są pliki wejściowe?
- Jaki jest bazowy zestaw plików wejściowych?
- W jaki sposób modyfikujemy wejście?
- Z jaką dokładnością wykrywamy błędy?
- Czy podejmujemy decyzje na podstawie zachowania programu dla poprzednich testów?
- W jaki sposób minimalizujemy interesujące zestawy testowe?
- W jaki sposób rozpoznajemy unikalne błędy?
- A co, jeśli aplikacja wymaga interakcji?
- A co, jeśli aplikacja *crashuje* się cały czas w jednym miejscu?
- A co, jeśli w fuzzowanym formacie znajdują się sumy kontrolne?

...

Jeśli “tak”, to poprzednie wyniki w zasadzie nie mają znaczenia.

Podejście 1

“Skoro ktoś już fuzzował X i znalazł błędy, to na pewno znalazł wszystkie – nie warto na to dalej patrzeć.”

Podejście 2

“Skoro program X jest tak dziurawy, że nawet osobie Y udało się znaleźć jakieś błędy, to o wiele więcej czeka tam na mnie!”

Przemyślenie

- Wybór podejścia oczywiście zależy od konkretnego targetu.
- *Podejście 1* nigdy nie wygeneruje żadnych błędów.
  - ale też nie dopuści do straconego czasu.
- *Podejście 2* działa w praktyce zaskakująco dobrze.
  - jeśli idzie w parze z wiedzą i pewnością siebie.

Przykład: czcionki

- Wybitnie atrakcyjny wektor ataku
  - wiele różnych formatów.
  - bardzo skomplikowane formaty (strukturalnie i semantycznie).
  - trudne do całkowicie poprawnego zaimplementowania.
  - większość parserów napisana w natywnych językach programowania (C/C++).
  - zdalny wektor
    - dokumenty z dołączonymi czcionkami
    - strony internetowe z dołączonymi czcionkami
    - każdy klient używający GDI do renderingu niezaufanych czcionek
  - łatwość exploitacji
    - kształty liter w TrueType / OpenType opisywane przez programy specjalnych maszyn wirtualnych

Skoro tak, to na pewno wszystkie błędy zostały już dawno zgłoszone ...

... dajmy na to, w Windowsie.

Microsoft Security Bulletin MS06-002 - Critical

This topic has not yet been rated - Rate this topic

Vulnerability in Embedded Web Fonts Could Allow Remote Code Execution (908519)

Published: January 10, 2006

Microsoft Security Bulletin MS09-029 - Critical

This topic has not yet been rated - Rate this topic

Vulnerabilities in the Embedded OpenType Font Engine Could Allow Remote Code Execution (961371)

Published: July 14, 2009 | Updated: August 25, 2009

Microsoft Security Bulletin MS10-001 - Critical

This topic has not yet been rated - Rate this topic

Vulnerability in the Embedded OpenType Font Engine Could Allow Remote Code Execution (972270)

Published: January 12, 2010 | Updated: January 19, 2011

Microsoft Security Bulletin MS10-037 - Important

This topic has not yet been rated - Rate this topic

Vulnerability in the OpenType Compact Font Format (CFF) Driver Could Allow Elevation of Privilege (980218)

Published: June 08, 2010

Microsoft Security Bulletin MS10-076 - Critical

This topic has not yet been rated - Rate this topic

Vulnerability in the Embedded OpenType Font Engine Could Allow Remote Code Execution (982132)

Published: October 12, 2010

Microsoft Security Bulletin MS10-078 - Important

This topic has not yet been rated - Rate this topic

Vulnerabilities in the OpenType Font (OTF) Format Driver Could Allow Elevation of Privilege (2279986)

Published: October 12, 2010

Microsoft Security Bulletin MS10-091 - Critical

This topic has not yet been rated - Rate this topic

Vulnerabilities in the OpenType Font (OTF) Driver Could Allow Remote Code Execution (2296199)

Published: December 14, 2010

Ufff... sporo tego, na pewno już wszystko znaleźli!

Microsoft Security Bulletin MS11-032 - Critical

This topic has not yet been rated - Rate this topic

Vulnerability in the OpenType Compact Font Format (CFF) Driver Could Allow Remote Code Execution (2507618)

Published: April 12, 2011

Microsoft Security Bulletin MS11-087 - Critical

This topic has not yet been rated - Rate this topic

Vulnerability in Windows Kernel-Mode Drivers Could Allow Remote Code Execution (2639417)

Published: December 13, 2011

Microsoft Security Bulletin MS13-053 - Critical

This topic has not yet been rated - Rate this topic

Vulnerabilities in Windows Kernel-Mode Drivers Could Allow Remote Code Execution (2850851)

Published: July 09, 2013

Microsoft Security Bulletin MS13-054 - Critical

2 out of 3 rated this helpful - Rate this topic

Vulnerability in GDI+ Could Allow Remote Code Execution (2848295)

Published: July 09, 2013 | Updated: December 16, 2013

Microsoft Security Bulletin MS13-060 - Critical

This topic has not yet been rated - Rate this topic

Vulnerability in Unicode Scripts Processor Could Allow Remote Code Execution (2850869)

Published: August 13, 2013

Microsoft Security Bulletin MS13-081 - Critical

0 out of 1 rated this helpful - Rate this topic

Vulnerabilities in Windows Kernel-Mode Drivers Could Allow Remote Code Execution (2870008)

Published: October 08, 2013 | Updated: January 14, 2014

No dobra, coś tam jeszcze było, ale teraz już jest na 100% bepiecznie.

Microsoft Security Bulletin MS14-045 - Important

This topic has not yet been rated - Rate this topic

Vulnerabilities in Kernel-Mode Drivers Could Allow Elevation of Privilege (2984615)

Published: August 12, 2014 | Updated: August 27, 2014

Microsoft Security Bulletin MS14-058 - Critical

This topic has not yet been rated - Rate this topic

Vulnerabilities in Kernel-Mode Driver Could Allow Remote Code Execution (3000061)

Published: October 14, 2014

Microsoft Security Bulletin MS15-010 - Critical

15 out of 31 rated this helpful - Rate this topic

Vulnerabilities in Windows Kernel-Mode Driver Could Allow Remote Code Execution (3036220)

Published: February 10, 2015 | Updated: February 18, 2015

yhm...

MS15-078	OpenType Font Driver Vulnerability	CVE-2015-2426	Mateusz Jurczyk of Google Project Zero
MS15-080	OpenType Font Parsing Vulnerability	CVE-2015-2432	Mateusz Jurczyk of Google Project Zero
MS15-080	TrueType Font Parsing Vulnerability	CVE-2015-2455	Mateusz Jurczyk of Google Project Zero
MS15-080	TrueType Font Parsing Vulnerability	CVE-2015-2456	Mateusz Jurczyk of Google Project Zero
MS15-080	OpenType Font Parsing Vulnerability	CVE-2015-2458	Mateusz Jurczyk of Google Project Zero
MS15-080	OpenType Font Parsing Vulnerability	CVE-2015-2459	Mateusz Jurczyk of Google Project Zero
MS15-080	OpenType Font Parsing Vulnerability	CVE-2015-2460	Mateusz Jurczyk of Google Project Zero
MS15-080	OpenType Font Parsing Vulnerability	CVE-2015-2461	Mateusz J
MS15-080	OpenType Font Parsing Vulnerability	CVE-2015-2462	Mateusz J
MS15-080	TrueType Font Parsing Vulnerability	CVE-2015-2463	Mateusz J
MS15-080	TrueType Font Parsing Vulnerability	CVE-2015-2464	Mateusz J
MS15-021	Adobe Font Driver Denial of Service Vulnerability	CVE-2015-0074	Mateusz Jurczyk of Google Project Zero
MS15-021	Adobe Font Driver Information Disclosure Vulnerability	CVE-2015-0087	Mateusz Jurczyk of Google Project Zero
MS15-021	Adobe Font Driver Remote Code Execution Vulnerability	CVE-2015-0088	Mateusz Jurczyk of Google Project Zero
MS15-021	Adobe Font Driver Information Disclosure Vulnerability	CVE-2015-0089	Mateusz Jurczyk of Google Project Zero
MS15-021	Adobe Font Driver Remote Code Execution Vulnerability	CVE-2015-0090	Mateusz Jurczyk of Google Project Zero
MS15-021	Adobe Font Driver Remote Code Execution Vulnerability	CVE-2015-0091	Mateusz Jurczyk of Google Project Zero
MS15-021	Adobe Font Driver Remote Code Execution Vulnerability	CVE-2015-0092	Mateusz Jurczyk of Google Project Zero
MS15-021	Adobe Font Driver Remote Code Execution Vulnerability	CVE-2015-0093	Mateusz Jurczyk of Google Project Zero
MS15-044	OpenType Font Parsing Vulnerability	CVE-2015-1670	Mateusz Jurczyk of Google Project Zero

Jądro Windows – 21 podatności

- **8 błędów** znalezionych podczas ręcznego audytu.
- **13 błędów** znalezionych przy pomocy fuzzingu.

- **15 błędów** w obsłudze czcionek Type 1 / OpenType (.OTF).
- **6 błędów** w obsłudze czcionek TrueType (.TTF).

Jądro Windows – 21 podatności

- **1 kolizja** z błędem znalezionym w wycieku Hacking Team (OTF).
- **1 kolizja** z błędem wykorzystanym przez Keen Team podczas pwn2own 2015 (TTF).

- **2 nominacje** do Pwnie Awards 2015 (*Best Client-Side Bug*, *Best Privilege Escalation Bug*).
- **1 wygrana** Pwnie Awards 2015 (*Best Client-Side Bug*).

- **2 błędy** poprawione w jutrzejszym Patch Tuesday. ☺

Nurtujące pytanie:

“kiedy to już naprawdę koniec?”

Przykład: IDA Pro

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

Przykład: IDA Pro

- 3 lata trwania bounty.
- Intuicyjnie “łatwy” target.
- Wysokie nagrody.
- Znane nazwiska w Hall of Fame.

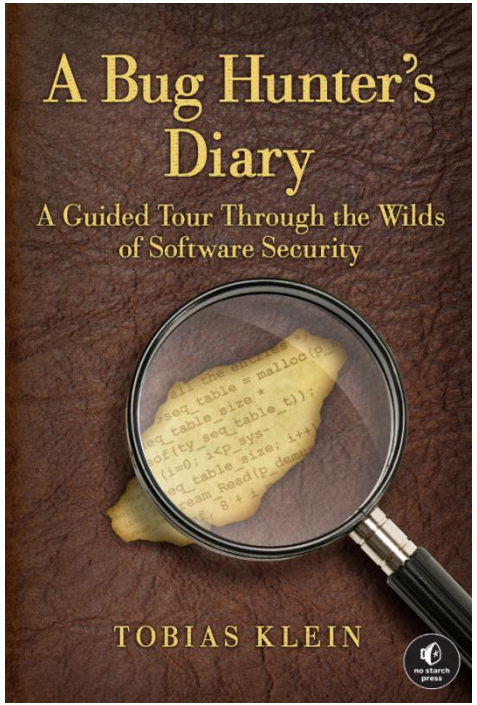
hmm...

Przykład: IDA Pro

2014-09-06 12:54	Mateusz Jurczyk	IDA versions up to 6.6	Multiple vulnerabilities
2014-11-19 23:34	Robert Święcki	IDA versions up to 6.6	Multiple vulnerabilities
2014-11-26 12:07	Mateusz Jurczyk	IDA versions up to 6.6	Multiple vulnerabilities
2014-12-03 01:59	Robert Święcki	IDA versions up to 6.6	Vulnerability in PE loader
2014-12-19 20:15	George Nosenko	IDA versions up to 6.6	Vulnerability in GDB debugger module
2015-01-08 20:48	Mateusz Jurczyk	IDA versions up to 6.7	Multiple vulnerabilities
2015-01-14 12:08	Mateusz Jurczyk	IDA versions up to 6.7	Multiple vulnerabilities
2015-01-27 21:08	Gynvael Coldwind and Mateusz Jurczyk	IDA versions up to 6.7	Multiple vulnerabilities

Przykład: FFmpeg

- *A Bug Hunter's Diary*, Tobias Klein
- Jeden z rozdziałów: "*Chapter 4: NULL Pointer FTW*", opis błędu znalezionego przez autora w projekcie FFmpeg.
- Szybki rzut okiem:
  - Ponad 0.5 mln linii słabej jakości kodu C.
  - Implementacja dziesiątek formatów multimediów.
  - Implemencja dziesiątek kodeków audio/video.
  - Raj dla bughuntera.



… Fast-forward do 2015 …

```
$ git log | grep j00ru | wc -l
1402
$
```

Nie tylko fuzzing

- Pytanie “*czy mogę zrobić to lepiej?*” można zadawać również w kontekście audytowania kodu.
- Jest trudniej:
  - Mniej trywialnych sposobów na ulepszenie techniki innych.
  - Jeśli ktoś już audytuje kod, to *prawdopodobnie* się na tym zna.
  - W ogólności audytowanie kodu jest nietrywialne: wytrwałość, spostrzegawczość, doświadczenie, dogłębna znajomość języka, …
  - Ciężej się wybić na tle innych.

Wciąż się da.

Przykład: Windows kernel trap handlers

Na początku 2010, Tavis Ormandy audytował implementację *trap handlerów* w jądrze Windows.

Rezultat?

Microsoft Windows NT #GP Trap Handler Allows Users to Switch Kernel Stack

From: Tavis Ormandy <taviso () sdf lonestar org>
Date: Tue, 19 Jan 2010 20:11:17 +0100

```
Microsoft Windows NT #GP Trap Handler Allows Users to Switch Kernel Stack
------------------------------------------------------------------------

CVE-2010-0232

In order to support BIOS service routines in legacy 16bit applications, the
Windows NT Kernel supports the concept of BIOS calls in the Virtual-8086 mode
monitor code. These are implemented in two stages, the kernel transitions to
the second stage when the #GP trap handler (nt!KiTrap0D) detects that the
faulting cs:eip matches specific magic values.

Transitioning to the second stage involves restoring execution context and
call stack (which had been previously saved) from the faulting trap frame once
authenticity has been verified.

This verification relies on the following incorrect assumptions:

  - Setting up a VDM context requires SeTcbPrivilege.
  - ring3 code cannot install arbitrary code segment selectors.
  - ring3 code cannot forge a trap frame.

This is believed to affect every release of the Windows NT kernel, from
Windows NT 3.1 (1993) up to and including Windows 7 (2009).

Working out the details of the attack is left as an exercise for the reader.

Just kidding, that was an homage to Derek Soeder :-)
```

Rezultat?

Microsoft to patch 17-year-old computer bug

A 17-year-old bug in Windows will be patched by Microsoft in its latest security update.

The February update for Windows will close the loophole that dates from the time of the DOS operating system.

First appearing in Windows NT 3.1, the vulnerability has been carried over into almost every version of Windows that has appeared since.

The bug dates from the days of Windows NT 3.1

The monthly security update will also tackle a further 25 holes in Windows, five of which are rated as "critical".

Rezultat?

Pwnie for Best Privilege Escalation Bug

Award to the person who discovered and/or exploited the most technically sophisticated and interesting privilege escalation vulnerability. As more defense-in-depth systems like Mandatory Access Control and Virtualization are deployed, privilege escalation vulnerabilities are becoming more important. These vulnerabilities can include local operating system privilege escalations, operating system sandbox escapes, and virtual machine guest breakout vulnerabilities.

- Windows NT #GP Trap Handler (CVE-2010-0232)
  Credit: Tavis Ormandy

  One of the most complicated vulnerabilities of 2010, this privilege escalation bug required more than a few tricks to exploit. Its discovery shows a rare understanding of some of the more obscure aspects of the Intel architecture. The bug was present in all versions of Windows from NT 3.1 all the way up to Windows 7.

Hmm, stary, niskopoziomowy kod?

Brzmi jak dobry target. ☺

Własne rezultaty

Microsoft Security Bulletin MS11-098 - Important

This topic has not yet been rated - Rate this topic

Vulnerability in Windows Kernel Could Allow Elevation of Privilege (2633171)

Published: December 13, 2011 | Updated: February 01, 2012

Version: 1.1

Acknowledgments

Microsoft thanks the following for working with us to help protect customers:

- Mateusz "j00ru" Jurczyk, working with VeriSign iDefense Labs, for reporting the Windows Kernel Exception Handler Vulnerability (CVE-2011-2018)

Własne rezultaty

The story of CVE-2011-2018 exploitation

Mateusz “j00ru” Jurczyk

February - April 2012

Abstract

Exploitation of Windows kernel vulnerabilities is recently drawing more and more attention, as observed in both monthly Microsoft advisories and technical talks presented on public security events. One of the most recent security flaws fixed in the Windows kernel was CVE-2011-2018[1], a vulnerability which could potentially allow a local attacker to execute arbitrary code with system privileges. The problem affected all - and only - 32-bit editions of the Windows NT-family line, up to Windows 8 Developer Preview[2]. In this article, I present how certain novel exploitation techniques can be used on different Windows platforms to reach an elevation of privileges through this specific kernel vulnerability.

Własne rezultaty

Pwnie for Best Privilege Escalation Bug

Awarded to the person who discovered or exploited the most technically sophisticated and interesting privilege escalation vulnerability. As more defense-in-depth systems like Mandatory Access Control and Virtualization are deployed, privilege escalation vulnerabilities are becoming more important. These vulnerabilities can include local operating system privilege escalations, operating system sandbox escapes, and virtual machine guest breakout vulnerabilities.

- MS11-098: Windows Kernel Exception Handler Vulnerability (CVE-2011-2018)
  Credit: Mateusz "j00ru" Jurczyk

  j00ru owned Windows. All of them. Ok, well just all of the 32-bit versions of Windows from NT through the Windows 8 Developer Preview. What have you done lately? And to top it off, he wrote a clear paper on it with some of the nicest boxy diagrams we have ever seen in a LaTeX paper.

No dobra...

- Błąd znaleziony nie podczas manualnego audytu, a pisania CrackMe na konkurs *secnews.pl*.
- Niestandardowe użycie 32-bitowego mechanizmu LDT.
- Losowe BSoD podczas testowego uruchamiania aplikacji.
- Okazało się, że to nietrywialny w exploitacji Privilege Escalation.

Tak naprawdę…

a. **CVE-2013-3196** (`nt!PushInt` *write-what-where condition*)

b. **CVE-2013-3197** (`nt!PushException` *write-what-where condition*)

c. **CVE-2013-3198** (`nt!VdmCallStringIoHandler` *write-where condition*)

d. **CVE-2013-3136** (`nt!KiTrap0E` *memory disclosure vulnerability*)

Cała gama błędów

- Wszystkie 32-bitowe wersje Windows:
  - 3 lokalne luki podniesienia uprawnień.
  - 1 kernel memory disclosure.
  - kilka lokalnych błędów DoS.
  - kilka lokalnych błędów ujawnienia adresów przestrzeni jądra (ASLR bypass).

Momenty WTF

- Czy da się wywalić cały system Windows dwiema instrukcjami?
  - teraz już nie, ale można było.

```
xor ebp, ebp
jmp 0x8327d1b7
```

```
nt!KiSystemServiceAccessTeb
```

Odnośniki

- NoSuchCon 2013, *Abusing the Windows Kernel: How to Crash an Operating System With Two Instructions*
- SEConference 2013, *Bezpieczeństwo jądra Windows, lub jak zabić system dwiema instrukcjami*
- ZeroNights 2013, *Windows Kernel Trap Handler and NTVDM Vulnerabilities – Case Study*

A co, jeśli...

- Nie zawsze dokładnie wiadomo, czy (i przez kogo) dany target był audytowany.
- Logiczne założenie: jeśli ma wielu użytkowników / developerów, to był wielokrotnie i sromotnie audytowany.
- W szczególności: otwarte oprogramowanie.

NOPE

Słowo klucz: rozproszenie odpowiedzialności

Przykładów nie trzeba daleko szukać

Myślę, że każdy na sali mógłby wymienić kilka krytycznych podatności w oprogramowaniu o otwartym kodzie, z ostatnich lat.

Brak ocalałych

- Weźmy za przykład popularne biblioteki obsługi obrazów.
- Ostatnia poważna podatność:
  - libjpeg6b: listopad 2013, *memory disclosure* (CVE-2013-6629)
  - libjpeg-turbo: listopad 2014, *buffer overflow* (CVE-2014-9092)
  - libpng: grudzień 2014, *buffer overflow* (CVE-2014-9495)
  - libtiff: bez przerwy są naprawiane kolejne błędy.

Przemyślenie

- Jeśli zastanawiasz się nad analizą konkretnego oprogramowania, weź pod uwagę, że najprawdopodobniej było audytowane przez mniej osób, niż Ci się wydaje.
  - nawet (a może zwłaszcza?), jeśli chodzi o *open-source*.
- Co oczywiście nie znaczy, że proste jest znalezienie podatności w *zlib* czy *libjpeg*.
  - z biegiem czasu coraz trudniejsze.

Wracając do błędu Tavisa...

Wiek kodu a podatności

- W latach ‘90 mało kto przejmował się pisaniem bezpiecznego kodu.
  - brak powszechnej wiedzy na ten temat,
  - brak literatury,
  - brak odpowiednich polityk u producentów oprogramowania.
- Jeśli już, liczyły się podatności w zdalnych usługach (*sendmail* etc.)

Wiek kodu a podatności

- Przez kolejne 2 epoki sytuacja drastycznie się zmieniła:
  - wiele prawdziwie zdalnych błędów zostało naprawionych.
  - *software security* zyskało na popularności.
  - producenci zaczęli kłaść nacisk na tworzenie bezpiecznego kodu.
  - wdrożone zostały uniwersalne zabezpieczenia (ASLR, NX, SSP, …)
  - błędy typu “remote remote” prawie wyginęły.

Era podatności local-remote i local

- *Remote code execution* poprzez skłonienie ofiary do (teoretycznie bezpiecznej) interakcji poprzez otwarcie strony internetowej, otwarcie dokumentu itp.
- *Local privilege escalation* w celu ucieczki z sandboxa i przejęcia całkowitej kontroli nad systemem.

Nagle okazuje się, że celem ataku w takim scenariuszu może być kod napisany 20-30 lat temu!

Czy jest aż tak źle?

- Natywny kod sprzed dekad prawie na pewno nie bierze pod uwagę *security*.
  - zwłaszcza, jeśli w międzyczasie nikt na niego nie patrzył.
- Nawet jeśli producent zmienił nawyki, prawie na pewno nie wykonał *refactoringu* starych komponentów.
- Idealny cel ataku!
  - Czasem nawet sam producent nie jest świadomy, że tak stare moduły wciąż znajdują się w aplikacji.

Przykłady

- Wspomniane wcześniej błędy w rdzennych częściach jądra Windows (*trap handlers*).
- Wspomniane wcześniej błędy w obsłudze czcionek.
- Niezliczone ilości błędów w *win32k.sys*, głównym module graficznym jądra Windows.

Przykłady

Rok: 2011

Cel: Apple QuickTime


QuickTime
00:32:40

Graphics and Still-Image Formats

Here is a list of common graphics and still-image formats:

- *BMP:* Standard bit-mapped graphics format used on Windows computers.
- *FlashPix:* A format for storing digital images, especially digital photographs, developed by Eastman Kodak Company.
- *GIF:* Graphic Interchange Format. A common bit-mapped graphics file format used on the web.
- *JPEG/JFIF:* Joint Photographics Experts Group. A "lossy" compression file format used for images. JFIF is JPEG File Interchange Format.
- *MacPaint (PNTG):* A monochrome file format used on early versions of the Mac operating system.
- *Photo JPEG:* An extremely popular file format because it can create highly compressed yet good-looking graphics files. You can choose grayscale or color as well as the amount of compression.
- *Photoshop (PSD):* You can import files created or saved in the Photoshop format, along with multilayered Photoshop files. (For more information, see Compositing and Layering.)
- *PICS:* A file format used on Mac OS computers for animation sequences. The format is no longer used, in favor of QuickTime.
- *PICT:* A common image format used on Mac OS computers. PICT files can use any of the standard QuickTime codecs for compression in color or grayscale.
- *PNG:* Portable Network Graphics. A file format for bitmapped graphic images designed as the successor to GIF.
- *QuickTime Image File (QTIF):* A QuickTime container file that contains an image using a supported QuickTime codec.
- *SGI:* Silicon Graphics Image file format.
- *TARGA (TGA):* The Targa file format. An uncompressed file format that stores images with millions of colors. Targa files are supported by nearly every platform and media application.
- *TIFF:* Common on Mac OS and Windows computers. TIFF files allow color depths from dithered black and white to millions of colors and one form of compression.

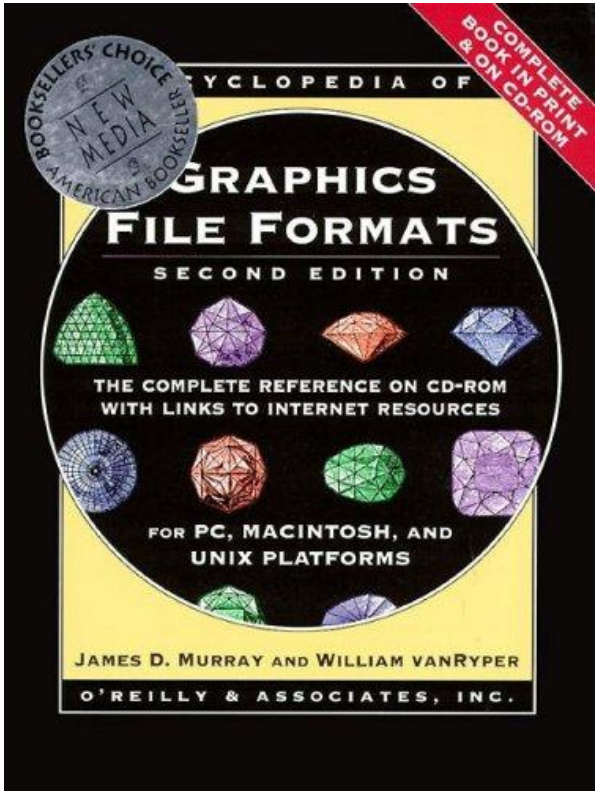
Note: Almost all of these file formats can contain an alpha channel.

- *PICT:* A common image format used on Mac OS computers.

PICT?

Biblioteczka *bughuntera*

- *Encyclopedia of Graphics File Formats.*
- Książka z 1996 roku.
- Znaleziona w antykwariacie w *Mountain View*.
- Opisuje wiele archaicznych formatów, m.in. PICT.
  - wiele z nich w dalszym ciągu jest obsługiwanych przez dzisiejsze programy.



W rezultacie…

- **QuickTime**

  Available for: Mac OS X v10.5.8, Mac OS X Server v10.5.8, Windows 7, Vista, XP SP2 or later

  Impact: Viewing a maliciously crafted movie file may lead to an unexpected application termination or arbitrary code execution

  Description: A heap buffer overflow existed in the handling of STSC atoms in QuickTime movie files. Viewing a maliciously crafted movie file may lead to an unexpected application termination or arbitrary code execution. This issue does not affect OS X Lion systems.

  CVE-ID

  CVE-2011-0249 : Matt 'j00ru' Jurczyk working with TippingPoint's Zero Day Initiative

Kod jest jak wino.

(dla bughuntera)
(szczególnie, jeśli dojrzewało w samotności)

Wychodząc w przyszłość

- Znajdujemy błąd.
- Zgłaszamy do producenta.
- Producent wypuszcza poprawkę.

Czy to już koniec?

Warto sprawdzić

- Jeśli błąd jest istotnie krytyczny, warto własnoręcznie potwierdzić.
  - nie tylko w przypadku własnych znalezisk.
- Nawet w przypadku *vendora* zamkniętego oprogramowania (który ma znacznie większą wiedzę od nas), zdarzają się błędy.

Przykład

- *Stuxnet LNK vulnerability* (CVE-2010-2568)
- Umożliwia zdalne wykonanie kodu ze 100% skutecznością.
- Naprawiony przez Microsoft w biuletynie MS10-046.

Microsoft Security Bulletin MS10-046 - Critical

Vulnerability in Windows Shell Could Allow Remote Code Execution (2286198)

Published: August 02, 2010 | Updated: August 24, 2010

Version: 1.2

Niepoprawnie naprawiona, możliwa do wykorzystania aż do Marca 2015.

Microsoft Security Bulletin MS15-020 - Critical

4 out of 4 rated this helpful - Rate this topic

Vulnerabilities in Microsoft Windows Could Allow Remote Code Execution (3041836)

Published: March 10, 2015 | Updated: March 10, 2015

Version: 1.1

Inny przykład

Maj 2013

Czerwiec 2013

Nowy, skomplikowany kod

- I to od Adobe! ☺
- Brzmi jak dobry cel fuzzingu.
  - przez pierwsze kilka dni nic.
  - po czym…

Jest crash!

```
==31664==ERROR: AddressSanitizer: stack-buffer-overflow on address
0x7fff76853d70 at pc 0x6d2d74 bp 0x7fff768537f0 sp 0x7fff768537e8
READ of size 1 at 0x7fff76853d70 thread T0
#0 0x6d2d73 in cf2_hintmap_build freetype2/src/cff/cf2hints.c:820
#1 0x6b85ab in cf2_interpT2CharString freetype2/src/cff/cf2intrp.c:1201
#2 0x6ae390 in cf2_getGlyphOutline freetype2/src/cff/cf2font.c:461
#3 0x6aafa6 in cf2_decoder_parse_charstrings freetype2/src/cff/cf2ft.c:369
#4 0x6a134c in cff_slot_load freetype2/src/cff/cffgload.c:2840
#5 0x66b8b0 in cff_glyph_load freetype2/src/cff/cffdrivr.c:185
#6 0x4a28e2 in FT_Load_Glyph freetype2/src/base/ftobjs.c:726
#7 0x492475 in test_load freetype2-demos/src/ftbench.c:249
#8 0x4930e7 in benchmark freetype2-demos/src/ftbench.c:216
#9 0x48f692 in main freetype2-demos/src/ftbench.c:102
```

Krótka analiza

- Stosowy *buffer overflow*.
- Błąd w implementacji *CharStringów* – programu wirtualnej maszyny CFF.
- Pozwala na wyczyszczenie dowolnych bitów na stosie.
- Pozwala ominąć *stack cookie*, łatwy i stabilny w wykorzystaniu.

```
$ ./checksec.sh --file freetype2-demos/bin/ftbench
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
Partial RELRO   Canary found      NX disabled   PIE enabled     No RPATH   No RUNPATH   freetype2-demos/bin/ftbench

$ gdb --args freetype2-demos/bin/ftbench exploit.otf
GNU gdb (GDB) 7.7-gg5
[…]
(gdb) set disable-randomization off
(gdb) r
Starting program: freetype2-demos/bin/ftbench exploit.otf

ftbench results for font `exploit.otf'
--------------------------------------------------------

[…]

executing tests:
  Load                      1205.440 us/op
  Load_Advances (Normal)    1.731 us/op
  Load_Advances (Fast)      1.601 us/op
  Render                    3.437 us/op
  Get_Glyph                 0.497 us/op
  Get_CBox                  0.318 us/op
  Get_Char_Index            0.145 us/op
  Iterate CMap              552.902 us/op
  New_Face                  642.765 us/op
  Embolden                  13.648 us/op
  Get_BBox                  0.665 us/op
process 13971 is executing new program: /bin/sh
sh-4.2$
```

Typowa procedura

- Błąd zgłoszony do *FreeType* 25 lutego 2014.
- Poprawka wysłana 28 lutego.
- Nowa wersja biblioteki wydana 8 marca.

Dobra robota, świat uratowany!

Czyżby?

Ciąg dalszy historii

- 21 listopada po raz kolejny fuzzuję *FreeType.*
  - nowy zestaw mutacji.
  - nowy zestaw plików testowych.
- Mija minuta, i…

Hmm, gdzieś to już widziałem…

```
=================================================================
==15055==ERROR: AddressSanitizer: stack-buffer-overflow on address
0x7fff2dc05b30 at pc 0x71134e bp 0x7fff2dc055b0 sp 0x7fff2dc055a8
READ of size 1 at 0x7fff2dc05b30 thread T0
#0 0x71134d in cf2_hintmap_build freetype2/src/cff/cf2hints.c:822
#1 0x7048e1 in cf2_glyphpath_moveTo freetype2/src/cff/cf2hints.c:1606
#2 0x6f5259 in cf2_interpT2CharString freetype2/src/cff/cf2intrp.c:1243
#3 0x6e8570 in cf2_getGlyphOutline freetype2/src/cff/cf2font.c:469
#4 0x6e4c5e in cf2_decoder_parse_charstrings freetype2/src/cff/cf2ft.c:367
#5 0x6da446 in cff_slot_load freetype2/src/cff/cffgload.c:2840
#6 0x69dbcc in cff_glyph_load freetype2/src/cff/cffdrivr.c:185
#7 0x4a427e in FT_Load_Glyph freetype2/src/base/ftobjs.c:726
#8 0x491d69 in test_load ft2demos-2.5.3/src/ftbench.c:249
#9 0x492b51 in benchmark ft2demos-2.5.3/src/ftbench.c:216
#10 0x48e962 in main ft2demos-2.5.3/src/ftbench.c:1020
```

Poprawny fix za drugim razem

- *Crash* zgłoszony 21 listopada 2014.
- Tym razem poprawna łatka wysłana na początku grudnia.
- Wersja 2.5.4 wydana 6 grudnia.

Wniosek: nie ufać przesadnie developerom. ☺

OTHER THOUGHTS (MAMY CZAS?)

Przemyślenia

- Błędne przeświadczenie, że jeśli nie istnieją publicznie znane błędy w oprogramowaniu, to jest ono bezpieczne.
- “Nie używam programu X aż do załatania problemów o których głośno mówi firma/researcher Y”.
- Błędy w aplikacjach istnieją cały czas (idą w setki), i na tym założeniu warto opierać swoją politykę bezpieczeństwa.

Przemyślenia

- Zamiast pojedynczych błędów, można skupiać się i znajdować całe (być może zupełnie nowe) ich klasy.
  - `LoadLibrary()` i DLL hijacking.
  - Jądra, sterowniki, i interakcja z pamięcią trybu użytkownika (*Bochspwn*).
  - …

Przemyślenia

- Inspiracja jest istotna.
  - konferencje,
  - artykuły naukowe,
  - posty na blogach,
  - *bindiffing*.

Przemyślenia

- Wybór targetu, w którym szukamy błędów, jest co najmniej tak samo ważny, jak sam proces *bughuntingu*.

Przemyślenia

- Bindiffing jako źródło 0-dayów.

Dziękuję za uwagę!

@j00ru

http://j00ru.vexillium.org/

j00ru.vx@gmail.com