Corrupted Memories of Memory Corruption

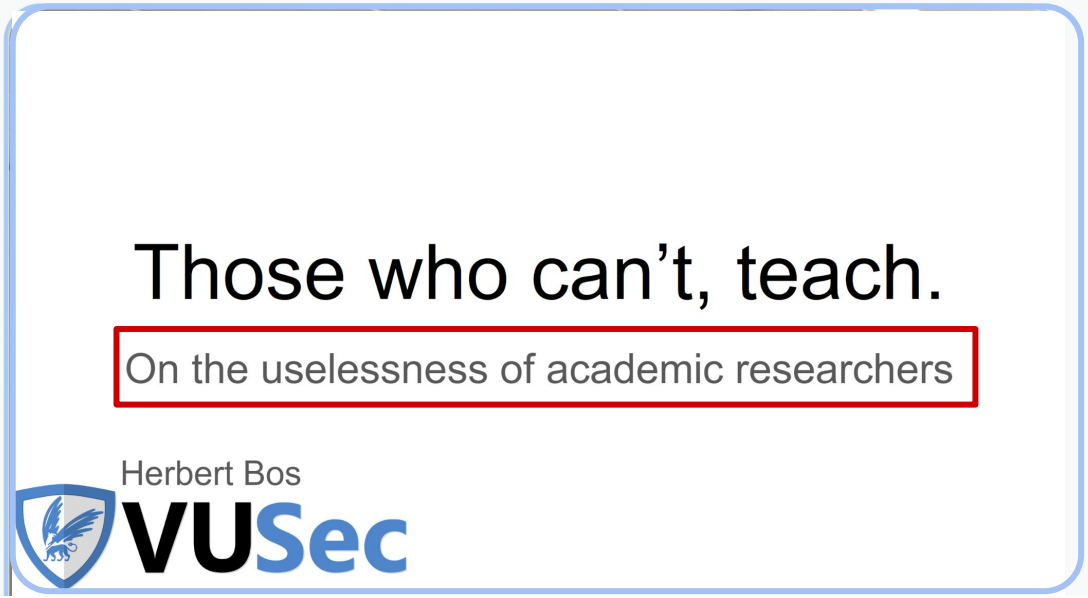
Offensive Security, Academia, and the Rest of the World

Herbert Bos

VUSec

ISC Conference 2023 @isc_conf · 16 nov. 2023

Day 2 of #ISC23 kicks off with an intriguing keynote by Herbert Bos @herbertbos @vu5ec "The Uselessness of Academic Researchers," sparking thought-provoking insights and discussions.

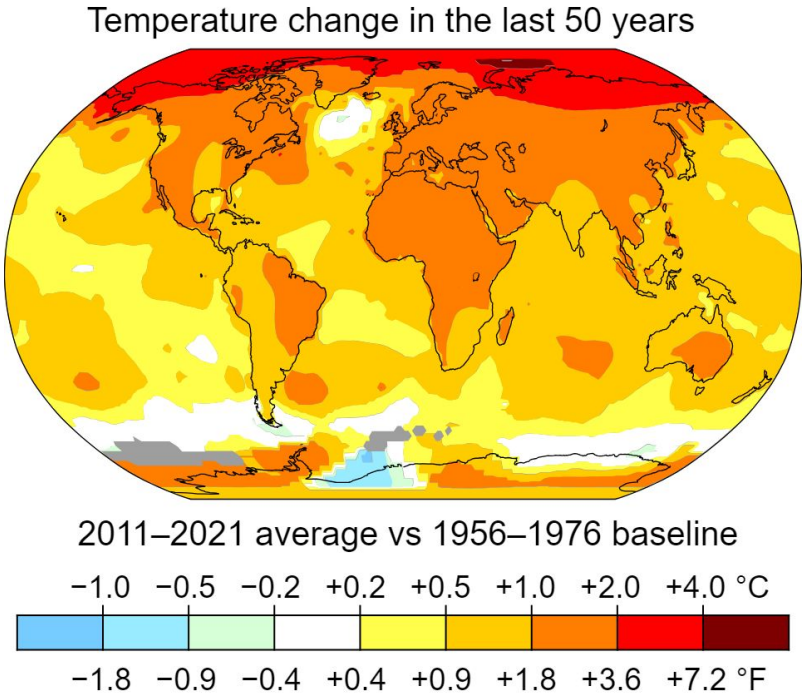


Hajo Reijers @profBPM · 16 nov. 2023

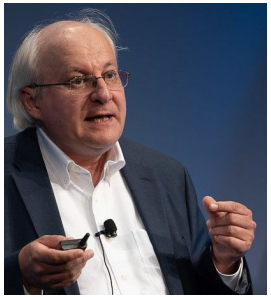
He is an expert on this! Listen carefully

In the grander scheme of things

This talk is relatively unimportant

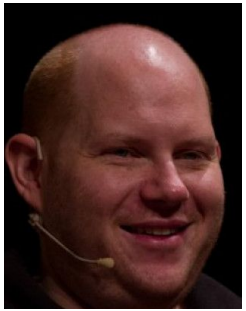


Thanks



Sergey Bratus

Thomas Dullien
(Halvar Flake)



Taddeus Grugq
(The Grugq)

Memory corruption and me and you

A minor philosophical point: I'd suggest shifting the focus slightly from memory corruption as such to

harnessing emergent properties of memory abstractions

What I hope to achieve

Honor those who got us here

Assess where we stand

Look forward

Memory Safety is So Simple

Stay inside your box

Memory **Corruption** is also Simple

Memory **Corruption** is also Simple

Do Not

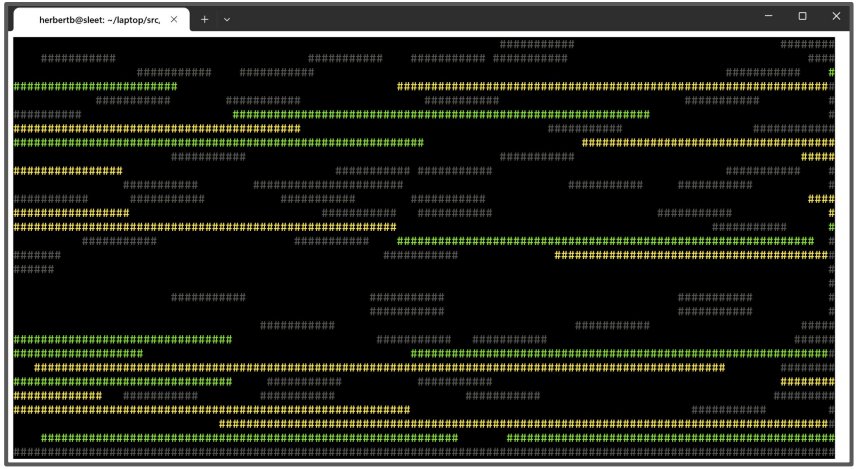
Stay inside your box

0

Nothing new

```
prot:
        .int 0
code:
        .quad 0
origin:
        add %rsi, %rdx
l1:
        mov code(%rip), %rdi
        inc %rdi
        cmp %rdi, %rsi
        cmova %rsi, %rdi
        cmp %rdi, %rdx
        cmovbe %rsi, %rdi
        mov %rdi, code(%rip)
        call *probe(%rsi)
        cmp $my_id, %rax
        je l1
        test %rax, %rax
        je l2
        call *kill(%rsi)
l2:
        sub %rbx, %rcx
        cmp $end - code, %rcx
        jl l1
        mov %rbx, %rdi
        call *claim(%rsi)
        mov $end - code, %rcx
l3:
        lea code(%rip), %rax
        mov (%rax, %rcx), %al
        mov %al, (%rbx, %rcx)
        loop l3
        jmp l1
end:
```

```
        # protected bytes
prot:
        .int 0, 1, 2, 3 # ...

code:

origin:
        jmp origin

        # PROBE example
        call *probe(%rsi)

        # KILL example
        call *kill(%rsi)

        # CLAIM example
        call *claim(%rsi)

end:
```



Darwin (1961)

Victor Vyssotsky

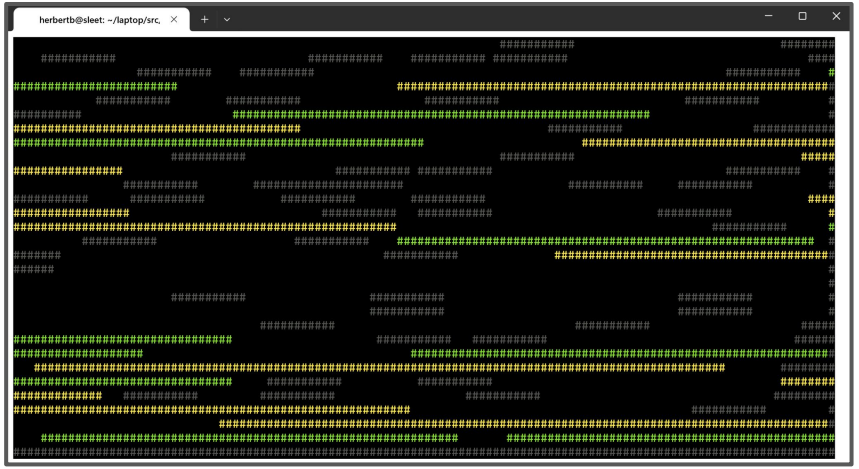
Robert Morris Sr.

Douglas McIlroy

Nothing new

```
prot:
        .int 0
code:
        .quad 0
origin:
        add %rsi, %rdx
l1:
        mov code(%rip), %rdi
        inc %rdi
        cmp %rdi, %rsi
        cmova %rsi, %rdi
        cmp %rdi, %rdx
        cmovbe %rsi, %rdi
        mov %rdi, code(%rip)
        call *probe(%rsi)
        cmp $my_id, %rax
        je l1
        test %rax, %rax
        je l2
        call *kill(%rsi)
l2:
        sub %rbx, %rcx
        cmp $end - code, %rcx
        jl l1
        mov %rbx, %rdi
        call *claim(%rsi)
        mov $end - code, %rcx
l3:
        lea code(%rip), %rax
        mov (%rax, %rcx), %al
        mov %al, (%rbx, %rcx)
        loop l3
        jmp l1
end:
```

```
        # protected bytes
prot:
        .int 0, 1, 2, 3 # ...

code:

origin:
        jmp origin

        # PROBE example
        call *probe(%rsi)

        # KILL example
        call *kill(%rsi)

        # CLAIM example
        call *claim(%rsi)

end:
```



Darwin (1961)

(Won definitely by Robert Morris)

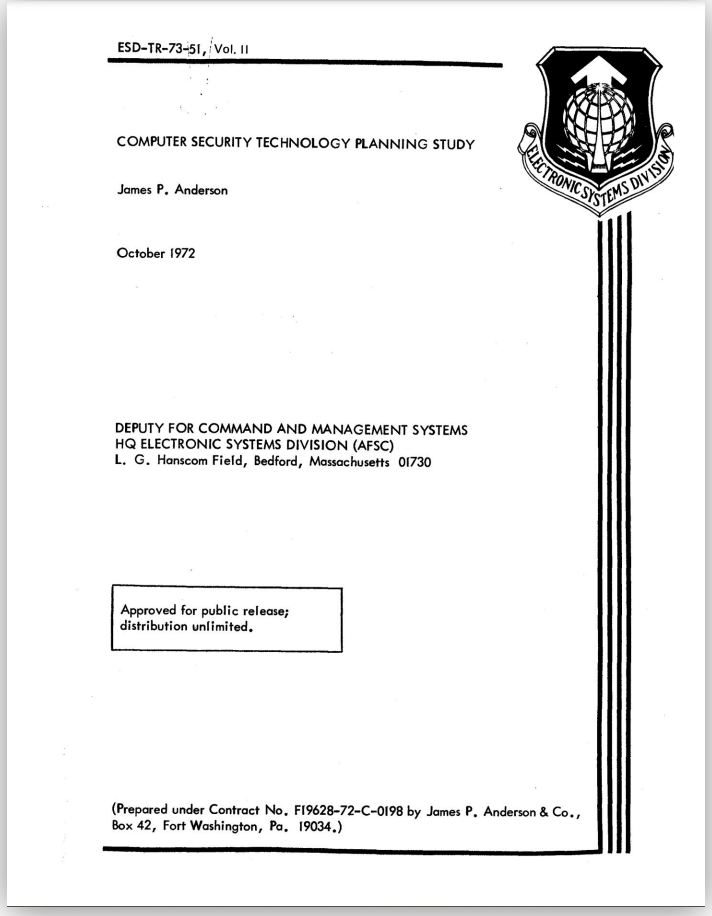
Victor Vyssotsky

Robert Morris Sr.

Douglas McIlroy

USAF Computer Security Technology Planning Study (1972)

Warning



ESD-TR-73-51, Vol. II

COMPUTER SECURITY TECHNOLOGY PLANNING STUDY

James P. Anderson

October 1972

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

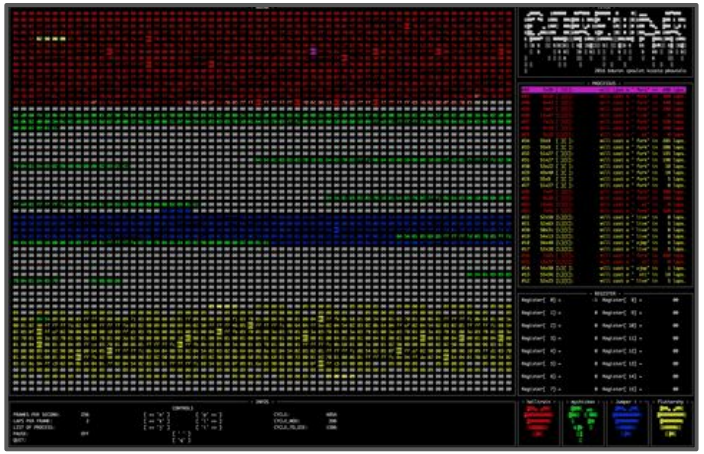
Approved for public release;
distribution unlimited.

(Prepared under Contract No. F19628-72-C-0198 by James P. Anderson & Co., Box 42, Fort Washington, Pa. 19034.)

Core War

Abstract assembly language (“Redcode”)

Still actively played



```
ADD #4, 3
MOV 2, @2
JMP -2
DAT #0, #0
```

Alexander Dewdney

GHOSTBUSTERS!

GHOSTBUSTERS

WHO YOU GONNA CALL?

GHOSTBUSTERS!

CRACKED BY BABYSOFT

GHOSTBUSTERS

WHO YOU GONNA CALL?

GHOSTBUSTERS!

CRACKED BY BABYSOFT KISSES TO HEI

1985 Phrack 01

How to: Acetylene balloon bomb

.:: Introduction... ::.

Issues: [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70]

Current issue : #1 | Release date : 1985-11-17 | Editor : Taran King — Get tar.gz

Introduction...	Taran King
Hacking SAM - A Description Of The Dial-Up Security System	Spitfire Hacker
Boot Tracing Made Easy	Cheap Shades
THE PHONE PHREAK'S FRY-UM GUIDE	Iron Soldier
Using MCI Calling Cards	Knight Lightning
How to Pick Master Locks	Ninja NYC
Acetylene Balloon Bomb	The Clashmaster & Gin Fizz
Schools and University Numbers	Phantom Phreaker

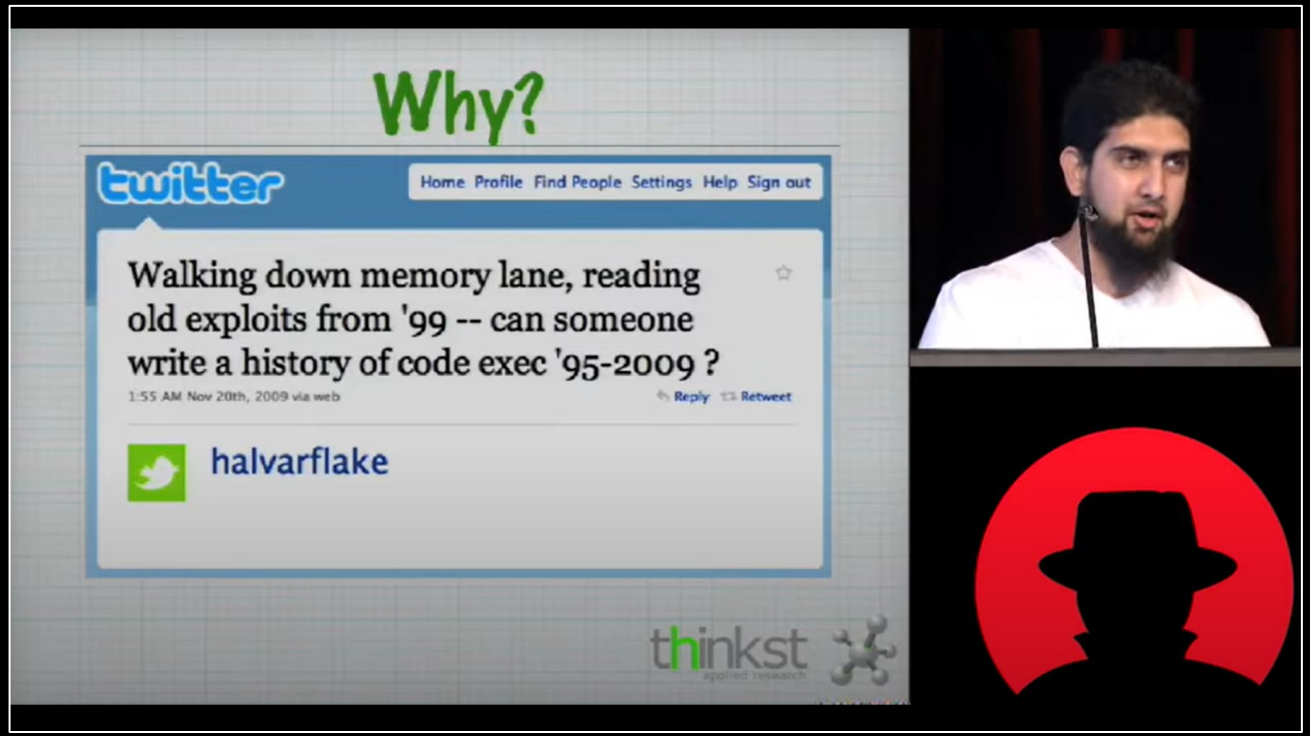
1988 Morris Worm

fingerd

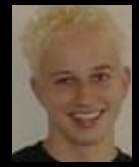
```
char line[512];
...
line[0] = '\0';
...
gets(line);
```

1

Go see Haroon's BlackHat Talk!



Heap Overflows
--- Matt Conover
& w00w00



CERT

Morris

Zardoz

Core Sec

BugTraq

phrack

“How to Write Buffer
Overflows” --Mudge

“Smashing the Stack”
-- Aleph One

Heap Overflows
-- Dildog

FP overwrite
--klog

Format string bug
-- Tymm Twyllman

‘85	‘86	‘87	‘88	‘89	‘90	‘91	‘92	‘93	‘94	‘95	‘96	‘97	‘98	‘99

Overflow in NCSA httpd
-- Thomas
Lopatic

“ret-2-libc”
-- Solar Designer

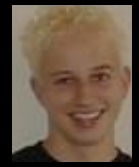
StackGuard
-- Crispin Cowan

Limitations -- Tim Newsham
“Code reuse chaining”

Windows BOs
-- Barnaby Jack

Format string bug in qmail
-- Dan Bernstein

Heap Overflows
--- Matt Conover
& w00w00



CERT

Morris

Zardoz

Core Sec

BugTraq

phrack

“How to Write Buffer
Overflows” --Mudge

“Smashing the Stack”
-- Aleph One

Heap Overflows
-- Dildog

FP overwrite
--klog

Format string bug
-- Tymm Twyllman

‘85	‘86	‘87	‘88	‘89	‘90	‘91	‘92	‘93	‘94	‘95	‘96	‘97	‘98	‘99

Overflow in NCSA httpd
-- Thomas
Lopatic

“ret-2-lib”
-- Solar Designer

StackGuard
-- Crispin Cowan

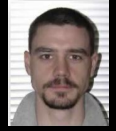
Limitations -- Tim Newsham
“Code reuse chaining”

Windows BOs
-- Barnaby Jack

Format string bug in qmail
-- Dan Bernstein

What about me?

Vudo malloc tricks
-- MaXX



"Non-Stack based exploitation"
-- David Litchfield

"Unsafe unlinking of the lookaside list" -- sh0k, Horovitz

ROP reaches academia!

Bypassing StackGuard + StackShield (GOT overwrite)
--Bulba & Kil3r

Heap Spray
-- eEye

Code Red
SEH handler overwrite

On the effectiveness of ASLR -Shacham et al.

"Geometry of Innocent Flesh" -- H. Shacham et al.

"Format String Attacks"
-- Tim Newsham

Advanced return-2-libc
-- nergal

MetaSploit
-- H.D. Moore

"double free"

"Application-Specific Attacks - Leveraging the ActionScript VM"
-- M. Dowd, A. Sotirov

"Smashing C++ VPTRS"
-- rix

"Unusual Bugs"
-- Ilja van Sprundel

'00 '01 '02 '03 '04 '05 '06 '07 '08 '09 '10

"Heap Feng Shui in JavaScript" -- A. Sotirov

"Exploiting non-terminated adjacent memory spaces"
-- twitch@vicar.org

ASLR

PaX

"Third Generation Exploits"
--Halvar Flake

"Borrowed Code Chunk Exploitation Technique"
--Sebastian Krahmer

"CFI"
-- Abadi, Budiu, Erlingsson, Ligatti

PAGEEXEC
MPROTECT

grsecurity
-- spender

"null ptr deref in kernel"

"Bypass HW DEP"
-- Matt Miller, Ken Johnson

Integer Overflows
-- Mark Dowd,
-- Chris Spencer,
-- Neel Metha,
-- Nishad Herath
-- Halvar Flake

"Bypassing PaX ASLR Protection" -- Tyler Durden

"Advances in Format String Exploitation"
"4 tricks to bypass StackGuard and StackShield
-- Gerardo Richarte (gera)

"Application-Specific Attacks - Leveraging the ActionScript VM"

"zero allocation vulnerabilities"
-- Julien Vanegue

What about me?

1978

I want to be a superhero

I want to be a superhero

2012

My lectures:

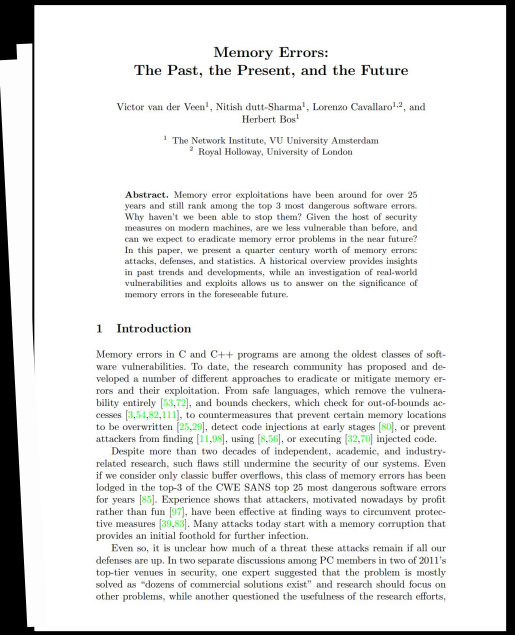
My lectures:

"Unfortunately, I ended up breaking up with my girlfriend towards the end of the period, she said she felt like I did not prioritize her and that I did not have enough time for her anymore… I guess 8 hours of only Computer & Network Security per day has its price. But it is okay, don't sweat it, I would do it all over again. Great course, I learned a lot!"

My lectures:

"Unfortunately, I ended up breaking up with my girlfriend towards the end of the period, she said she felt like I did not prioritize her and that I did not have enough time for her anymore… I guess 8 hours of only Computer & Network Security per day has its price. But it is okay, don't sweat it, I would do it all over again. Great course, I learned a lot!"

Memory Errors



Memory Errors:
The Past, the Present, and the Future

Victor van der Veen[1], Nitish dutt-Sharma[1], Lorenzo Cavallaro[1,2], and Herbert Bos[1]

[1] The Network Institute, VU University Amsterdam
[2] Royal Holloway, University of London

Abstract. Memory error exploitations have been around for over 25 years and still rank among the top 3 most dangerous software errors. Why haven't we been able to stop them? Given the host of security measures on modern machines, are we less vulnerable than before, and can we expect to eradicate memory error problems in the near future? In this paper, we present a quarter century worth of memory errors: attacks, defenses, and statistics. A historical overview provides insights in past trends and developments, while an investigation of real-world vulnerabilities and exploits allows us to answer on the significance of memory errors in the foreseeable future.

1 Introduction

Memory errors in C and C++ programs are among the oldest classes of software vulnerabilities. To date, the research community has proposed and developed a number of different approaches to eradicate or mitigate memory errors and their exploitation. From safe languages, which remove the vulnerability entirely [53,72], and bounds checkers, which check for out-of-bounds accesses [3,54,82,111], to countermeasures that prevent certain memory locations to be overwritten [25,29], detect code injections at early stages [80], or prevent attackers from finding [11,98], using [8,56], or executing [32,70] injected code.

Despite more than two decades of independent, academic, and industry-related research, such flaws still undermine the security of our systems. Even if we consider only classic buffer overflows, this class of memory errors has been lodged in the top-3 of the CWE SANS top 25 most dangerous software errors for years [85]. Experience shows that attackers, motivated nowadays by profit rather than fun [97], have been effective at finding ways to circumvent protective measures [39,83]. Many attacks today start with a memory corruption that provides an initial foothold for further infection.

Even so, it is unclear how much of a threat these attacks remain if all our defenses are up. In two separate discussions among PC members in two of 2011's top-tier venues in security, one expert suggested that the problem is mostly solved as "dozens of commercial solutions exist" and research should focus on other problems, while another questioned the usefulness of the research efforts,

Memory Errors

Memory Errors:
The Past, the Present, and the Future

Victor van der Veen[1], Nitish dutt-Sharma[1], Lorenzo Cavallaro[1,2], and Herbert Bos[1]

[1] The Network Institute, VU University Amsterdam
[2] Royal Holloway, University of London

Abstract. Memory error exploitations have been around for over 25 years and still rank among the top 3 most dangerous software errors. Why haven't we been able to stop them? Given the host of security measures on modern machines, are we less vulnerable than before, and can we expect to eradicate memory error problems in the near future? In this paper, we present a quarter century worth of memory errors: attacks, defenses, and statistics. A historical overview provides insights in past trends and developments, while an investigation of real-world vulnerabilities and exploits allows us to answer on the significance of memory errors in the foreseeable future.

1 Introduction

Memory errors in C and C++ programs are among the oldest classes of software vulnerabilities. To date, the research community has proposed and developed a number of different approaches to eradicate or mitigate memory errors and their exploitation. From safe languages, which remove the vulnerability entirely [53,72], and bounds checkers, which check for out-of-bounds accesses [3,54,82,111], to countermeasures that prevent certain memory locations to be overwritten [25,29], detect code injections at early stages [80], or prevent attackers from finding [11,98], using [8,56], or executing [32,70] injected code.

Despite more than two decades of independent, academic, and industry-related research, such flaws still undermine the security of our systems. Even if we consider only classic buffer overflows, this class of memory errors has been lodged in the top-3 of the CWE SANS top 25 most dangerous software errors for years [85]. Experience shows that attackers, motivated nowadays by profit rather than fun [97], have been effective at finding ways to circumvent protective measures [39,83]. Many attacks today start with a memory corruption that provides an initial foothold for further infection.

Even so, it is unclear how much of a threat these attacks remain if all our defenses are up. In two separate discussions among PC members in two of 2011's top-tier venues in security, one expert suggested that the problem is mostly solved as "dozens of commercial solutions exist" and research should focus on other problems, while another questioned the usefulness of the research efforts,

Memory Errors:
The Past, the Present, and the Future

Victor van der Veen[1], Nitish dutt-Sharma[1], Lorenzo Cavallaro[1,2], and Herbert Bos[1]

[1] The Network Institute, VU University Amsterdam
[2] Royal Holloway, University of London

You forgot us!


“Who are you?”


Well…
David Litchfield
Michael Howard
Alexander Anisimov
Alex Sotirov
twitch@vicar.org
kil3r
Tyler Durden
Bulba
skape
rix
w00w00
Sebastian Krahmer
Taeh Oh
Ben Hawkes
gera
Tim Newsham
Dion Blazakis
Peiter Zatko
“Mudge”
Tilo Müller
Halvar Flake
klog
Barnaby Jack
“dark spyrit”
Lamagra
Argamal
Tymm Twillman
Matt Miller
Julien
Vanegue
riq
paxteam
Thomas
Lopatic
DaveG
blexim
SoBelt
zolo
Elias Levy
“Aleph One”
Nergal
spender
Juan M. Bello Rivas
VicM
Tymm
Twillman
Alexander Peslyak
“Solar Designer”
The
Grugq
Crispin Cowan
Matt Conover
MaXX
DilDog
Nicolas Falliere.
Mark Dowd
Ken Johnson
“sh0k”

vendors released essential patches. In contrast, Bugtraq offered an effective tool to *publicly* discuss on the subject, without relying on vendors' responsiveness [88].

In 1995, Thomas Lopatic boosted the interest in memory errors by describing a step-by-step exploitation of an error in the NCSA HTTP daemon [63]. Shortly after, Peiter Zatko (Mudge) released a private note on how to exploit the now classic memory error: stack-based buffer overflows [112]. So far, nobody really discussed memory error countermeasures, but after Mudge's notes and the well-known document by Elias Levy (Aleph One) on stack smashing [4], discussions on memory errors and protection mechanisms proliferated.

The introduction of the non-executable (NX) stack opened a new direction in the attack-defense arms-race as the first countermeasure to address specifically code injection attacks in stack-based buffer overflows. Alexander Peslyak (Solar Designer) released a first implementation of an NX-like system, StackPatch [34], in April 1997. We discuss NX in Section 2.1.

A few months later, in January 1998, Cowan et al. proposed placing specific patterns (canaries) between stack variables and a function's return address to detect corruptions of the latter [29]. Further details are discussed in Section 2.2.

After the first stack-based countermeasures, researchers started exploring other areas of the process address space—specifically the heap. In early 1999, Matt Conover and the w00w00 security team were the first to describe heap overflow exploitations [27], which we discuss in Section 2.3.

On September 20, 1999, Tymm Twillman introduced format string attacks by posting an exploit against ProFTPD on Bugtraq [101]. Format string exploits became popular in the next few years and we discuss them in Section 2.4.

The idea of adding randomness to prevent exploits from working (e.g., in StackGuard) was brought to a new level with the introduction of Address Space Layout Randomization (ASLR) by the PaX Team in July 2001. We discuss the various types of ASLR and its related attacks in Section 2.5.

Around the same time as the introduction of ASLR, another type of vulnerability, `NULL` pointer dereference, a form of dangling pointer, was disclosed in May 2001. Many assumed that such dangling pointers were unlikely to cause more harm than a simple denial of service attacks. In 2007 and 2008, however, Afek and Sharabani and Mark Dowd showed that these vulnerabilities could very well be used for arbitrary code injection as well [1,37]. Unfortunately, specific defenses against dangling pointers are still mostly research-driven efforts [2].

Due to space limitations, a number of historical details were omitted in this paper. The interested reader can refer to [102] for more information.

“What just happened?”

Not everyone likes academics

← **Plaatsen**

Thomas H. Ptacek @tqbf · 7 jun. 2018
Paper link: arxiv.org/pdf/1711.01254...

For double fetch background, Wang is good: usenix.org/system/files/c...

2 8 21

grsecurity @grsecurity · 7 jun. 2018
I mentioned this before, but everyone seems to have forgotten sgrakkyu/twiz exploited "double fetch" before "double fetch" was a thing, back in 2007: phrack.org/issues/64/6.ht... (2.4.2), see also this from 2008: osronline.com/article.cfm?ar...

2 17 44

Halvar Flake @halvarflake · 9 jun. 2018
nobody has forgotten :)

1 2

grsecurity @grsecurity · 9 jun. 2018
I guess it's why those things were cited in that USENIX paper. Oh wait...

1 5

Halvar Flake
@halvarflake

My ability to get upset at academics not citing extraacademic sources properly was exhausted ca. 2008, so kudos to you for still finding the energy to care.

Dan Kaminsky @dakami · 18 nov. 2018
You should write it. We both should write more long form.
1 18

thaddeus e. grugq thegrugq@infosec.exch... @thegr... · 18 nov. 2018
I am and I have written long form. It is encouraging to see it show up as a strong influence on other people’s pieces that get wide syndication and give no credit to the OP ;)

No more free thoughts!
1 10

Dan Kaminsky @dakami · 18 nov. 2018
“It is amazing what you can accomplish if you don’t care who gets credit” :)
1 2 22

thaddeus e. grugq thegrugq@infosec.exch... @thegr... · 18 nov. 2018
Is it? Must be why those academics are so lax about citation.
3 1 7

Halvar Flake @halvarflake · 18 nov. 2018
in my experience, they are not lax w citing academic pubs, everything else is deemed as "not citeable"...
4 10

thaddeus e. grugq thegrugq@infosec.exch... @thegr... · 18 nov. 2018

The likely reason for this is they know the authors of the “not citeable” content will never be on a review board. 🤓 //cc @SteveBellovin @gannimo @matthew_d_green @mattblaze @johnregehr

4 1 9

Mathias Payer @gannimo · 18 nov. 2018

Trust me, I've argued against papers that simply reimplement hacker work. For example, if it is published as a phrack article, I consider this prior work and, if your work is similar, it must be compared against the phrack article!

4 5 30

Steven M. Bellovin @SteveBellovin · 18 nov. 2018

Strong agreement

1 1

Dan Kaminsky @dakami · 18 nov. 2018

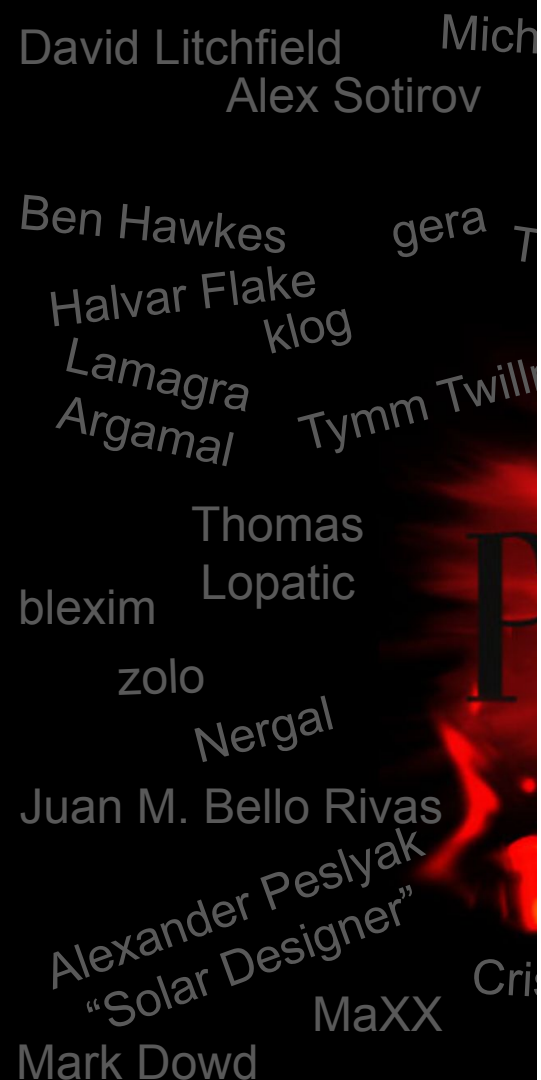
they just...copy the methods, rote, from outside their little island? people do this? and it has to be argued against?

They at least add some analysis and context?

1

Mathias Payer @gannimo · 18 nov. 2018

Yes, unfortunately. But I would not call it copy, it's more a reinvention. Don't attribute to malice what can be explained by a reinvention of the same idea and unawareness of the other side



Herbert Bos @herbertbos · 18 nov. 2018

Colour me naive, but i also think that in most cases the authors are simply not aware of prior art in a different community.

3 1 6

Halvar Flake @halvarflake · 18 nov. 2018

I have seen both variants -- not being aware, and being aware and omitting it, betting on the review committee not being aware. The latter was more rare, but when I saw it it was more infuriating. :-) (and the people that did it are on my very short permanent shitlist)

1 1 7

Mathias Payer
@gannimo

Hahaha, yeah, I had a recent issue with some systems researchers whom I contacted 2-3 times and they continuously ignored our (academic) work. Fun times...

Post vertalen

12:44 p.m. · 18 nov. 2018

ROP

Make it concrete, please!

Dave Aitel @daveaitel · 6 nov.

ROP?

2 7 583

Michael Brown
@MichaelBrownUC

Lol this is the most egregious example

Post vertalen

4:25 a.m. · 6 nov. 2023 · **106** Weergaven

Let’s have a look

1996

.oO Phrack 49 Oo.

Volume Seven, Issue Forty-Nine File 14 of 16

BugTraq, r00t, and Underground.Org

bring you

Smashing The Stack For Fun And Profit

Aleph One

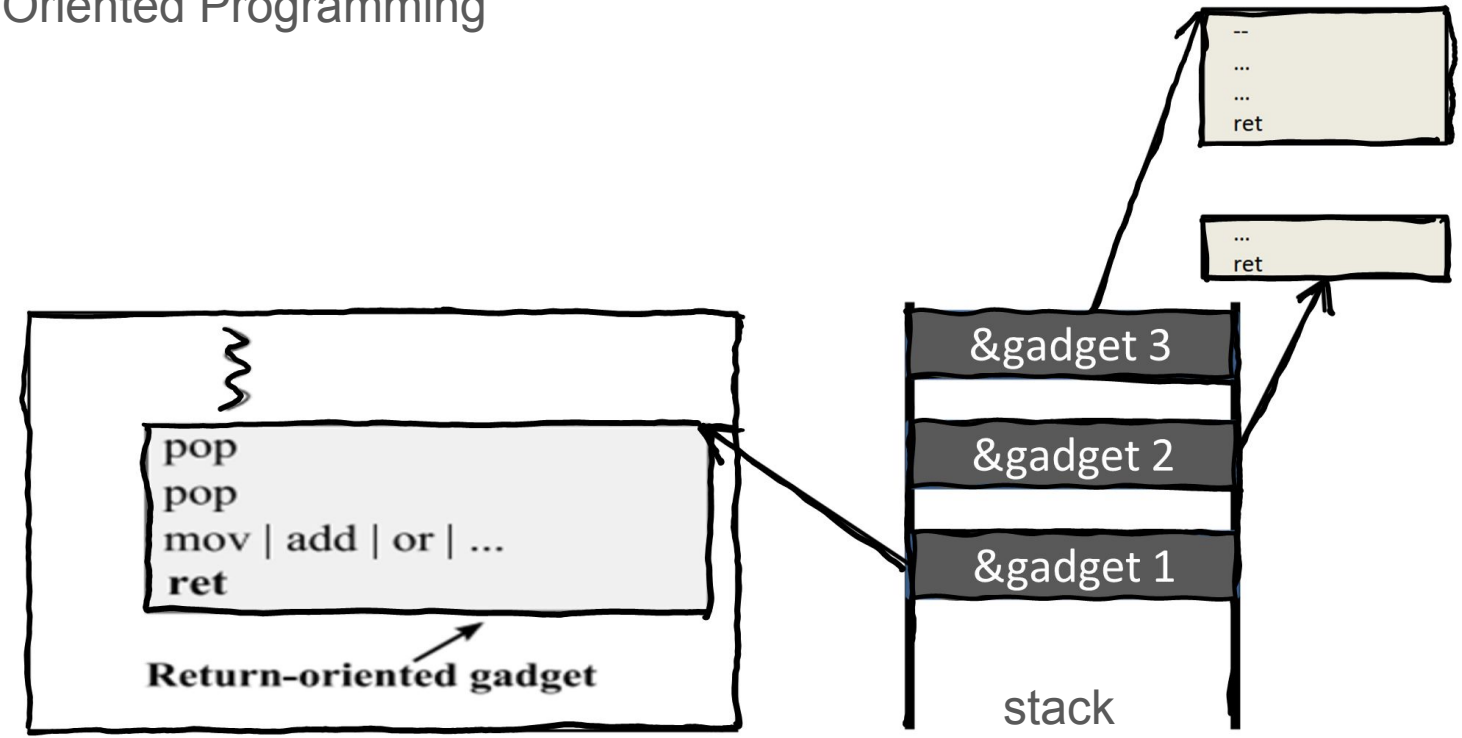
aleph1@underground.org

ROP: Memory is now W^X

```
$ cat /proc/self/maps
55c684813000-55c68481b000 r-xp 00000000 103:02 5505049      /bin/cat
55c684a1a000-55c684a1b000 r--p 00007000 103:02 5505049      /bin/cat
55c684a1b000-55c684a1c000 rw-p 00008000 103:02 5505049      /bin/cat
55c6852c9000-55c6852ea000 rw-p 00000000 00:00 0             [heap]
7f2fff97b000-7f2fffc5a000 r--p 00000000 103:02 21497232     /usr/lib/locale/locale-archive
7f2fffc5a000-7f2fffe41000 r-xp 00000000 103:02 19661405     /lib/x86_64-linux-gnu/libc-2.27.so
7f2fffe41000-7f3000041000 ---p 001e7000 103:02 19661405     /lib/x86_64-linux-gnu/libc-2.27.so
7f3000041000-7f3000045000 r--p 001e7000 103:02 19661405     /lib/x86_64-linux-gnu/libc-2.27.so
7f3000045000-7f3000047000 rw-p 001eb000 103:02 19661405     /lib/x86_64-linux-gnu/libc-2.27.so
7f3000047000-7f300004b000 rw-p 00000000 00:00 0
7f300004b000-7f3000074000 r-xp 00000000 103:02 19660823     /lib/x86_64-linux-gnu/ld-2.27.so
7f3000239000-7f300025d000 rw-p 00000000 00:00 0
7f3000274000-7f3000275000 r--p 00029000 103:02 19660823     /lib/x86_64-linux-gnu/ld-2.27.so
7f3000275000-7f3000276000 rw-p 0002a000 103:02 19660823     /lib/x86_64-linux-gnu/ld-2.27.so
7f3000276000-7f3000277000 rw-p 00000000 00:00 0
7ffc0ecd4000-7ffc0ecf5000 rw-p 00000000 00:00 0             [stack]
7ffc0ed50000-7ffc0ed53000 r--p 00000000 00:00 0             [vvar]
7ffc0ed53000-7ffc0ed54000 r-xp 00000000 00:00 0             [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0     [vsyscall]
```

Solution: reuse code already present

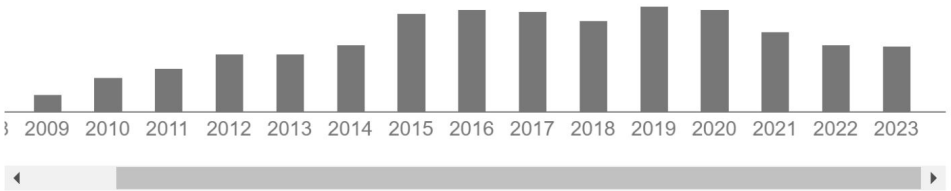
"Return Oriented Programming"



ROP

The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)

Authors	Hovav Shacham
Publication date	2007/10/28
Book	Proceedings of the 14th ACM conference on Computer and communications security
Pages	552-561
Description	We present new techniques that allow a return-into-libc attack to be mounted on x86 executables that calls *no functions at all*. Our attack combines a large number of short instruction sequences to build *gadgets* that allow arbitrary computation. We show how to discover such instruction sequences by means of static analysis. We make use, in an essential way, of the properties of the x86 instruction set.
Total citations	Cited by 1949



Scholar articles	The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86) H Shacham - Proceedings of the 14th ACM conference on Computer …, 2007 Cited by 1949 Related articles All 55 versions

But was it the first?

Getting around non-executable stack (and fix)
From: solar () FALSE COM (Solar Designer)
Date: Sun, 10 Aug 1997 17:29:46 -0300

Hello!

I finally decided to post a return-into-libc overflow exploit. This method has been discussed on linux-kernel list a few months ago (special thanks to Pavel Machek), but there was still no exploit. I'll start by speaking about the fix, you can find the exploits (local only) below.

[...]

You can find the fixed version of my non-executable stack Linux kernel patch at http://www.false.com/security/linux-stack/.

[...]

Actually, using this method it is possible to call two functions in a row if the first one has exactly one parameter. The stack should look like this:

```
                              pointer to "/bin/sh"
                              pointer to the UID (usually to 0)
                              pointer to system()
 stack pointer ->             pointer to setuid()
```

Solar Designer 1997

The ability to overwrite the stack with arbitrary data is very powerful. Besides return addresses the stack is also used to save register values and to hold variables. Most programs have segments of code that look like:

```
     restore some registers from the stack
     return from subroutine
```

If an attacker knows the address of such code, he can provide register contents on the stack and set the return address to point to this code. When the next return happens, registers are set with whatever values he put on the stack, another return is done pulling another address off the stack. Say the next return address on the stack pointed to code that trapped to the system call vector. We just put arbitrary values in registers and then trapped to the system - we have the ability to do arbitrary system calls. All the code that was executed was from the code segment.

By controlling the stack, an attacker can cause execution to thread through segments of existing code with a great degree of freedom. The attacks have to accurately compute the location of stack positions and code addresses so the attack is definitely a lot harder than the cookie-cutter stack overflows that you see today, but its still ``just a simple matter of coding''.
...

Tim Newsham, Bugtraq, 1997

But was it the first?

2001 Nergal describes fully featured ROP attack

`The advanced return-into-lib(c) exploits: PaX case study`

`Phrack Volume 0x0b, Issue 0x3a, Phile #0x04 of 0x0e`

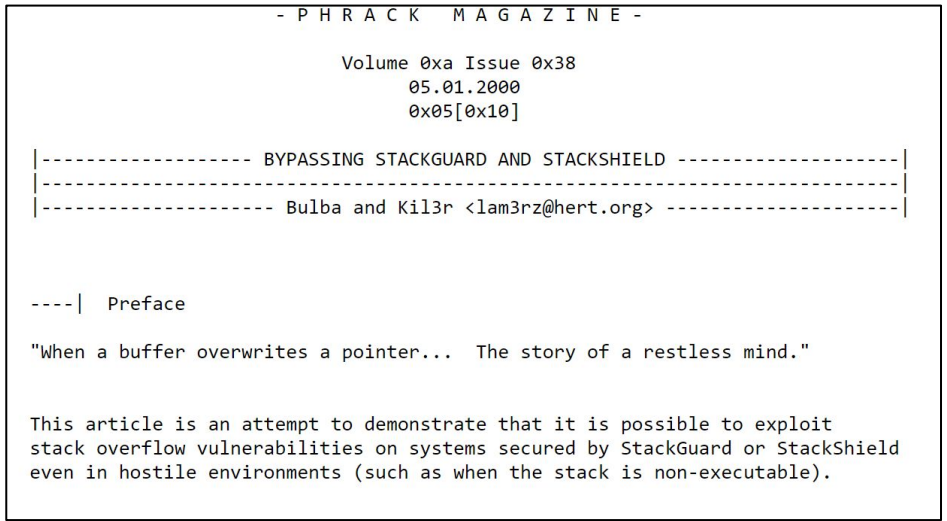
```
                              ==Phrack Inc.==

                  Volume 0x0b, Issue 0x3a, Phile #0x04 of 0x0e

|=------------=[ The advanced return-into-lib(c) exploits: ]=------------=|
|=-------------------------=[ PaX case study ]=--------------------------=|
|=----------------------------------------------------------------------=|
|=-----------------=[ by Nergal <nergal@owl.openwall.com> ]=--------------=|


                                                  May this night carry my will
                        And may these old mountains forever remember this night
                                                  May the forest whisper my name
                    And may the storm bring these words to the end of all worlds

                                                                Ihsahn, "Alsvartr"


--[ 1 - Intro

 1 - Intro

 2 - Classical return-into-libc

 3 - Chaining return-into-libc calls
   3.1 - Problems with the classical approach
   3.2 - "esp lifting" method
   3.3 - frame faking
   3.4 - Inserting null bytes
   3.5 - Summary
   3.6 - The sample code

 4 - PaX features
   4.1 - PaX basics
   4.2 - PaX and return-into-lib exploits
   4.3 - PaX and mmap base randomization

 5 - The dynamic linker's dl-resolve() function
   5.1 - A few ELF data types
   5.2 - A few ELF data structures
   5.3 - How dl-resolve() is called from PLT
   5.4 - The conclusion

 6 - Defeating PaX
   6.1 - Requirements
   6.2 - Building the exploit

 7 - Misc
   7.1 - Portability
   7.2 - Other types of vulnerabilities
   7.3 - Other non-exec solutions
   7.4 - Improving existing non-exec schemes
   7.5 - The versions used

 8 - Referenced publications and projects
```

But was it the first?

2002, David Litcfield: “Non-Stack based exploitation” → essentially ret2libc on Win32

2002, Bulba and Kil3r: “Bypassing StackGuard and StackShield” (Phrack)
→ used existing code to jump to shellcode on stack

2004, Jack and Nemo: “Jump Oriented Programming” on the SPARC architecture (Phrack): use existing code to jump to arbitrary addresses.



```
                    - P H R A C K   M A G A Z I N E -

                          Volume 0xa Issue 0x38
                               05.01.2000
                               0x05[0x10]

|------------------- BYPASSING STACKGUARD AND STACKSHIELD --------------------|
|-------------------------------------------------------------------------------|
|--------------------- Bulba and Kil3r <lam3rz@hert.org> ----------------------|


----|  Preface

"When a buffer overwrites a pointer...  The story of a restless mind."


This article is an attempt to demonstrate that it is possible to exploit
stack overflow vulnerabilities on systems secured by StackGuard or StackShield
even in hostile environments (such as when the stack is non-executable).
```

But was it the first?

In 2005, Sebastian Krahmer published:
“x86-64 buffer overflow exploits and
the borrowed code chunks exploitation technique”

x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique

Sebastian Krahmer *krahmer@suse.de*

September 28, 2005

Abstract

The x86-64 CPU platform (i.e. AMD64 or Hammer) introduces new features to protect against exploitation of buffer overflows, the so called No Execute (NX) or Advanced Virus Protection (AVP). This non-executable enforcement of data pages and the ELF64 SystemV ABI render common buffer overflow exploitation techniques useless. This paper describes and analyzes the protection mechanisms in depth. Research and target platform was a SUSE Linux 9.3 x86-64 system but the results can be expanded to non-Linux systems as well.
search engine tag: SET-krahmer-bccet-2005.

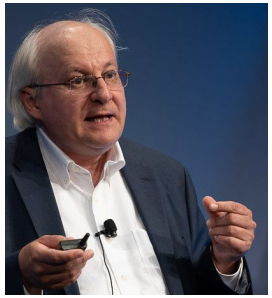
Contents

Were there any differences?

Yes.

The previous attacks used short sequences as glue in combining the invocations of functions in libc or in jump-starting the execution of attacker-injected code. Our technique shows that short code sequences, combined in appropriate ways, can express any computation an attacker might want to carry out, without the use of any functions.

Of the previous uses discussed here, Krahmer's borrowed code chunks exploitation technique [15] is the closest to ours. Krahmer uses static analysis to look for register-pop sequences. He describes a shellcode-building tool that combines these sequences to allow arbitrary arguments to be passed to libc functions. However, exploits constructed using Krahmer's techniques are still straight-line limited and still rely on specific functions in libc— like other traditional return-into-libc attacks, and unlike the new attack we propose.



“framing (a certain kind of) exploitability as a mathematical property that can be proved as a theorem.”

“It opened the floodgates”

“When non academics develop something [...], they use an example implementation to demonstrate a broader point that they’re making. Only, they suck at making it clear that “here is theory X, and a simple demonstration of the theory is present here as X1”

Did Hovav mention the original work?

Yes.

1.2.4 *Previous Uses of Short Sequences in Attacks*

Some previous return-into-libc attacks have used short code snippets from libc. Notably, code segments of the form pop %*reg*; ret to set registers have been used to set function arguments on architectures where these are passed in registers, such as SPARC [20] and x86-64 [15]. Other examples are Nergal's "pop-ret" sequences [21] and the "register spring" technique introduced by dark spyrit [6] and discussed by Crandall, Wu, and Chong [5]. Our attack differs

Meanwhile, everybody cites Hovav's paper…

… and nobody mentions Krahmer, Nergal, Newsham, or Solar Designer

Mathias Payer @gannimo · 18 nov. 2018

Don't attribute to malice what can be explained by a reinvention of the same idea and unawareness of the other side

Rare?

Herbert Bos @herbertbos · 5 nov.

Hey Infosec twitter/X, I am looking for examples where academic researchers did not credit hackers/non-academic security researcher, or vice versa.

Just reply here, or DM me if you don't want do this in public.

· 6 nov.

Als antwoord op @gannimo en @herbertbos

Let’s not forget the dude who published the fist academic research paper on improving AFL and now everyone cites that work, instead of @lcamtuf

[redacted] · 6 nov.

Als antwoord op @herbertbos

When I co-authored this paper [redacted].github.io/papers/[redacted], the academics on the project outright refused to include citations I wanted to reverse engineering publications that had helped me because they were "not academic enough".

4 4 21


Man, i got really angry last night
Just seeing it all again pissed me off so much.
Haha. It’s stupid, since it isn’t like i get anything if i was cited
David Litchfield
Michael Howard
Alexander Anisimov
Alex Sotirov
twitch@vicar.org
kil3r
Tyler Durden
Bulba
Sebastian Krahmer
skape
rix
w00w00
Taeh Oh
Ben Hawkes
Tilo Müller
Halvar Flake
klog
arnaby Jack
Lamagra
dark spyrit”
Argamal
Tymm
Julien
paxteam
Vanegue
Thomas
Lopatic
DaveG
blexim
SoBelt
zolo
Elias Levy
Aleph One
Nergal
spender
Juan M. Bello Rivas
Tymm
Twillman
Alexander Peslyak
“Solar Designer”
The
Grugq
Crispin Cowan
Matt Conover
MaXX
DilDog
Nicolas Falliere.
Ken Johnson
“sh0k”
Mark Dowd

[redacted] · 7 nov.

Als antwoord op @[redacted]es @halvarflake en @herbertbos

Yeah been there, on anything technical sources that are academic enough are either so abstract that they are useless, or at worst technically incorrect.

3

[redacted]ph · 19 jun. 2022

The reality is that academics make the best trolls because even they believe a lot of the BS they spout.

Why Professors Are Writing Crap That Nobody Reads

Half of academic papers are never read by anyone other than their authors, peer reviewers, and journal editors.

NI.A by Editor — April 15, 2022 in Around The World, Education, Offbeat

127 1 AA

Reading Time: 2 mins read

aborsy on Aug 2, 2020 | prev [–]

The vast majority of academics are there for power and status. I rarely see a true scholar, and nearly always bizarre characters and politicians to say the least.

Politics in academia is especially vicious.

Security Research: Non Academic  Academic

What happened?

A hacker culture emerges

At MIT

The Tech Model Railroad Club

The first computer wizards who called themselves hackers started underneath a toy train layout at MIT's Building 20

Marvin Minsky - champion of hackers

Hackers

“These kids are like superheroes.”

“They have special abilities.”

“They often don’t fit in.”

Yes.

Also for academics

Perhaps less cool

Independent ways

“Industry/gov” ⟷ “academics” ⟷ “Hackers/crackers”

Academic venues would not/rarely accept attacks

- IEEE S&P 1980
- ACSAC 1985
- USENIX Security 1988
- ACM CCS 1993
- NDSS 1993

Separate venues emerged for hackers community

- CCC 1984
- DEF CON 1993
- Black Hat 1997

Why the bad blood?

arrogance?

motives?

research culture?

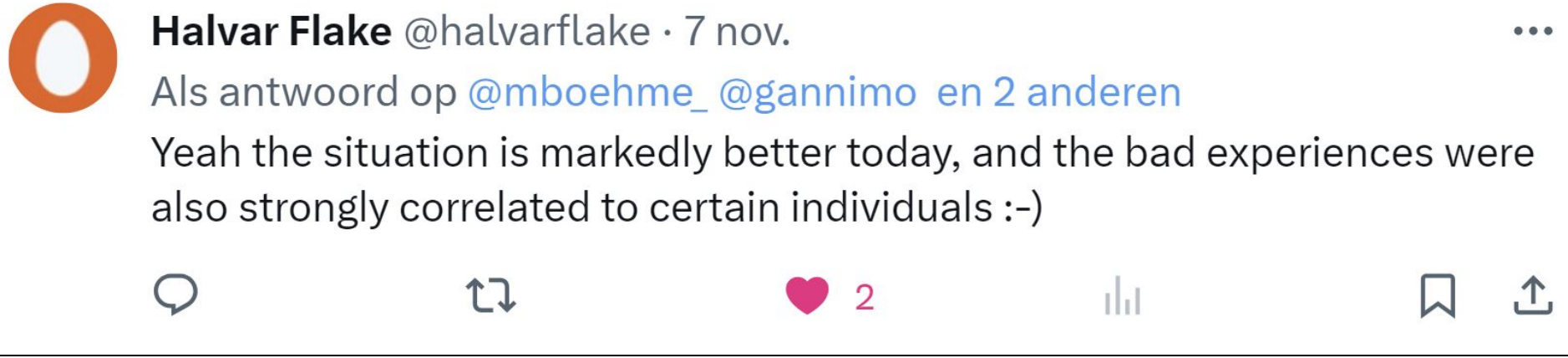
lack of recognition?

The communities getting closer again

Hackers in academic communities

Offensive research recognized

Recognition from non academic security community (and vice versa?)



So… back to memory safety

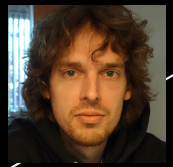
2

“Flipping bits” -- Y. Kim, R. Daly, J.S. Kim, C. Fallin, J-H Lee, D. Lee, C.B. Wilkerson, K. Lai, O. Mutlu

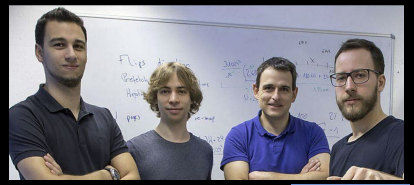
“Exploiting the DRAM Rowhammer bug to gain kernel privileges”
-- Mark Seaborn, Thomas Dullien

“Meltdown: Reading Kernel Memory from User Space”
“Spectre Attacks: Exploiting Speculative Execution”
-- Jan Horn, Werner Haas, Thomas Prescher, Daniel Gruss, Moritz Lipp, Stefan Mangard, Michael Schwarz, Paul Kocher, Daniel Genkin, Mik Hamburg, Yuval Yarom

VUSec



“Dedup est machina”
-- Erik Bosman et al.



“Row Hammer Refresh Command” (Patent)
-- K. S. Bains, J. B. Halbert, C. P. Mozak, T. Z. Schoenborn, and Z. Greenfield

CYBER.WTF

“Negative Result: Reading Kernel Memory From User Mode”
-- Anders Fogh

rowhammer.js -- D. Gruss, C.Maurice, S.Mangard

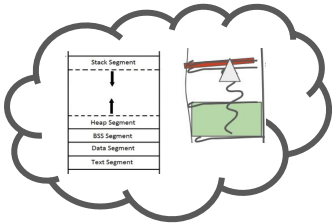
‘11 ‘12 ‘13 ‘14 ‘15 ‘16 ‘17 ‘18 ‘19 ‘20 ‘21 ‘22 ‘23 ‘24

VUSec
UNIVERSITY OF MICHIGAN
TU Graz
WPI
KU LEUVEN
ETH

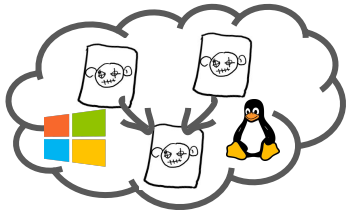
Memory (Un)Safety

Software

Traditional exploits

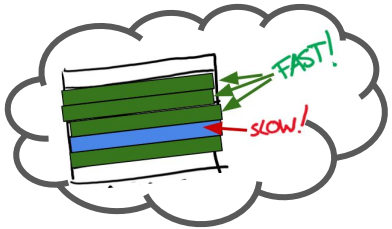


Side channels & artefacts

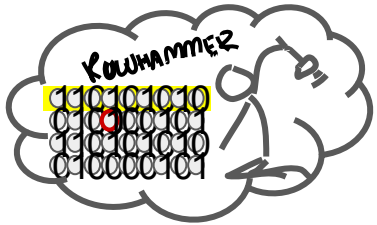


Hardware

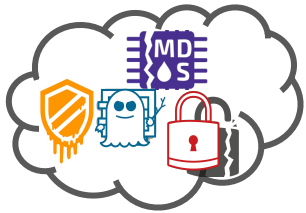
Hardware side channels



Rowhammer



Transient execution

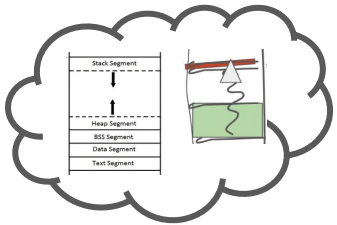


3

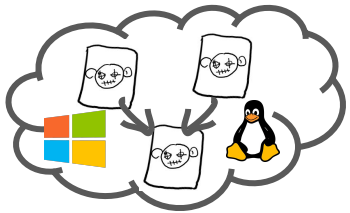
Memory (Un)Safety

Software

Traditional exploits

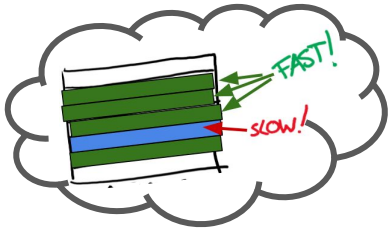


Side channels & artefacts

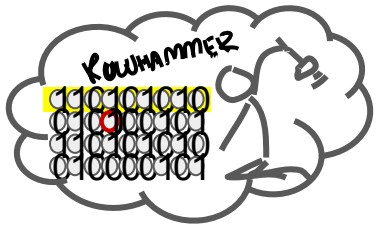


Hardware

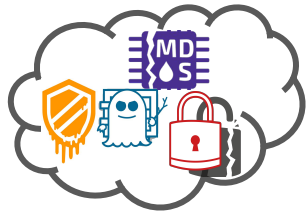
Hardware side channels



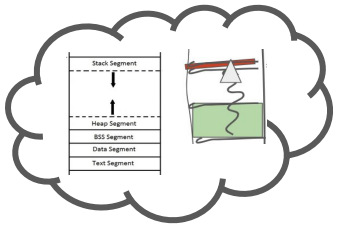
Rowhammer



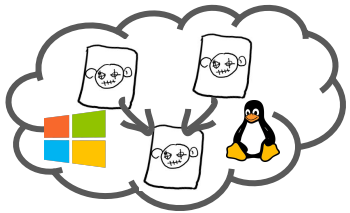
Transient execution



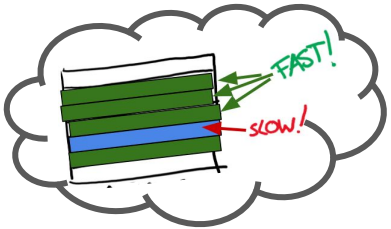
Traditional exploits



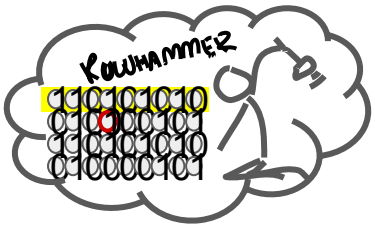
Side channels & artefacts



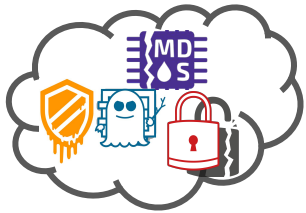
Hardware side channels



Rowhammer



Transient execution



Traditional exploits

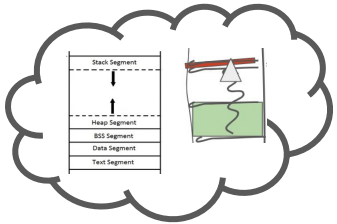
Side channels & artefacts

Hardware side channels

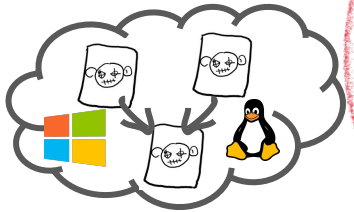
Rowhammer

Transient execution

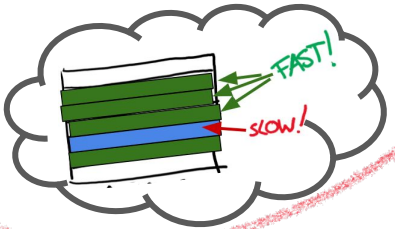
Traditional exploits



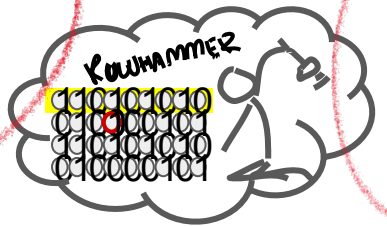
Side channels & artefacts



Hardware side channels

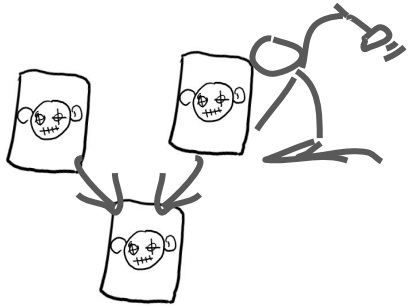


Rowhammer

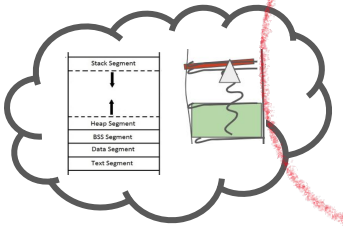


Transient execution

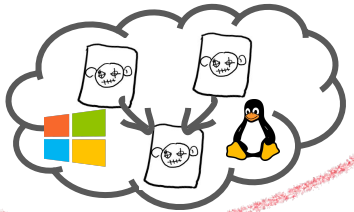
- Dedup est machina, S&P'16
- Flip Feng Shui, USENIX Security'16



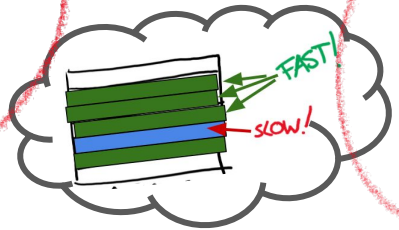
Traditional exploits



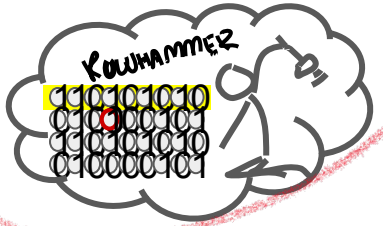
Side channels & artefacts



Hardware side channels

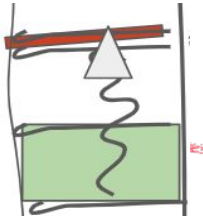


Rowhammer



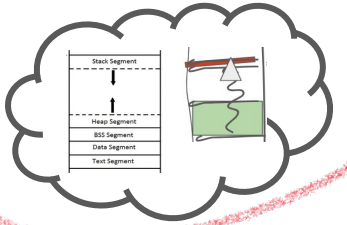
Transient execution

- Speculative Probing (“BlindSide”), CCS’20
- PAC-MAN Attack, ISCA’22
- [embargo], [embargo]’24

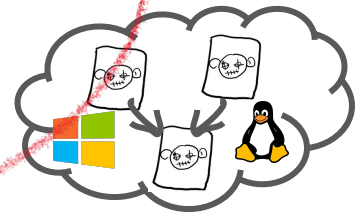


```
if (slow-condition) {
  call [corrupt-ptr];
}
```

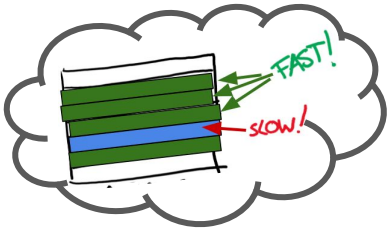
Traditional exploits



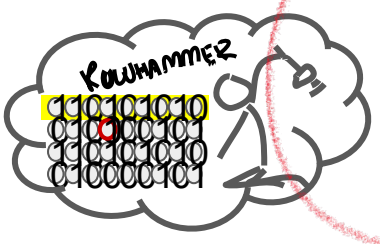
Side channels & artefacts



Hardware side channels



Rowhammer



Transient execution

Potential gadget:

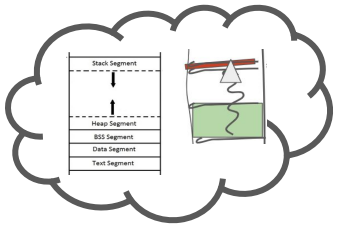
```
if (index < bounds) { // not attacker-controlled
  data = array1[index];
  val = array2[data];
}
```

Usable gadget:

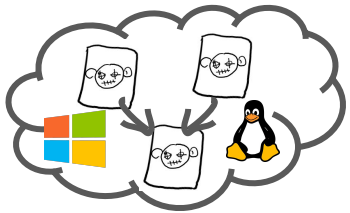
```
if (index < bounds) { // attacker-controlled!
  data = array1[index];
  val = array2[data];
}
```

SpecHammer, S&P'22

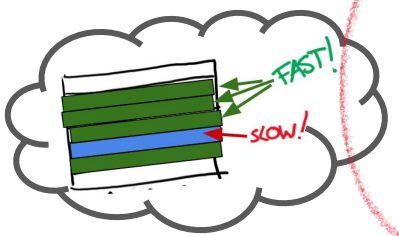
Traditional exploits



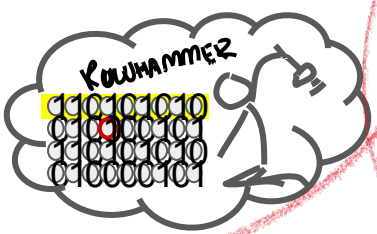
Side channels & artefacts



Hardware side channels



Rowhammer

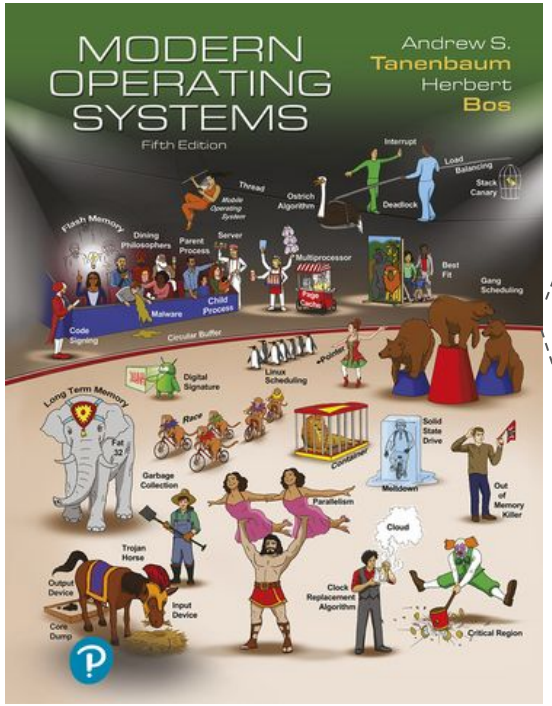


Transient execution

Memory (Un)Safety: “All Things Under The Hood”

We should not need to be aware of them

Abstraction: Fundamental Tenet of Software Engineering



SEC. 1.1 WHAT IS AN OPERATING SYSTEM? 5

This abstraction is the key to managing all this complexity. Good abstractions turn a nearly impossible task into two manageable ones. The first is defining and implementing the abstractions. The second is using these abstractions to solve the problem at hand. One abstraction that almost every computer user understands is the file, as mentioned above. It is a useful piece of information, such as a digital photo, saved email message, song, or Web page. It is much easier to deal with photos, emails, songs, and Web pages than with the details of SATA (or other) disks. The job of the operating system is to create good abstractions and then implement and manage the abstract objects thus created. In this book, we will talk a lot about abstractions. They are one of the keys to understanding operating systems.

This point is so important that it is worth repeating but in different words. With all due respect to the industrial engineers who so very carefully designed the Apple Macintosh computers (now known simply as "Macs"), hardware is grotesque. Real processors, memories, Flash drives, disks, and other devices are very complicated and present difficult, awkward, idiosyncratic, and inconsistent interfaces to the people who have to write software to use them. Sometimes this is due to the need for backward compatibility with older hardware. Other times it is an attempt to save money. Often, however, the hardware designers do not realize (or care) how much trouble they are causing for the software. One of the major tasks of the operating system is to hide the hardware and present programs (and their programmers) with nice, clean, elegant, consistent, abstractions to work with instead. Operating systems turn the awful into the beautiful, as shown in Fig. 1-2.

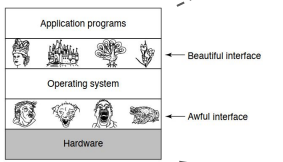


Figure 1-2. Operating systems turn awful hardware into beautiful abstractions.

It should be noted that the operating system's real customers are the application programs (via the application programmers, of course). They are the ones who deal directly with the operating system and its abstractions. In contrast, end users deal with the abstractions provided by the user interface, either a command-line shell or a graphical interface. While the abstractions at the user interface may be similar to the ones provided by the operating system, this is not always the case. To make this point clearer, consider the normal Windows desktop and the

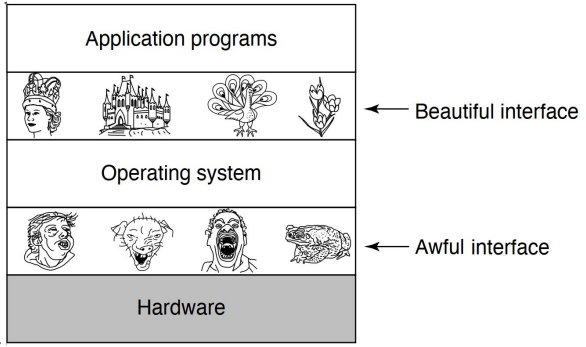


Figure 1-2. Operating systems turn awful hardware into beautiful abstractions

Abstractions, layers, partitioning

Fundamental concepts

We *need* them to understand the world

Vulnerabilities are where abstractions break down

Conclusions (1)

Abstractions considered harmful *and* essential

To write secure code you must know everything (?!)

Memory Corruption Phase 3: The MultiVerse

VUSec
https://vusec.net
info@vusec.net
@vu5ec

Conclusion (2)

We have treated the non-academic security community poorly.

We owe these people a lot.

VUSec

https://vusec.net

info@vusec.net

@vu5ec

Modest proposal

Academic community

Always cite any available prior art
(But do not reject papers because someone somewhere wrote a blog post)

Cite the *earliest* sources in addition to (recent) academic work

Non-academic community

Work on making it easier to find stuf

https://vusec.net
info@vusec.net
@vu5ec

Less Modest proposal

Academic community

Be more accepting toward papers from non-academic researchers
Explain better what we expect

Recognize the achievements of hackers.Why doesn't this conference have, say, a

Dark Spyrit Award for Embedded Systems Security? or a

Dan Kaminsky Award For Best Internet Security Achievement ?

https://vusec.net
info@vusec.net
@vu5ec