Hacking & Computer Science

Sergey Bratus
(with many contributions from others)

PKI/Trust Lab
Dartmouth College

Where I'm from

Dartmouth College, New Hampshire, USA

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

Disclaimer

(added after the talk's Q&A)

- This is not a critique of academic CS or its methods or approaches
- Rather, I argue that ”hacking” is closer in essence to the core CS topics than one might think
- For the record, a number of academic labs produce first-class hacking & some academic CS conferences finally started recognizing hacker research – but we can do better.

Realization

”How come I learned more about the nature of computers & programming from **hackers** than from **graduate school**?”

```
\cite{phrack58:9}
\cite{bugtraq-gera-2000-10-30}
...
```

My answer & more questions

"***Hacking***" is a unique and distinct **engineering/research discipline** (though not yet formally defined as such)

- How defined?
- What major human need it deals with?
- Anything worth the name is difficult – what hurdles make it hard to do?
- Why is it mathematically / theoretically hard?

What ”hacking”?

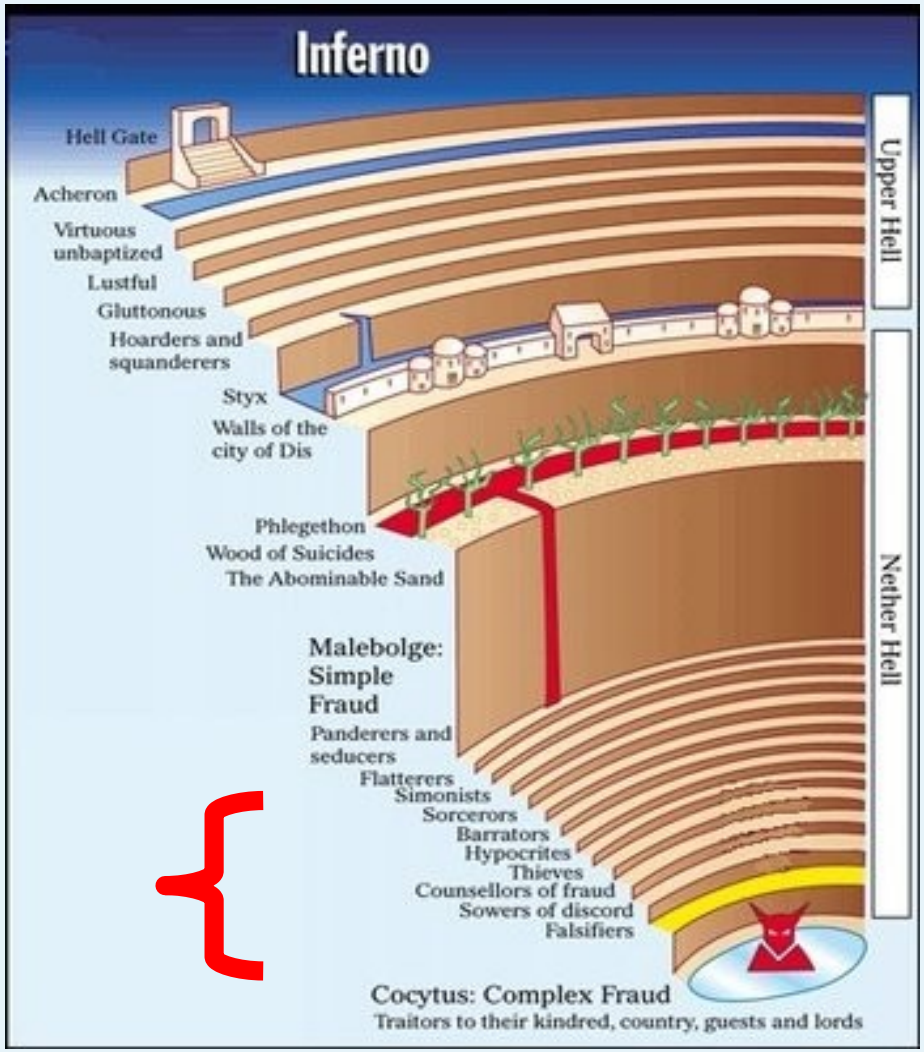
- Community perpetuates itself by its communications, just like other traditonal research/engineering communities:
- For several generations, new people join the community, learn the skills, advance & affect actual industrial security state-of-the-art
- No matter how people think of hacking, there is a reliable transmission of skills, intuitions & methods going on

Major human need: TRUST

- Humans cannot function without trust
- Trust makes us more productive
- Cultures, economies and entire ways of life are defined by levels of trust
  - ”**High Trust**” vs ”**Low Trust**” societies theory
  - Personal: born & raised in the USSR, a very low trust society

Trust is crucial to human condition

Dante's ”Inferno”: betrayers of trust placed in the 9th Circle of Hell



”Just trust our nice computers”

Hacking (n.):

the capability & skill set to expose and verify trust (security, control) assumptions expressed in software, hardware, and human-in-the-loop processes that use them

Here's hoping for 

:)

The essence of InfoSec

- FX, Bratzke @ SiS 2007:

  Pragmatically, InfoSec is about ”**working towards computer systems we can finally trust**”

- Also, cf. ”Defense is not dead” this CCC

Teaching social engineering = practical manupulation of trust

- No comprehensive **penetration** test or security **assessment** is complete without it
- But how many schools actually teach it?
- I am aware of just one such course
  - Historical hacker case studies
  - Techniques and literature review
  - Ethics and getting it past the lawyers
  - Surviving to tell the tale & the art of an executive summary

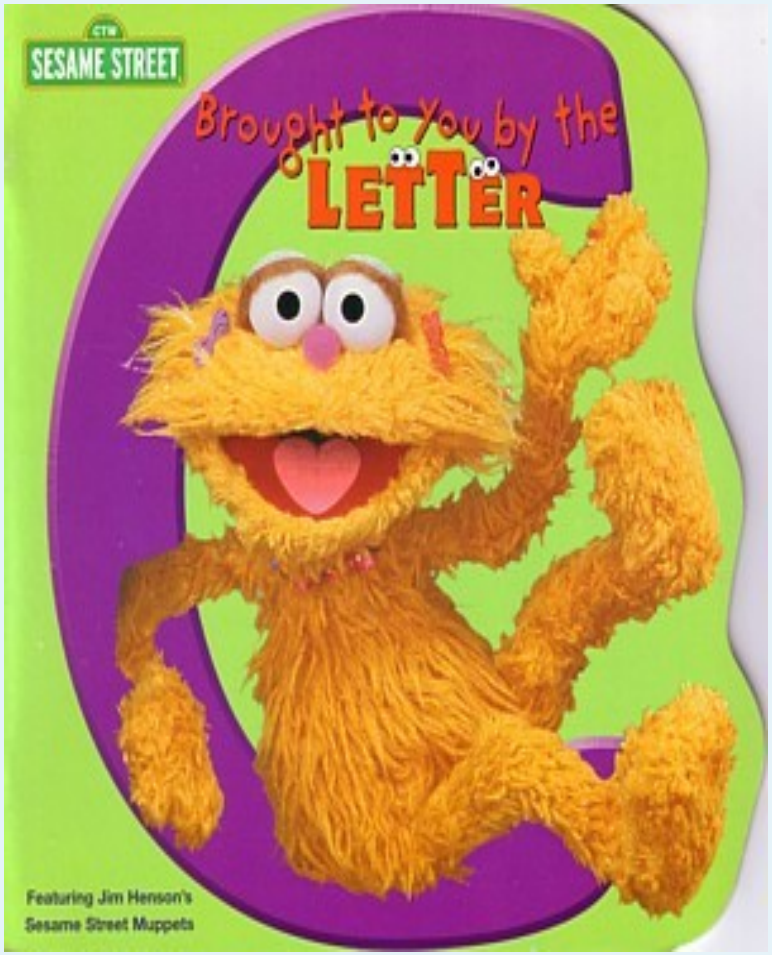
What trust in computers means

- Sociological definition of trust: the trustee **behaves as expected** (despite potential capability to violate expectations)
- Computer system behaves as expected = only **expected** kinds of **computations** occur
  - ”Uh-oh, my server process just dropped shell”

Brought to you by the letter ”C”

- **Complexity**
- **Composition**
- **Computation**

all **core** subjects of academic CS



Why building trustworthy systems is so hard?

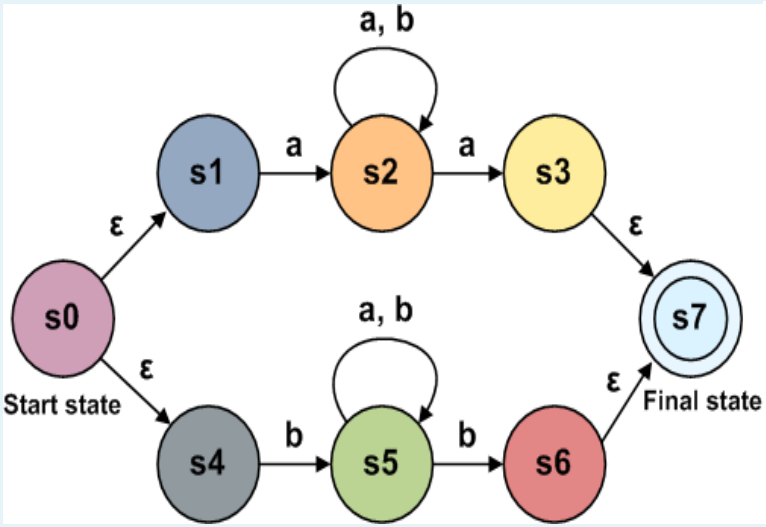
- Humans build complex systems by **composing** pre-existing pieces
- Composition of computational systems has very bad mathematical properties
  - gets **undecidable**, fast (”halting problem”)
  - stay tuned for a rigorous example :)

Security does not get better until hacker tools establish a practical attack surface
– *Joshua Wright @ Toorcon 2009*

Computation in theory

- Many kinds, hierarchically arranged by power:
  - Finite automata (~ regexps)
  - ...
  - Pushdown automata (~ recursion)
  - ...
  - Turing Machines (~ everything we think of as computable)

Computation in theory



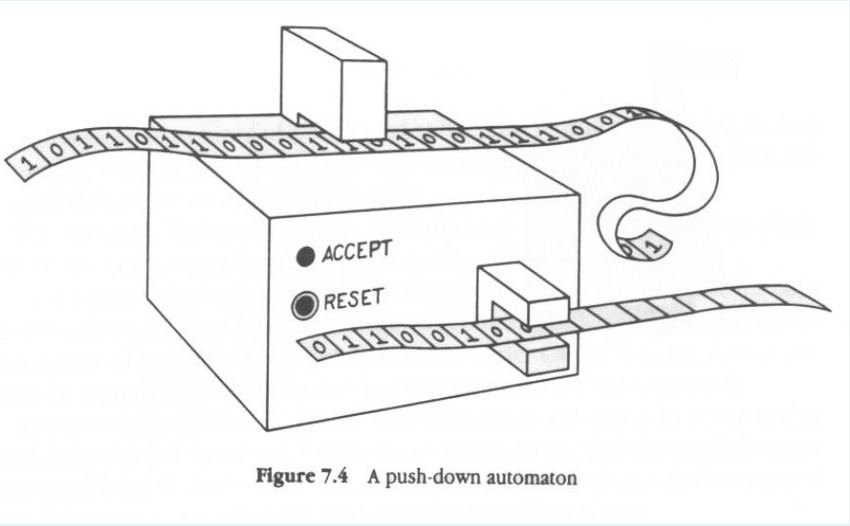
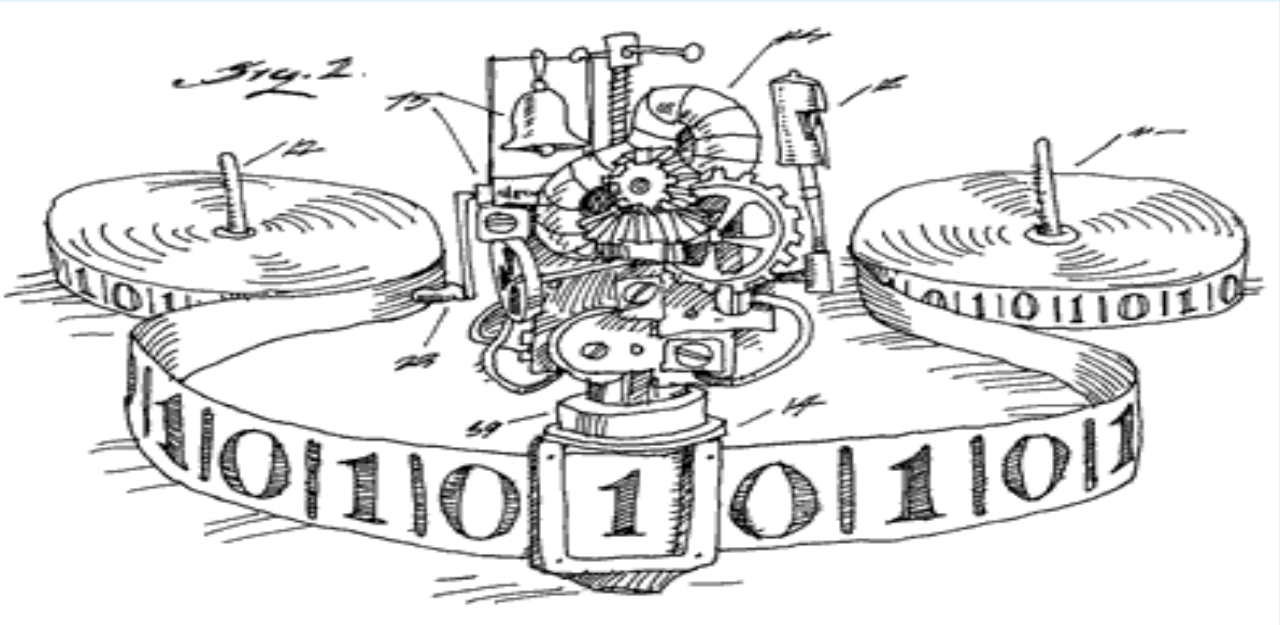
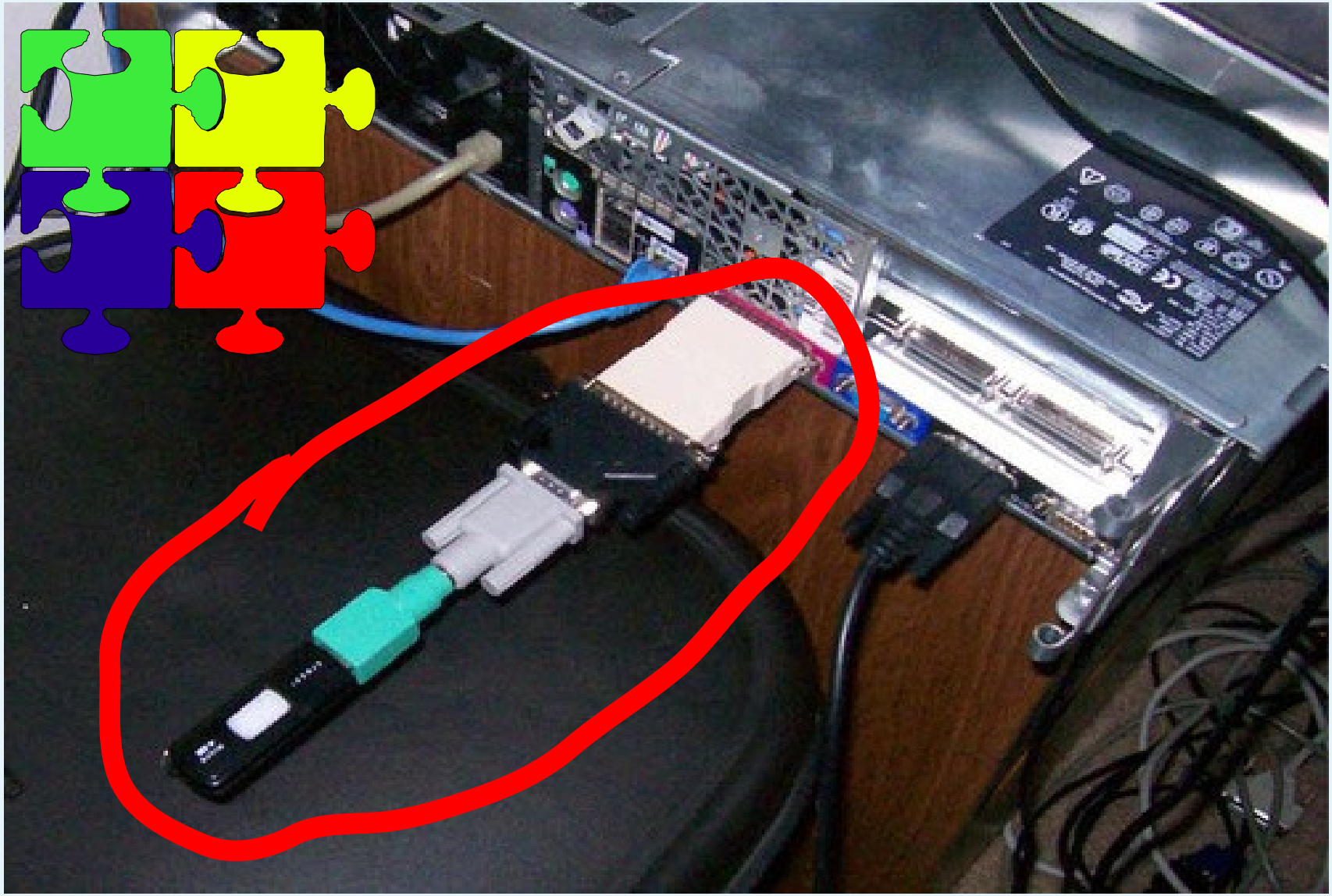


Figure 7.4 A push-down automaton



Engineering is about composition



”Composition kills”

- Compose two well-understood tools and/or processes
- Get a system with deadly properties

Computation in practice

- Real-life software and hardware quickly got too complex for theoretical **analysis** of their **behaviors**
- **Actual systems more computationally powerful than intended/expected**
- Theory moved on to theoretically tractable ”models” and ”prototypes”
  - Intractable systems are hard to publish about (!publish => perish)
  - AEG dispute (http://seclists.org/dailydave/2010/q4/)

Hacking to the rescue

- Hacker research stepped up to fill the need for **practical trust analysis** of actual **behaviors** of actual computer systems

  Trust ~ Behavior ~ Computation

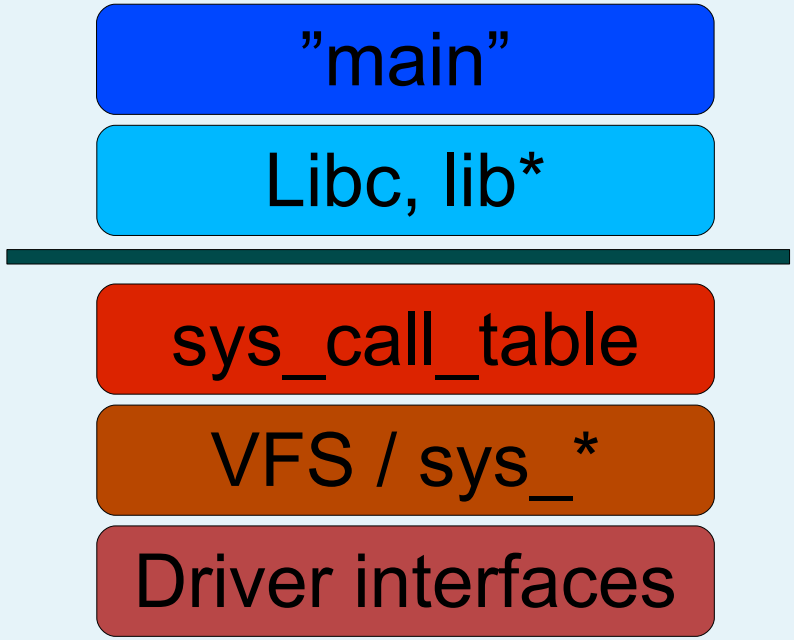
- ”What can the system **really** compute?”
- ”Can the system's human trust components be manipulated?”

”Hacker methodology”

- **Finding reliable mechanisms for unexpected computations**
- **Cross-layer** analysis of layered designs: finding the unexpected computational power
  - Systems, OSI networking, now hardware
  - Layer abstractions tend to ”leak”
- **”Weird machines”**: programming with unintended automata & Turing machines inside the target

”Cross-layer approach”

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



Layers are magical

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

Layers are magical

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

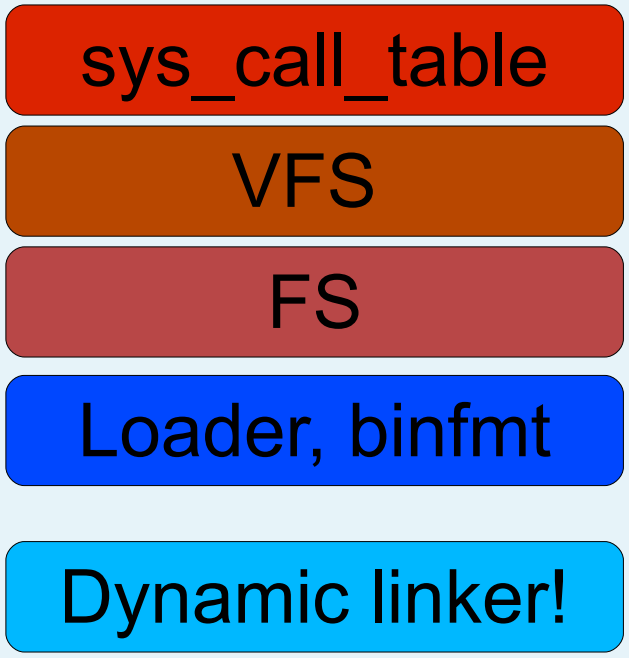
NOT! ;-)

Layers are magical

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

Best OS course reading ever :)

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



sys_execve, ”The Classic”

do_execve, ”The Obvious”

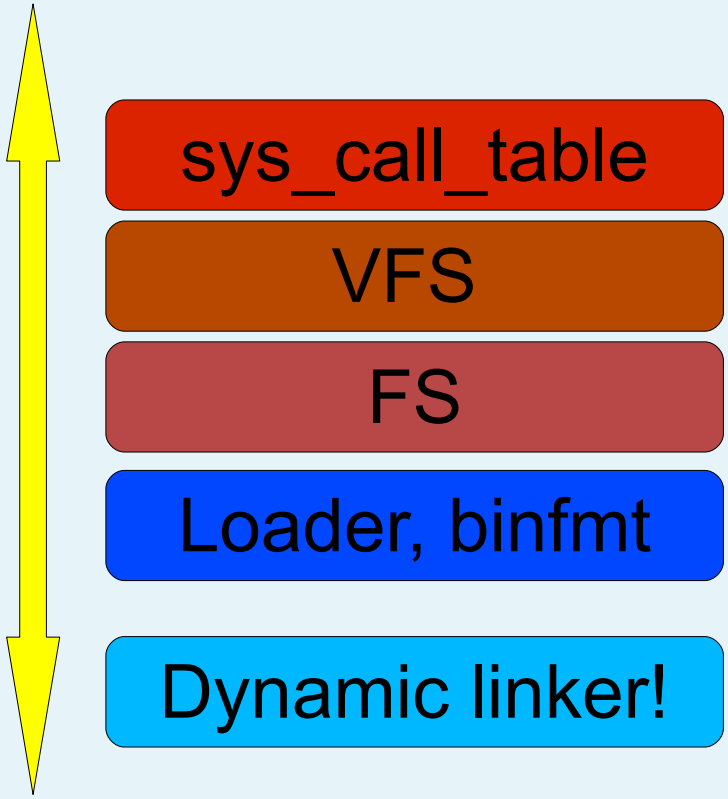
open_exec, ”The Waiter”

load_binary, ”The Nexus”

mmap/mprotect, ”The Lord”

”Cross-layer approach” in action

- *”**Deception in depth**” :*
  the main principle of rootkit engineering



sys_execve, ”The Classic”

do_execve, ”The Obvious”

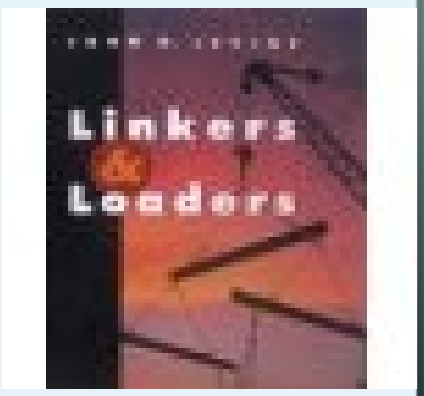
open_exec, ”The Waiter”

load_binary, ”The Nexus”

mmap/mprotect, ”The Lord”

Learning about ABI? Phrack!

- **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 (ELFSH > **ERESI**)
  - ”Cheating the ELF”, the grugq
  - ”ELF virus writing HOWTO”
  - Uninformed.org (**LOCREATE**, ...)

Weird Machines

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 a ”weird machine” inside target machine** (via crafted input)
- ”Weird assembly instructions”:
  - target's **bugs** (e.g., memory corruptions)
  - **features** (in-band signaling)
- A.k.a.: reliable implicit data & control flows
  - Hello SMT & theorem provers :)
  - Can we automatically derive **minimal descriptions** of ”weird machines”?

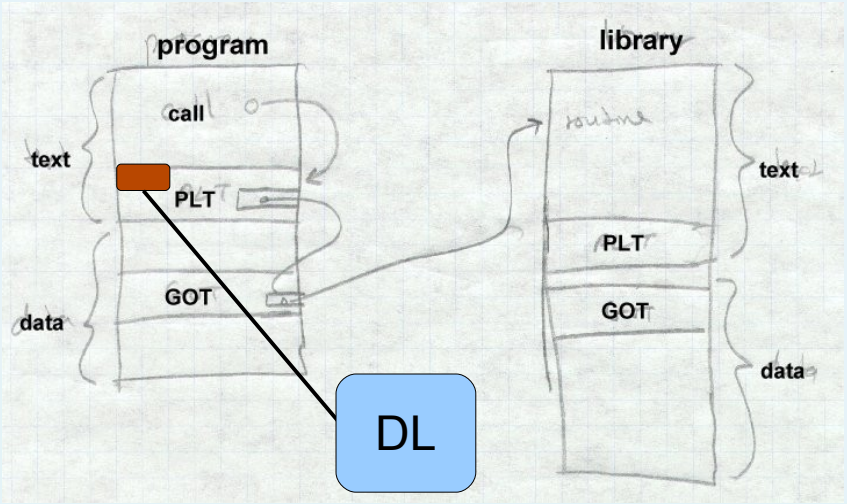
ROP timeline

- Solar Designer, "Getting around non-executable stack”, 1997
- Rafal Wojtczuk, "Defeating Solar Designer non-executable stack patch", 1998
- 2000: Tim Newsham: frame chaining
- Phrack 58:4 (Nergal), 59:5 (Durden)
- Shacham et al., 2007-2008
  - ”The geometry 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 re-target code snippets into programs

PaX non-exec, ASLR bypass

Phrack 58:4, 59:5 (Durden)

- 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-dyn-linker***”

But wait...

- Bugtraq, 2000: Gerardo Richarte (gera):

> *"I present a way to **code any program**, or almost any program, in a way such that it can be fetched into a buffer overflow in a platform where the stack (and any other place in memory, but libc) is non-executable"* – ***Oct 30, 2000***

- 2009: RoP compiler paper published by Hund, Holz & Freiling (USENIX 2009)
- 2010: Dino DaiZovi: RoP compiler (BH '10)

Memory corruption: ”creating extra computational power since 19xx”

- Haroon Meer, BlackHat 2010: ”History of memory corruption”
  - A timeline of memory corruption vulns
- Should be: ”history of **memory corruption-based programming**”
  - Memory corruption can turn an **innocent finite automaton** into a **Turing-complete environment**

Security ~ computational equivalence

- Len Sassaman, Meredith Patterson: ”**Hacking the forest with trees**” (PhNeutral, BlackHat 2010)
- Key insight: SSL security is **formally** predicated on **computational equivalence** of parsers at CA and client
- Yet verifying that two such parsers accept the same language is **undecidable!**

Composition + comp. equivalence => undecidability

- Have two parsers – or any other **data/protocol processors** – in a distributed system; require exactly matching results
- If the protocol requires more than a Non-deterministic Pushdown Automaton (~ deterministic context-free language), verifying equivalence is **undecidable**
  - Parsers for nested recursive structures ([...]) are **hard** to get equivalent => differences will abound

Other non-equivalence examples

- IDS evasion (Ptacek-Newsham, Paxson,...):

  protocol parser/stream reassembly on IDS sees a different picture than the target

- Active fingerprinting: different computation by network stacks on crafted inputs exposes targets

- VM & hypervisor ”red pills”

”OMG, it's Turing-complete!”

Data flows and security

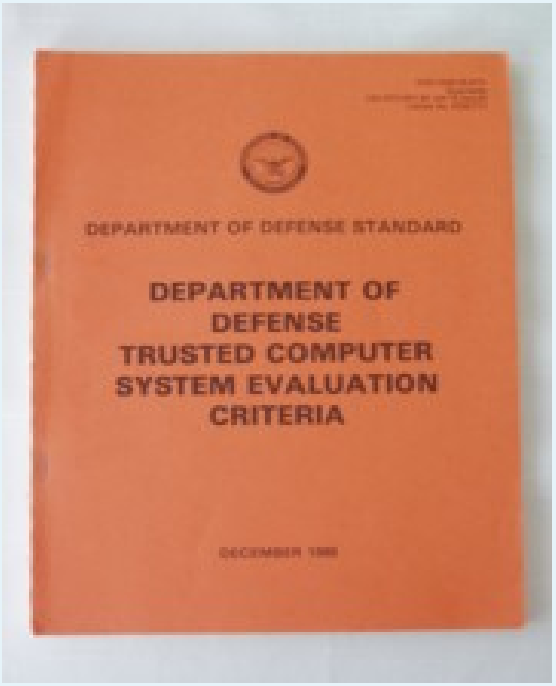
- Memory corruptions, in-band signaling turn **implicit data flows** into **control flows**
  - cf. DJB, ”Some thoughts on security after 10 years of qmail 1.0”:
  - Much more useful than ”least privilege”
- **Prove** absence of data flows (formally), generate **flawless** software (languages)

  or

  block them when they occur, with hardware (MMU) help: **tagged architectures**

The ”Orange Book” approach

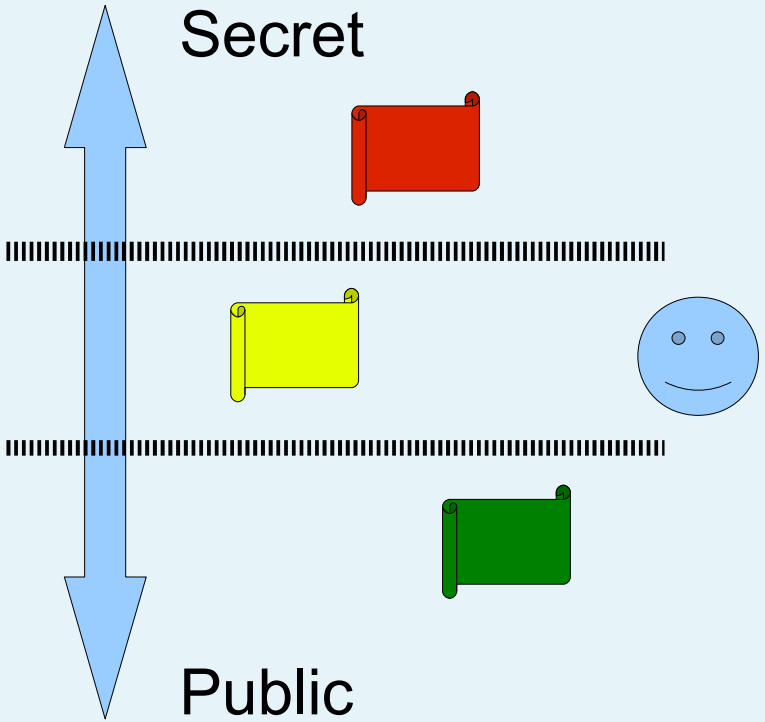
- Mandatory access control
  - Each principal is **labeled**
- All data is **labeled**
  - ”Everything is a file”
- Labels are checked at each operation by a ***reference monitor***
  - Most trusted part of OS, ”trusted code base”



The ”Orange Book”
US DoD
”Rainbow Series”

Bell-LaPadula Formalism (1973)

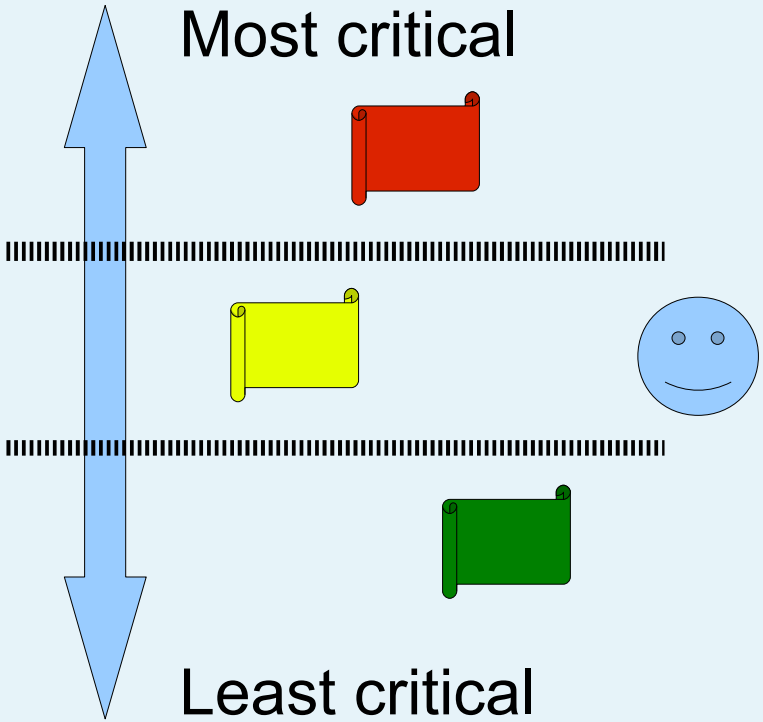
Goal: control **information flow**, protect secrets from colluding malicious users



- ”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

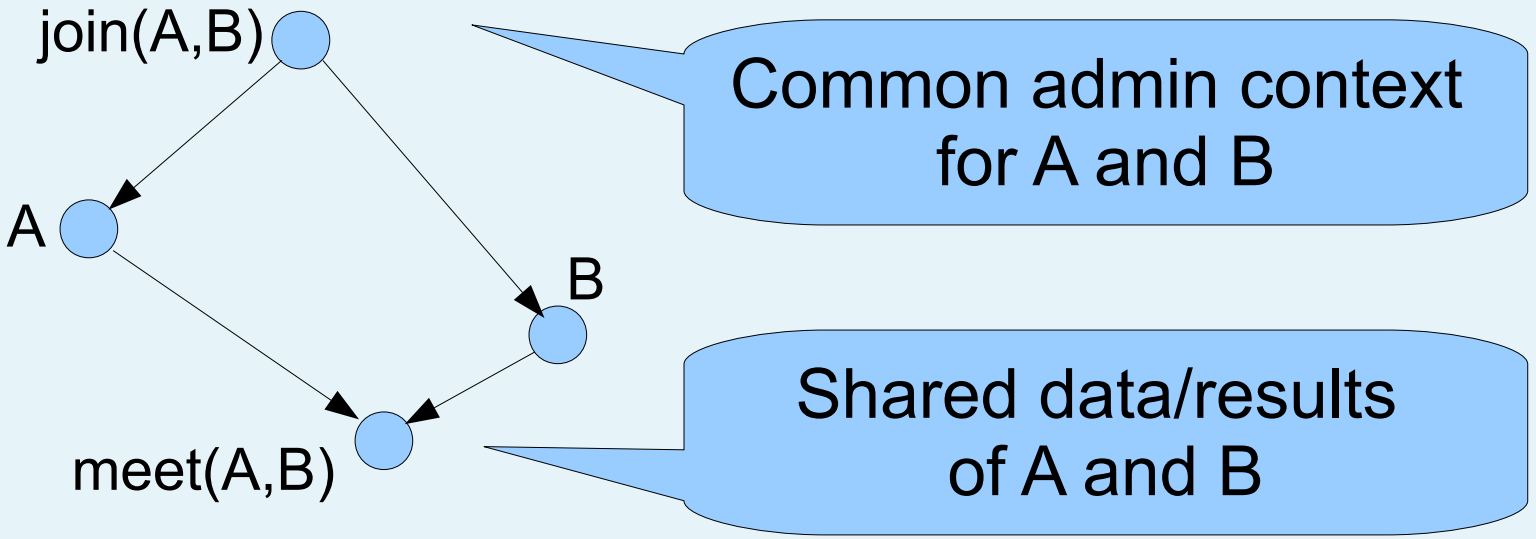


a principal

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

”It's a lattice out there!”

- Partial order on all labels
  - Some are not comparable and will not interact directly
- Every pair has a unique ”join” and ”meet”



Once there was hardware...

- The general ”Orange Book” way:
  - Memory objects labeled according to roles they play security-wise
  - Labeling enforced by OS and/or HW: illicit data flows get **trapped by MMU**

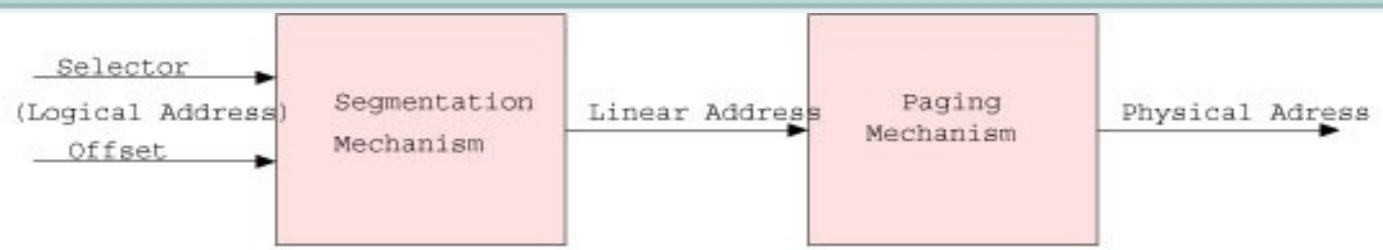
- Tagged architectures
- MMU memory segmentation

and then there was x86...

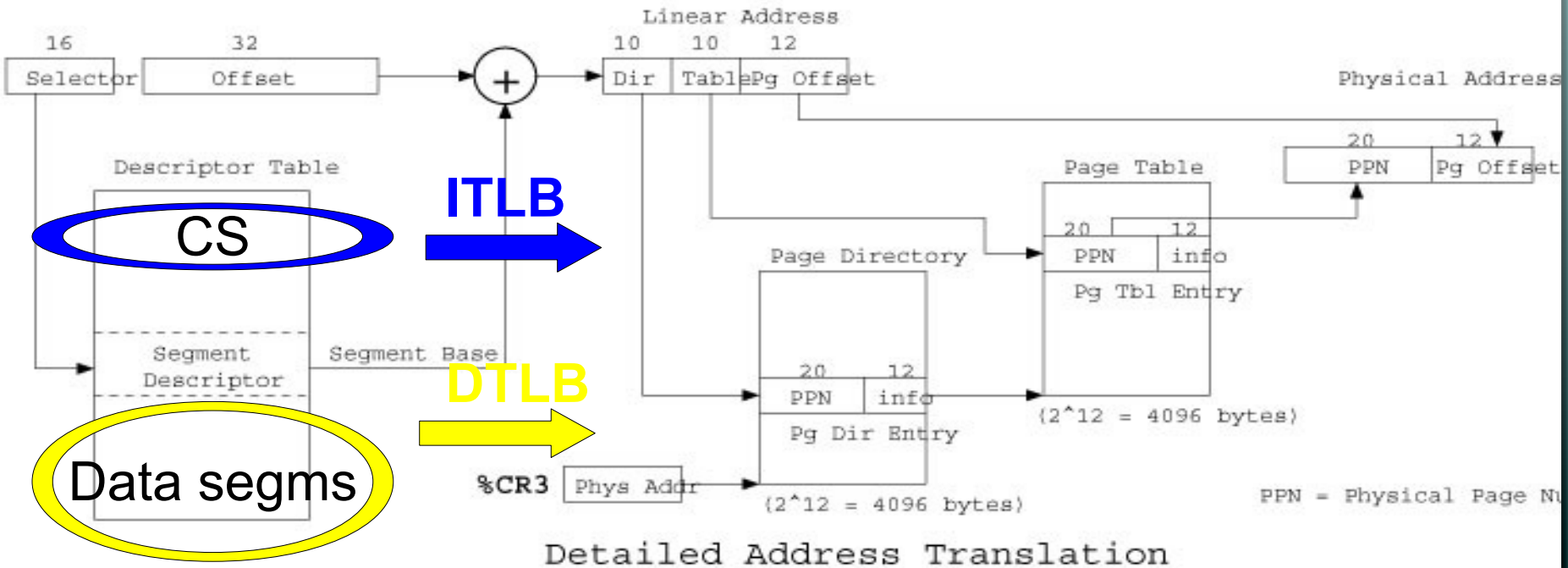
PaX and OpenWall brought tagging back on x86, for ”NX”

- Well, sort of: the tags are page-granular, and ***spread across bits in x86 segment descriptors and PTEs***

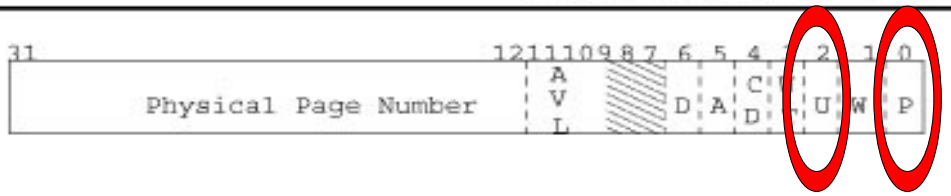
- PAGEEXEC: overload PTE's Supervisor bit, in conjunction with split TLB
- SEGMEXEC: map code and data twice, via different x86 segments (instruction fetches from data-only segment trap)



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.

Good design re-born through hacking

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

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

Thanks to

- *You for listening*
- *FX & Recurity, who listened to this first & encouraged it*
- *Len Sassaman & Meredith Patterson, who showed me the perfect, fundamental example of composition => undecidability*
- *Shelley Keating, who designed & teaches a college course in social engineering*
- *ERNW for many discussions of trust/risk*
- *Ed Feustel, who introduced me to tagged architectures*
- *Greg Conti, Michael Locasto, Anna Shubina and my other co-authors on papers related to hacking*
- *Many, many others*