What hacker research taught me

Sergey Bratus
Dartmouth College

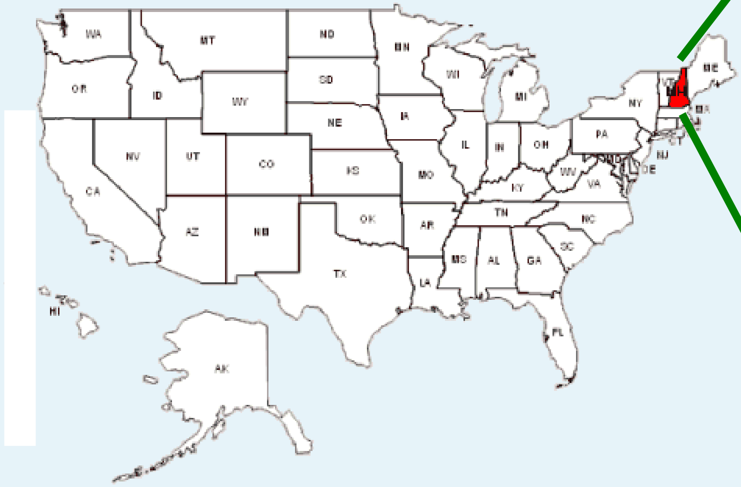
TROOPERS10
IT-Security Conference & Workshops

What this is about?

- A personal rant / ”quest”
- The fun and huge presumpion of defining ”**hacking**” :-)
- An excuse for citing Phrack, Uninformed, Defcon/Recon/Shmoocon/Toorcon/...
- Realization that ”hacking” goes to the **heart** of fundamental Computer Science problems

Who am I?

- Dartmouth College
- ”Research Assistant Professor”



”Hackers!”

- The Adversary
- Harbingers of Future Technologies
- Engineers / researchers of a unique specialization (not yet formally defined)
  - **”What kind of engines?”**

”Hackers!”

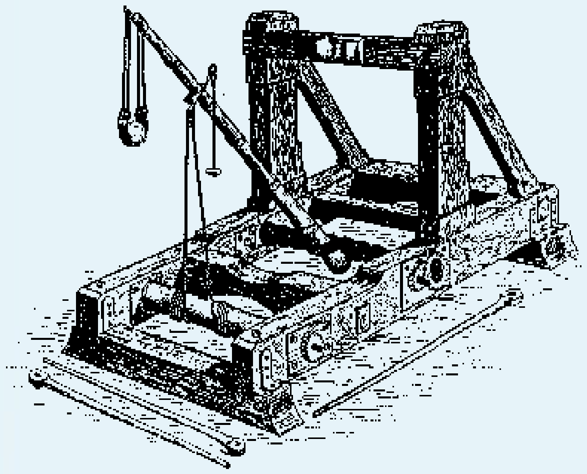
- The Adversary
  - Media + politicians
    Notice how they are always selflessly saving us from something or other?
  - ”*We may need to forego certain freedoms to make the Internet a safer place*”
    e.g., John Markoff, NYT, Feb. 2009 (paraphrased)
  - Enough said :-(

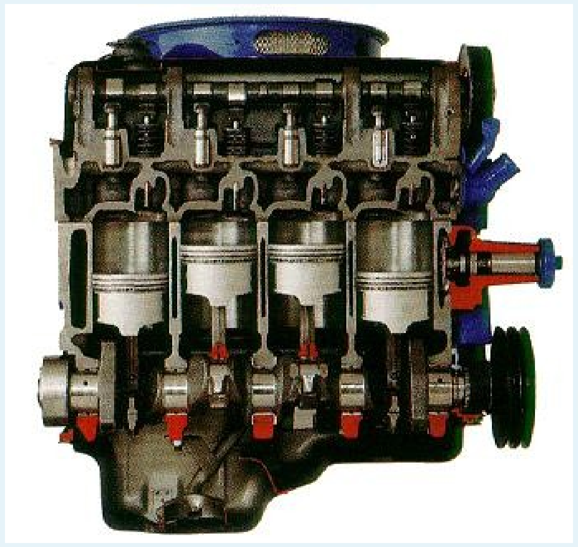
”Hackers!”

- Harbingers of the Future
  - Hackers realized the potential of universal, ubiquitous, cheap connectivity long before actual technology owners
    Emmanuel Goldstein, Toorcamp '09
  - Phone companies initially expected their revenues to come from ”customers” connecting to (for-pay) ”services”, not subscribers talking with other subscribers
    Andrew Odlyzko (AT&T Research)
    ”*Content is not King*”, 2001

”Hackers!”

- Engineers of a unique kind / not yet formally defined <u>discipline of engineering</u>
- ***”What kind of engines?”***





”Hackers!”

- **Engineers** of a unique kind / not yet formally defined <u>discipline of engineering</u>
- ”*What kind of engines?*”
  - **What kind of fundamental, hard problems are they up against?**
    - E.g.: energy to motion is hard, storing energy is hard, etc.
  - **What laws of nature are involved?**
    - E.g.: Newtonian conservation laws, laws of thermodynamics, P != NP (?), ...

The defining challenges

- Something really, provably hard (as in ”NP”, RSA, other ”God's own math”)
- Something really human, what we must do every day

The defining challenges of Hacking as a discipline

- Something really, provably <u>hard</u> (as in ”NP”, RSA, other ”God's own math”)

Composition

- Something really human, what we <u>must</u> do every day

The defining challenges of Hacking as a discipline

- Something really, provably <u>hard</u> (as in ”NP”, RSA, other ”God's own math”)

Composition

- Something really human, what we <u>must</u> do every day

Trust

Composition is hard

- Informally: even if non-trivial <u>properties of parts</u> are known, the same properties of the <u>combined</u> system **cannot** be deduced by <u>any general formal algorithm</u>
- A.k.a. ”**Security is not composable**”
- Kind of formally:

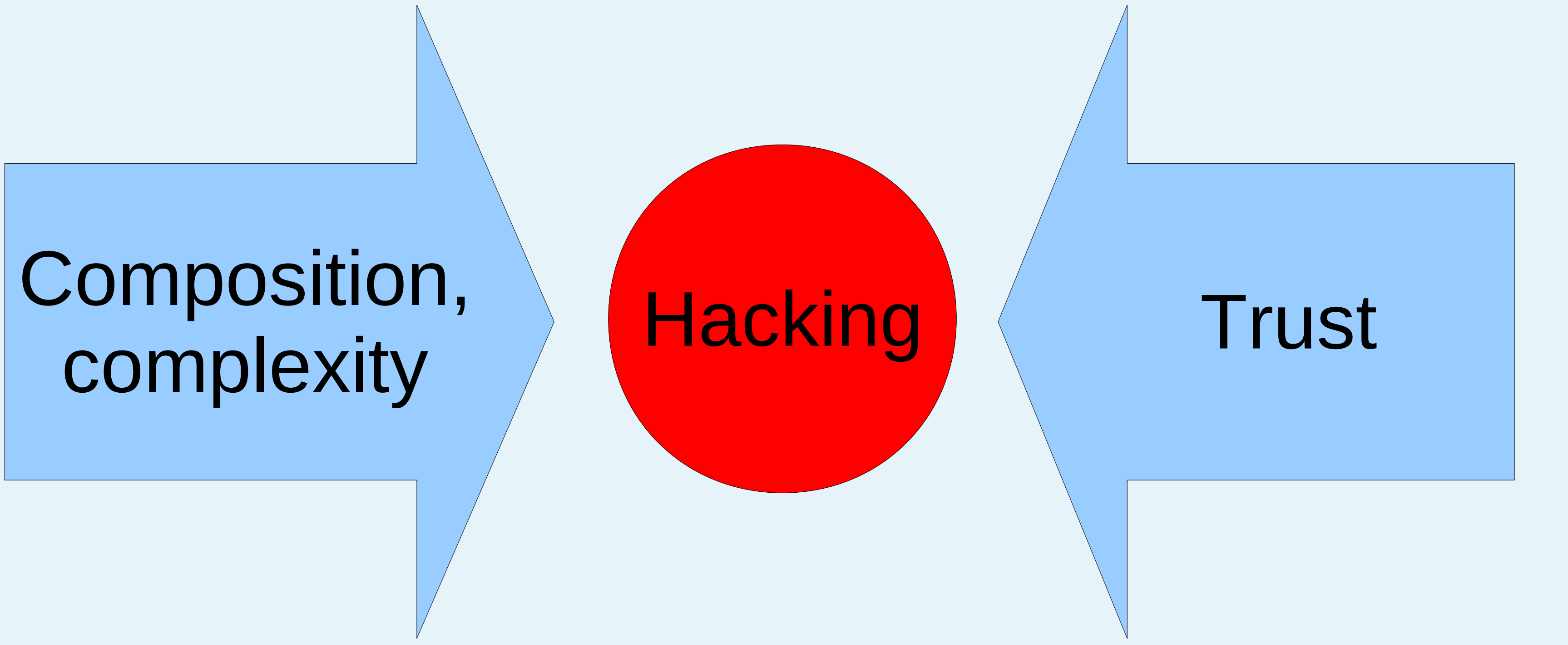
  Rice's Theorem ~ Halting problem
- There is a reason why humans don't deal well with **complexity**

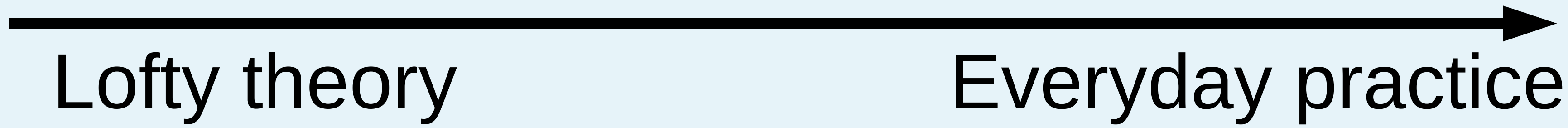
Trust is crucial to human activities

- Economies and ways of life are defined by levels of trust
  - ”High Trust” vs ”Low Trust” societies theory
  - Personal experience :-)
- FX, Bratzke @ SiS 2007:

  ”Pragmatically, InfoSec is about ***working towards computer systems we can finally <u>trust</u>****”*

The discipline of hacking at a glance





Hacking as R&D

Hacking (n.):

the capability/skill set to question and verify trust (security, control) assumptions expressed in complex software and hardware (as well as in human-in-the-loop processes that use them)

...here's hoping for WIKIPEDIA *The Free Encyclopedia* :-)

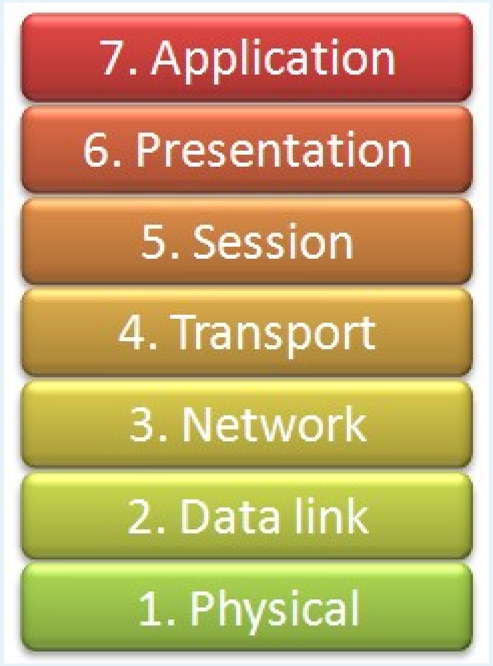
Hacking as R&D

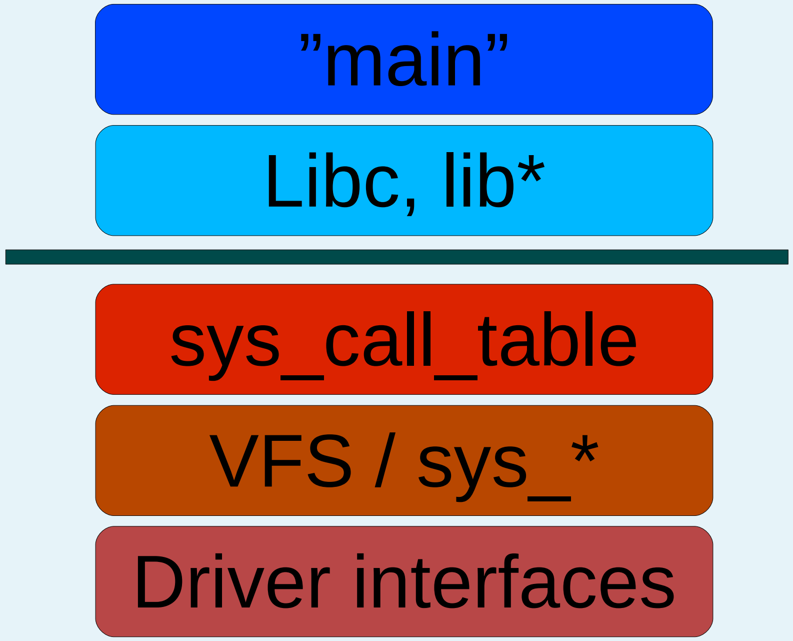
Exploitation (n.):

causing a (complex) computer or human-computer system to behave contrary to the trust assumptions and/or expectations of its designers or operators

Lesson 1: **Look across layers**

- Humans aren't good at handling complexity
- Engineers fight it by <u>layered</u> designs:





Layers are magical

- They just work, especially the ones below
- One layer has proper security =>
  the whose system is trustworthy

Layers are magical

- They just work, especially ones below
- One layer has proper security =>
  the whose system is trustworthy

NOT! ;-)

Layers are magical

- *”They just work, especially ones below”*
- *”One layer has proper security =>*
  the whose system is trustworthy”
- **In real life, layer boundaries become <u>boundaries of competence</u>**

Layers are magical

- *”They just work, especially ones below”*
- *”One layer has proper security =>*
  the whose system is trustworthy”
- In real life, layer boundaries become <u>boundaries of competence</u>
- Hacker methodology in a word:

<u>cross-layer</u> approach

Best OS course reading ever :-)

- Phrack 59:5, palmers@team-teso
  "5 Short Stories about execve",
  "Deception in depth"

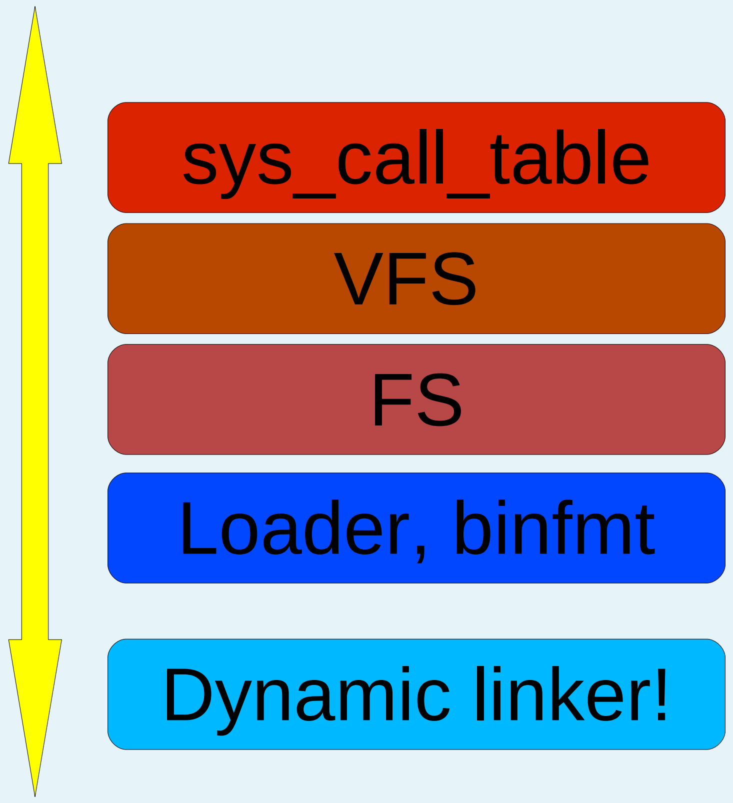
sys_call_table	sys_execve, "The Classic"
VFS	do_execve, "The Obvious"
FS	open_exec, "The Waiter"
Loader, binfmt	load_binary, "The Nexus"
Dynamic linker!	mmap/mprotect, "The Lord"

”Cross-layer approach” in action

- Phrack 59:5, palmers@team-teso

  ”5 Short Stories about execve”,

  ”Deception in depth”



sys_execve, ”The Classic”

do_execve, ”The Obvious”

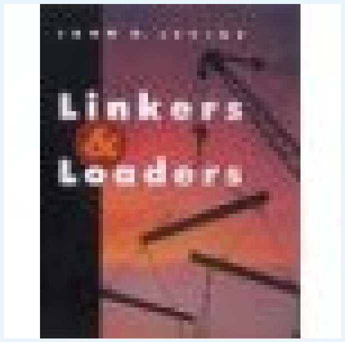
open_exec, ”The Waiter”

load_binary, ”The Nexus”

mmap/mprotect, ”The Lord”

Learning about ABI? Rant.

- **One** (!) accesible ”non-hacker” book on ABI:
  - John Levine, *”Linkers & Loaders”*



- Everything else worth reading and available is hacker sources:
  - Silvio Cesare (Phrack 56:7, etc.)
  - Phrack 61–63 (including Elfsh > ERESI)
  - ”Cheating the ELF”, the grugq
  - ”ELF virus writing HOWTO” (Bartolich)
  - Uninformed.org (”Locreate”, ...)

Lesson 2: **Composition is Weird**

Any complex execution environment is actually **many:**

One intended machine, endless **weird machines**

Exploit is ”code” that runs on a ”weird machine”, in its ”weird instructions”

Exploitation is ...

- Programming the ”weird machine” inside your machine (via crafted input)

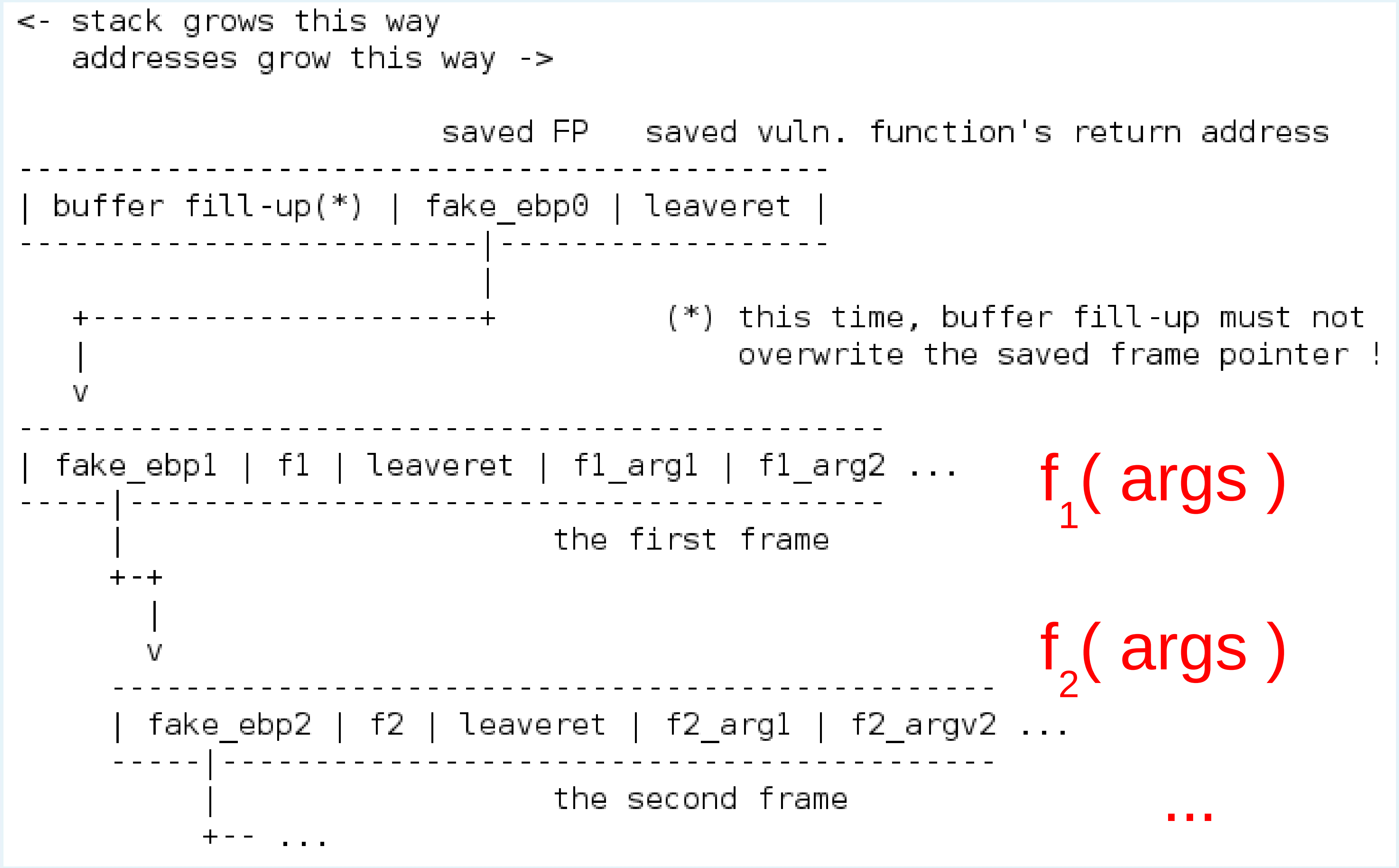
- One case study:

  from return-into-libc (1997?) to
  ”return-oriented programming” (2008)

”Malicious computation”

- All the work is done by code fragments <u>already present in the trusted code</u>!
- In 2008, academia calls this threat ”malicious <u>computation</u>” vs ”malicious <u>code</u>”
  - Hacker publications and countermeasures: 1997-- (Solar Designer, Wojtczuk, …)
  - Phrack 58 #4 (Nergal, 2001) spells it out
  - CCS 2008, it gets the cool name *”return-oriented programming”*

Phrack 58 #4 (2001)



```
<- stack grows this way
   addresses grow this way ->

                       saved FP   saved vuln. function's return address
--------------------------------------------
| buffer fill-up(*) | fake_ebp0 | leaveret |
-------------------------|------------------
                         |
   +---------------------+         (*) this time, buffer fill-up must not
   |                                   overwrite the saved frame pointer !
   v
-----------------------------------------------
| fake_ebp1 | f1 | leaveret | f1_arg1 | f1_arg2 ...
-----|-----------------------------------------
     |                       the first frame
     +-+
       |
       v
     ------------------------------------------------
     | fake_ebp2 | f2 | leaveret | f2_arg1 | f2_argv2 ...
     -----|------------------------------------------
          |                  the second frame
          +-- ...
```

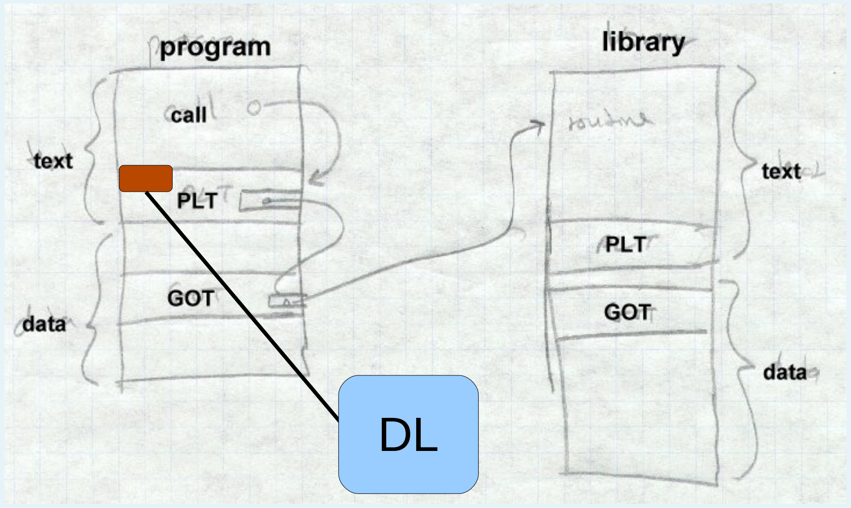
f_1(args)

f_2(args)

...

Phrack 58 #4

- Sequence stack frames (pointers & args) just so that existing code fragments are chained into programs of any length
  - Just like TCL or Forth programs
  - Pointers to functions can be provided by OS's dynamic linker itself



Another elementary instruction of the ”weird machine”, called through PLT: ”return-into-Dynamic-Linker”

Case study timeline

- Solar Designer, "Getting around non-executable stack (and fix)", 1997
- Rafal Wojtczuk, "Defeating Solar Designer non-executable stack patch", 1998
- Phrack 58:4 (Nergal), 59:5 (Durden)
- Shacham et al., 2007-2008
  - ”The geomerty of innocent flesh on the bone”, 2007
  - ”Return-Oriented Programming: Exploits Without Code Injection”, 2008
- Hund, Holz, Freiling, ”Return-oriented rootkits”, 2009
  - Actual ”compiler” to locate and assemble return-target code snippets into programs

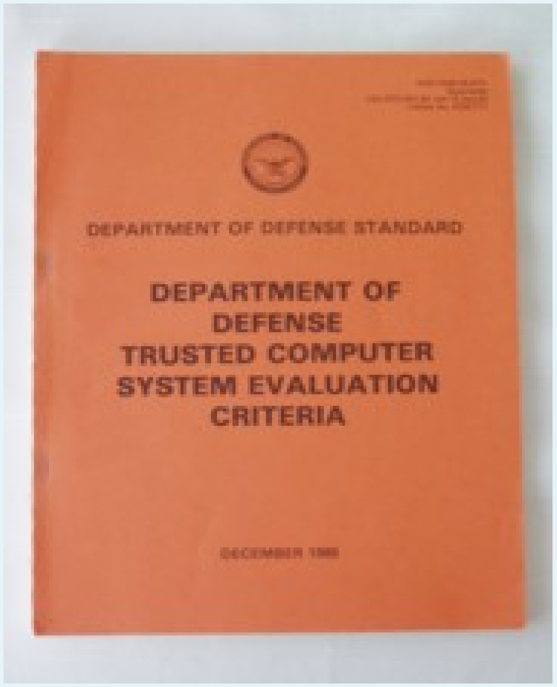
”PaX case study”
ASLR activity

So we are waiting for...

- Double-free –oriented programming? :-)
- In each case, the original code contains snippets usable as ”instructions” of a ”weird machine” that can be composed together

”OMG, it's Turing-complete!”

Lesson 3: Solid design ideas will be reborn in ”hacking”

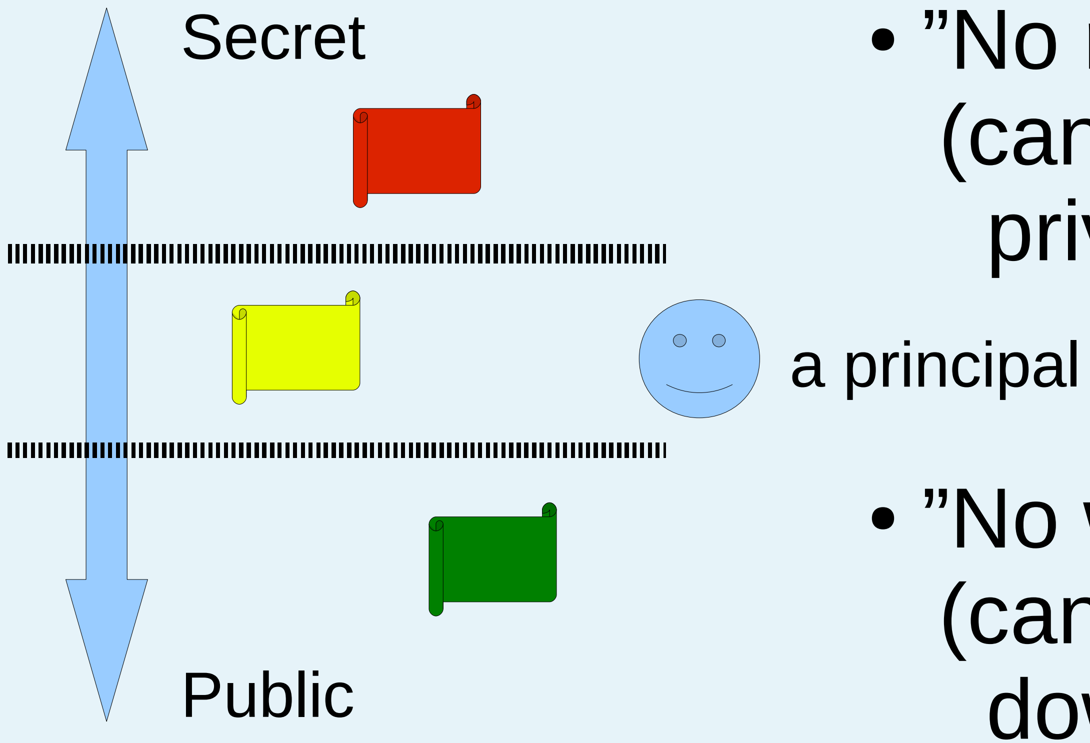

DEPARTMENT OF DEFENSE STANDARD
DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA
DECEMBER 1985

The ”Orange Book”
US DoD
”Rainbow series”

- Mandatory access control
  - Each principal is **labeled**
- All data is labeled
  - ”Everything is a file”
- Labels are checked at each operation by a ***reference monitor***
  - Most trusted part of OS, ”trusted code base”

Bell-LaPadula Formalism (1973)

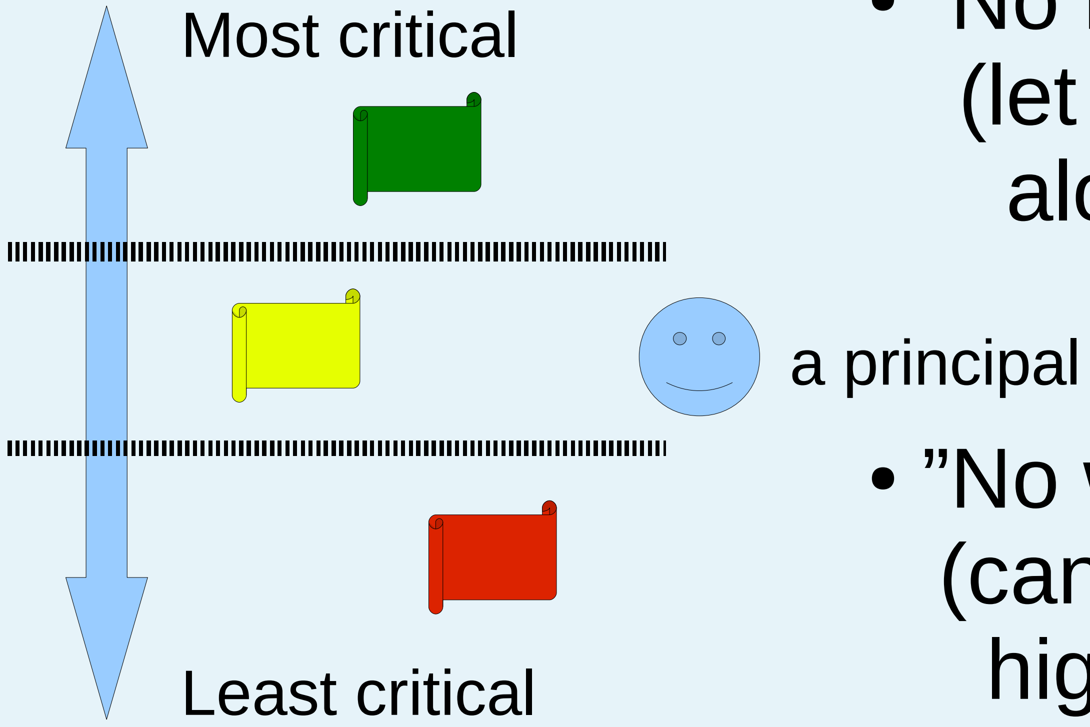
Goal: coltrol information flow, protect secrets from colluding malicious users



- "No read up" (can't read higher privs' data)
- "No write down" (can't willfully downgrade data)

Biba integrity model (1977)

Goal: prevent integrity violations by and through lower level users



- ”No read down” (let untrusted stuff alone)
- ”No write up” (can't clobber higher layers)

Once there was hardware...

- The general ”Orange Book” approach:
  - System objects get labeled according to parts they play security-wise
  - Labeling enforced by OS and/or HW

- Tagged architectures
- MMU memory segmentation

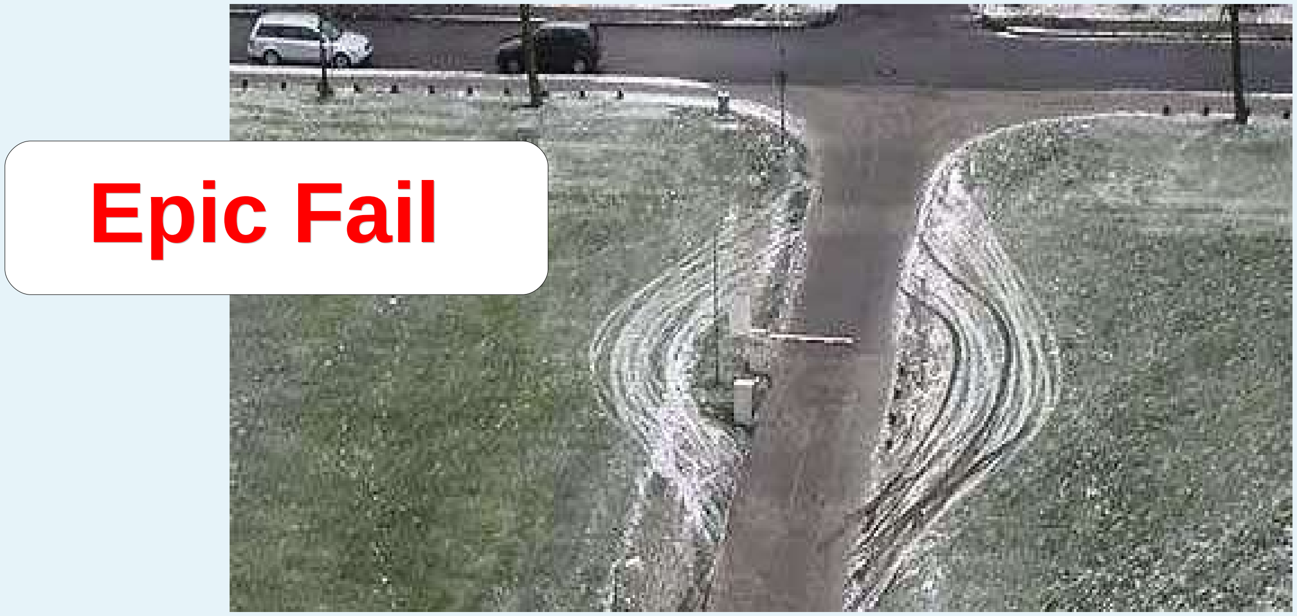
...time passes...

- The general ”Orange Book” approach:
  - System objects get labeled according to parts they play security-wise
  - Labeling enforced by OS and/or HW
- Being **executable** – ”<u>code</u>” vs ”<u>data</u>” – is a most fundamental <u>trust-wise</u> distinction between ”bunches of bytes” in RAM
  - Code runs, does stuff
  - Data kind of sits there

...epic fail...

- Being **executable** – ”code” vs ”data” – is a most fundamental trust-wise distinction between ”bunches of bytes” in RAM...

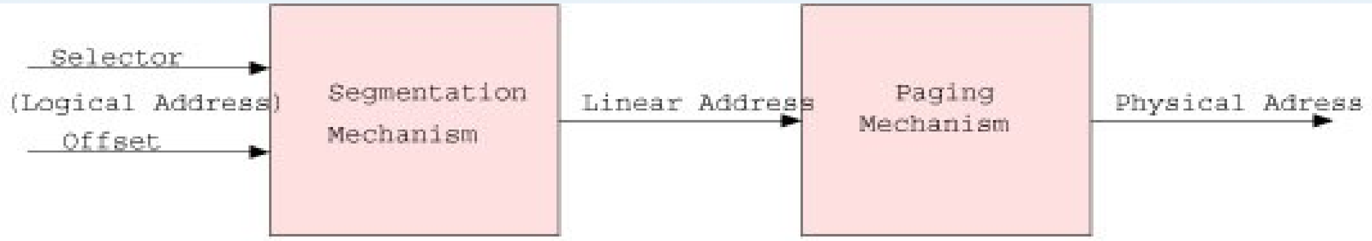
...and yet commodity systems ignored it!



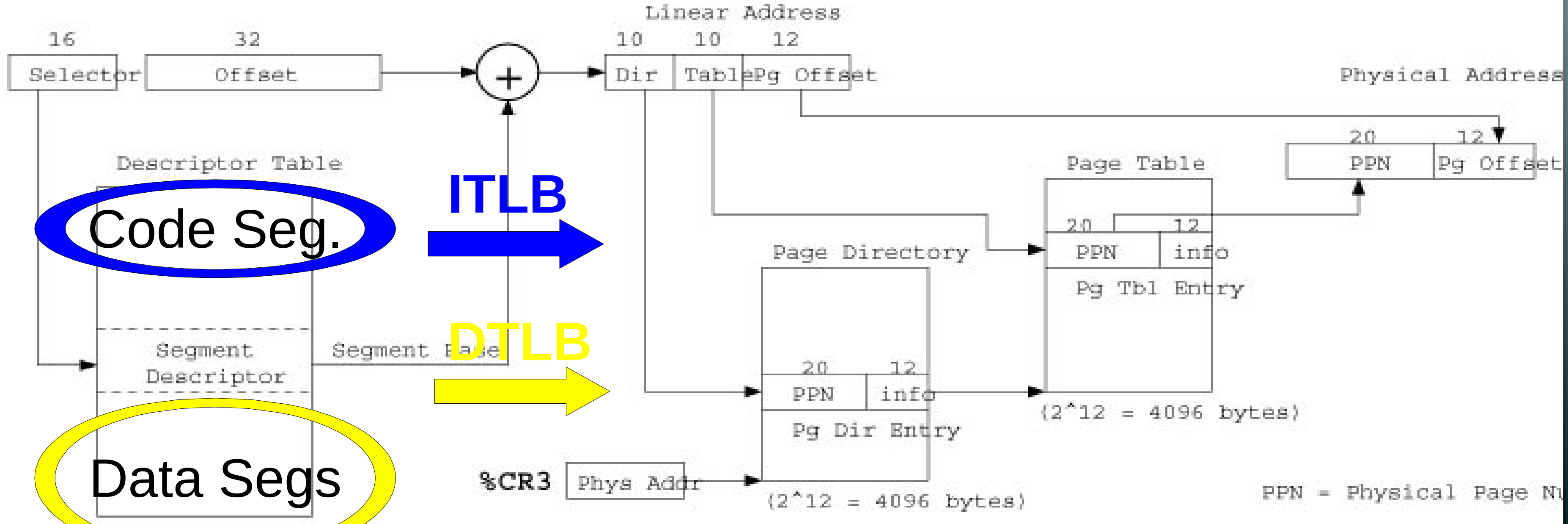
Enter hacker patches

- Label x86 pages as **non-executable**
- Emulate absent NX trapping bits to enforce
  - pageexec:
    - Overload PTE's Supervisor bit, in conjunction with split TLB
  - segmemexec:
    - Map code and data twice, via different x86 segments
    - Instruction fetches from data-only segment will trap

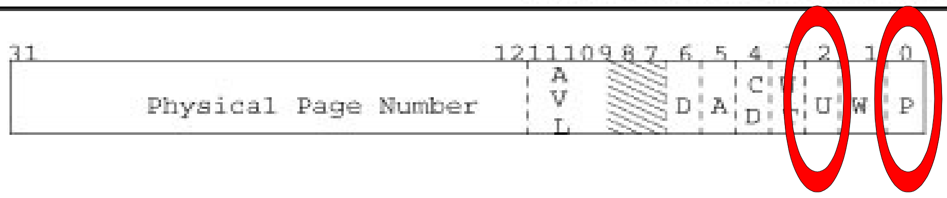
PaX



Protected-mode address translation



Detailed Address Translation



Page Table Entry

P -- Present
W -- Writable
U -- User
WT -- Write-Through (1), Write-Back (0)
CD -- Cache Disabled
A -- Accessed
D -- Dirty
AVL -- Available for system use

Page Directory Entries are identical except that bit 6 (the Dirty bit) is unused.

This is Beautiful

- ”Like Xmas for trust engineering”
- ”Hackers keep the dream alive!”

- Labels (NX) are kept <u>as close to their objects as possible</u> – right where they belong!
- Enforcement is by <u>trapping</u> – as efficient as it gets
- Page fault handler is a part of the ”<u>reference monitor</u>”

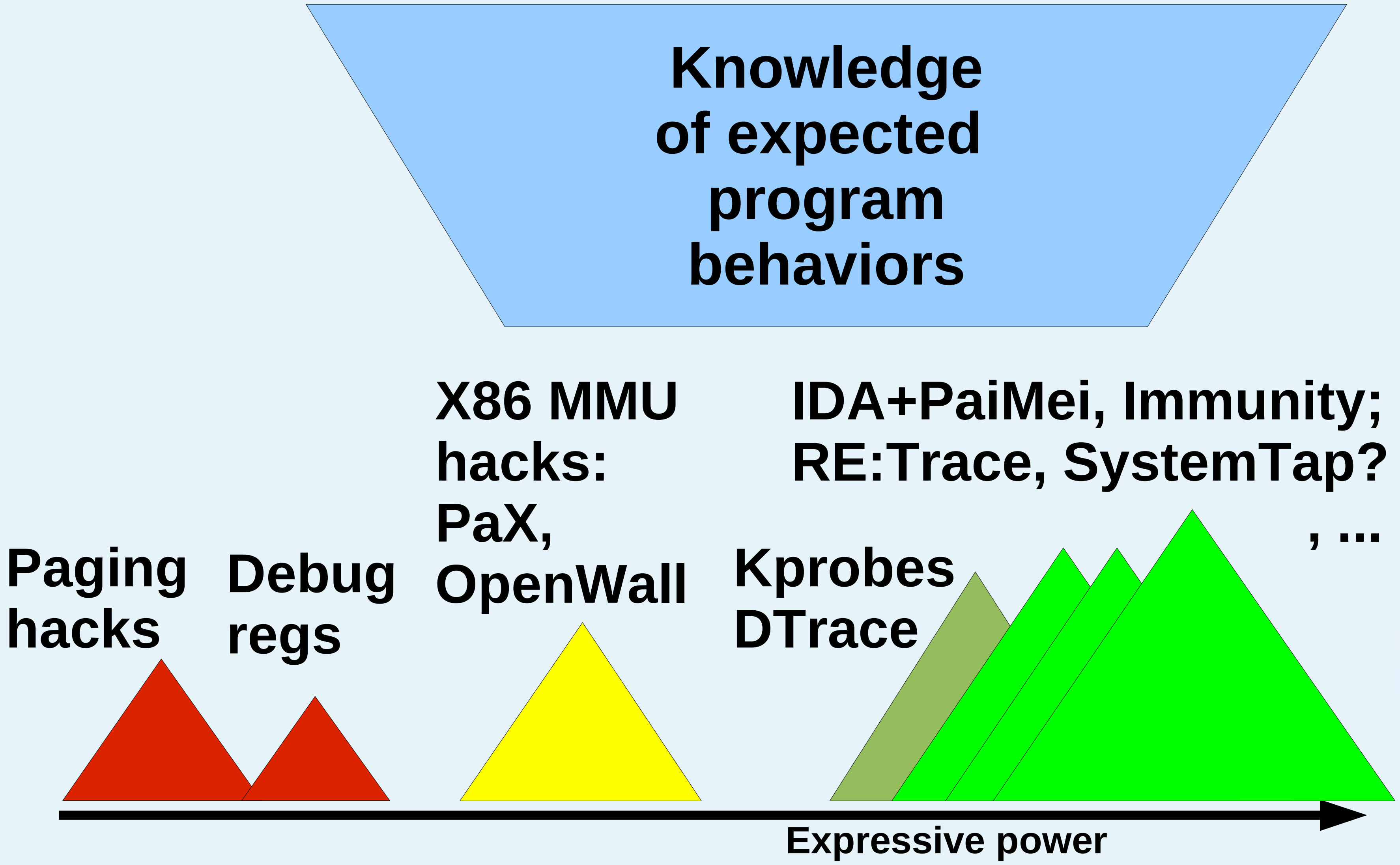
Lesson 4: Debugging ~ Trust ~ Security

- Trust is ”*relying on an entity to behave <u>as expected</u>*”
- Debugging is an activity that links <u>expected behavior</u> with <u>actual behavior</u>
- **So does security policy enforcement!**
- Hacker debuggers approach full-fledged programmable, scriptable environments

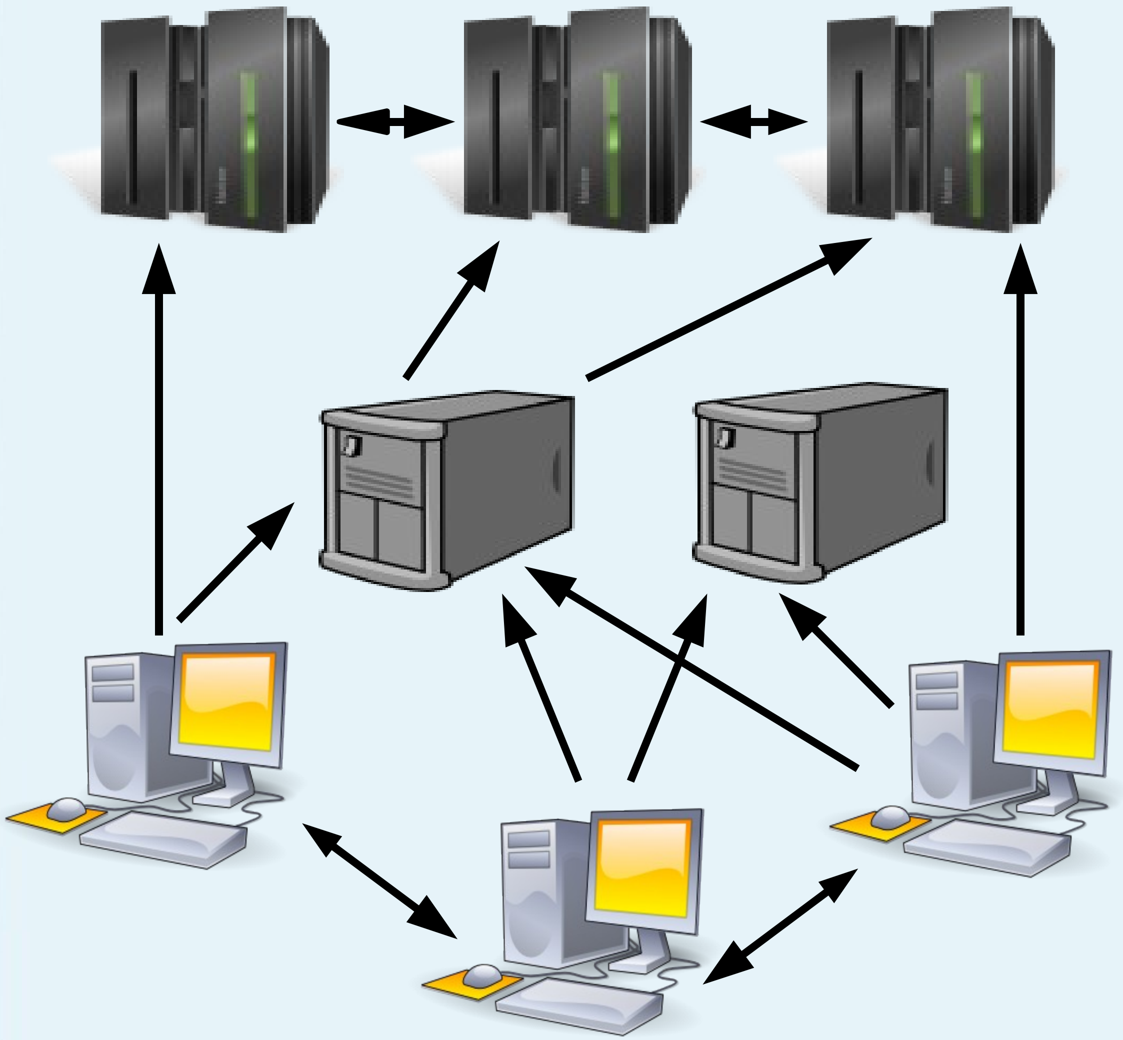
Thou shalt know how they debugger works

- Hackers are <u>leading</u> makers of debuggers
- ”Unconventional” debugging
  - Dum(b)ug
  - Rr0d Rasta debugger
  - RE:Trace, RE:Dbg
    - Uses DTrace
  - OllyBone (”special trap” case)
    - Traps on ”instr fetch from a page jsut written”

"The march of debuggers"


Knowledge of expected program behaviors
Paging hacks
Debug regs
X86 MMU hacks: PaX, OpenWall
Kprobes DTrace
IDA+PaiMei, Immunity; RE:Trace, SystemTap? , ...
Expressive power

Lesson 5: **Trust relations are first-class networking objects**



Common & well-used tools

- Get, deduce
- Check
- Describe
- Manage

"first class" kinds of objects

Follow trust relations

Trust (-relationship) mapping of networks: an industry created by hacker tools.

Thank you!

- I think I learned more about the real challenges of CS from hacker research than from any other source
- **”Hackers are a national resource”**
  Angus Blitter
- **Security does not get better until hacker tools establish a practical attack surface**
  Joshua Wright

I owe many thanks to

- *FX, who inspired me to give this talk at RSS*
- *Enno Rey and ERNW team for having me here and many discussions of trust and control in industry practice*
- *Greg Conti, who did a lot to promote the value of hacker research in academia*
- *Sean Smith, who encouraged me to write ”What hackers know that the rest of us don't” and came up with that title*