What hacker research taught me

Sergey Bratus
Dartmouth College
What this is about

• A personal rant / "quest"
• The fun and huge presumption 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
”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"
    John Markoff, NYT, 2009
  - 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)
    ”Why content is not King”
"Hackers!"

- Engineers of a unique kind / not yet formally defined discipline of engineering
- "What kind of engines?"
”Hackers!”

• Engineers of a unique kind / not yet formally defined discipline of engineering

• ”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, $P \neq NP$ (?), ...
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

- 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 hard (as in "NP", RSA, other "God's own math")

Composition

- Something really human, what we must do every day

Trust
Composition is hard

- Informally: even if non-trivial properties of parts are known, the same properties of the combined system cannot be deduced by any general formal algorithm

- 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 '07: Pragmatically, InfoSec is about "working towards computer systems we can finally trust"
The discipline of hacking at a glance

Composition complexity → Hacking ↔ Trust → Everyday practice

Lofty theory
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)
Lesson 1: Look across layers

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

```

  7. Application  
  6. Presentation 
  5. Session     
  4. Transport  
  3. Network    
  2. Data link  
  1. Physical

“main”
Libc, lib*
sys_call_table
VFS / sys_*
Driver interfaces
```
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 boundaries of competence
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 boundaries of competence
- Hacker methodology in a word: cross-layer approach
Best OS course reading ever :-)  

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

- sys_call_table
- VFS
- FS
- Loader, binfmt
- Dynamic linker!
- 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

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

sys_call_table
VFS
FS
Loader, binfmt
Dynamic linker!

sys_execve, "The Classic"
do_execve, "The Obvious"
open_exec, "The Waiter"
load_binary, "The Nexus"
mmap/mprotect, "The Lord"
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)
Exploitation is ...

- Programming the "weird machine" inside your machine (via crafted input)
- In 2008, academia calls this threat "malicious computation" vs "malicious code"
  - 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

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

| buffer fill-up(*) | fake_ebp0 | leaveret |

| saved FP | saved vuln. function's return address |

| (*) this time, buffer fill-up must not overwrite the saved frame pointer |

<table>
<thead>
<tr>
<th>fake_ebp1</th>
<th>f1</th>
<th>leaveret</th>
<th>f1_arg1</th>
<th>f1_arg2</th>
</tr>
</thead>
</table>

| f1( args ) |

| the first frame |

<table>
<thead>
<tr>
<th>++</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>v</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>fake_ebp2</th>
<th>f2</th>
<th>leaveret</th>
<th>f2_arg1</th>
<th>f2_argv2</th>
</tr>
</thead>
</table>

| f2( args ) |

| the second frame |

<table>
<thead>
<tr>
<th>+-+</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>v</th>
</tr>
</thead>
</table>

| ... |
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-DL"
Case study timeline

- Solar Designer, "Getting around non-executable stack (and fix)", 1997
- Phrack 58:4 (Nergal), 59:5 (Durden)
- Shacham et al., 2007-2008
  - "The geometry of innocent flesh on the bone", 2007
  - 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? :-)
- DL-malloc –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!"
Hacking and Multi-level Security
DoD idea of Trusted Systems

- 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"
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

- "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"

![Diagram](diag.png)

- Join(A,B)
- Meet(A,B)

Common admin context for A and B

Shared data/results of A and B
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 – "code" vs "data" – is a most fundamental trust-wise 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
  • SEGMEXEC
    – Map code and data twice, via different x86 segments
    – Instruction fetches from data-only segment will trap
Protected-mode address translation

Detailed Address Translation

Page Table Entry

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 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"
Why stop at pages?

- We want to label **objects** not **pages**!
- ELF describes many objects, inter-related

- Objects have intimate & exclusive code–data relationships
What I hope to see:

- The **Return of the Lattice**, on ELF objects
- Why shouldn't the **loader** know what the **linker** knows?
- ELF **Sections** table already describes trees of datastructures (e.g., _DYNAMIC)
- We could enforce granular code–data "ownership" through the MMU trapping!
  - Like **Biba MLS for code and data units** within a process virtual address space
Learning about ABI? Rant.

• **One (!) accessible "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 grugg
  - "ELF virus writing HOWTO"
  - Uninformed.org ("Locreate", ...)
Lesson 3: Trapping is King

- Traps shape enforceable policies
- A policy must prevent reaching "untrusted states" from "trusted states"

Policy goals are expressed in terms of states. Policy checks are in terms of events/transitions. **Event system** determines policy design, mechanism & policy language.
Trapping is overloaded

- It makes **paging**-based security work
  - Page Fault handler isn't just for swapping :-)
  - PaX, OpenWall, KnoX, ...
- It makes **virtualization** work
  - Multiplexes physical devices, IO, ...
- It makes **OS-level** **isolation** work
  - ”Virtual machines, VMMs for security”
- It makes **debuggers** work
Thou shalt know how thy debugger works

- Hackers are leading 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"
Debugging ~ Trust ~ Security

- Trust is "relying on an entity to behave as expected"
- Debugging is an activity that links expected behavior with actual behavior
- So does security policy enforcement!
- Hacker debugger use is like a full-fledged programmable, scriptable environment
  - "An interpreter for C and kernel"
"The march of debuggers"

- Knowledge of expected program behaviors
- IDA+PaiMei, Immunity; RE:Trace, SystemTap?, ...
Lesson 4: Follow trust relations

Trust (-relationship) mapping of networks: 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*