Foundations of Platform Security

Nate Lawson
Computer Security Class, SJSU
May 1, 2006

Cryptography Research, Inc.
www.cryptography.com
575 Market St., 21st Floor, San Francisco, CA 94105

About myself

- Focus on network security and cryptography (13 years)
- Built various products
  - First network intrusion detection system (ISS)
  - Several storage encryption systems
  - DRM software for next-gen DVD players
- FreeBSD committer
  - ACPI and power management
  - Storage

About Cryptography Research

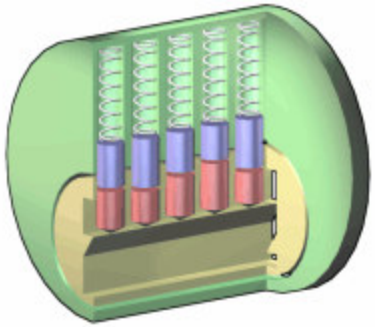
- Founded in 1995 by Paul Kocher
- Past projects
  - SSL 3.0, EFF DES cracker
- Recent and ongoing work
  - Differential power analysis (DPA)
  - Tamper resistant ASICs for pay TV
  - Content security for Blu-ray optical disc format
- We're hiring…

Overview

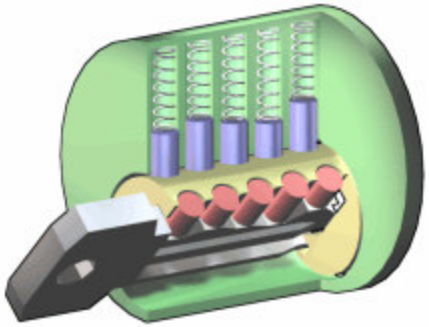
- Attributes of a good security system
  - Asymmetry: passing the security check has to be much more difficult for the attacker than the defender
  - Analogy: build up the full protection from small parts
  - Avoid complexity: each part must be very simple
- Keep these in mind when thinking about common systems you know

Lock example

- A lock is composed of these parts:
  - Plug rotates in casing, pulling the bolt from the door frame
  - Pins keep plug from rotating if not aligned
  - Key hole allows insertion of correct key to align pins



Pins prevent rotation along shear line



Key aligns pins, allowing rotation

Images courtesy of Wikipedia

Lock analysis

- Compare a good lock's design against our attributes
  - Asymmetry
    - Proper key is easy to use (insert and twist)
    - It should be hard to pick the lock
  - Analogy
    - Smallest building block is a single pin in its chamber
    - More security can be had by adding more pins
  - Avoid complexity
    - A single pin's operation is easy to test and understand
    - No multi-axis levers, gears, or other unnecessary parts
- So why do locks fail?

Lock failure scenarios

- Note that there are a lot of assumptions in the previous statements
  - Key hole is assumed to limit access to internals (Asymmetry)
    - If pins are observable and can be manipulated, lock fails
    - Fundamental limitation: key needs access to pins
  - Pins assumed independent (Analogy)
    - If pins can be individually solved, adding more pins does not increase security
    - Example: lock-picking with friction
  - Simple pins just cylindrical (Avoid complexity)
    - But need to make pin design more complex to prevent known attacks
    - Example: mushroom-shaped pins
- Anything else?

Facility security assessment

- A lock is not an island unto itself
  - Door jam may be weak
  - Window may be open
  - Wall could be broken through
  - Ceiling vent
  - Floor may be raised
  - Key can be copied or given to attacker
- What are you really trying to protect?
- All systems eventually will fail
  - How much attacker effort is expected?
  - Must know the true value of the asset to be protected and from whom

From locks to software

- Apply this analogy to a database application
- Authentication module (the "pins")
  - Verifies a password by comparing it to one stored in a file
  - Assumes file not accessible through other means, instruction execution is reliable, variables are private
  - Assumes disk storage is correct, OS has no bugs, libraries are correct
- Platform application runs on (the "key hole")
  - Standalone server: depends on network input, outbound net connections, other servers (e.g., DNS)
  - Shared virtual server: also depends on cache latency, track/sector latency on drive
  - Smart card: also depends on power, clock, data voltages and freqs, heat, radiation, physical tampering, audio monitoring

Assumptions are deadly

- Just listing all the dependencies for this database's security would take a while
- Most security professionals give up too early
  - "That overflow is not exploitable, the shellcode would have to be in Unicode"
  - "Network jitter far outweighs any timing information an attacker could recover"
- Solve by using the following rules
  - Analogy: build from small parts, making sure the parts' interfaces have no hidden assumptions
  - Avoid complexity: simplify each part until you can handle listing all its assumptions

OS security features

- These can help you avoid complexity and isolate impact of bugs
  - Least privilege principle: give each component the minimum privileges necessary to do its job
  - Defense in depth: isolate each component so that if one fails, it has the least impact on the others
- UNIX
  - chroot(): limit visible file tree per process
  - jail(): adds sockets, process visibility, shared memory, etc.
  - File permissions (r/w/x, coarse-grained). ACLs recently added
- Windows
  - RunAs: change user credentials irrevocably before running a program
  - No equivalent to chroot()
  - Extended object ACLs (r/w/x/d, fine-grained)

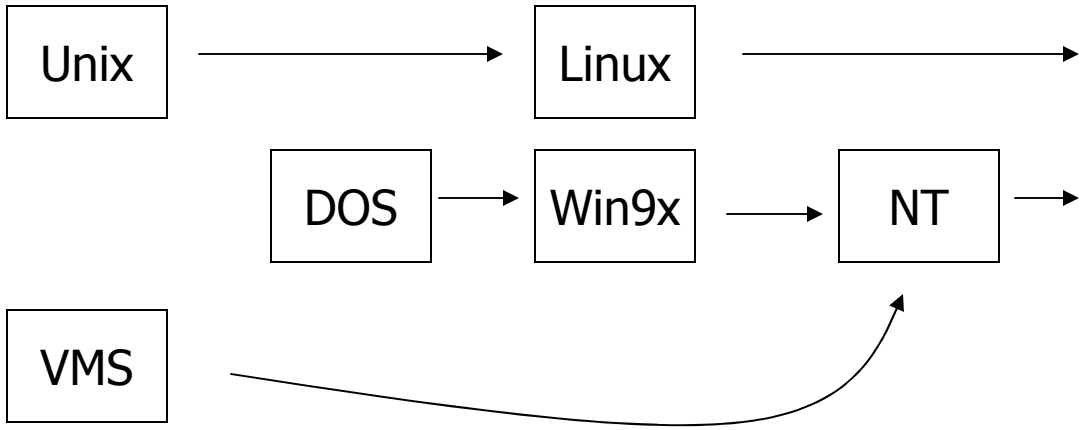
x86 hardware security features

- Page protection
  - Used to implement process separation
  - Other resources not as well partitioned (debug registers, TSC, interrupt descriptors, etc.)
- VMX instructions
  - Virtualize even privileged instructions, SMM, etc.
  - Run an unmodified OS in a partition
- Xen, VMWare
  - OS virtualization on existing x86 hardware
  - Xen: move kernel into ring 1 with some mods
  - Both moving towards VMX support

For the attacker perspective on virtualization, see "SubVirt: Implementing malware with virtual machines"; King, et al; May 2006

What's gone wrong

- Compartmentalized OS available in 1960s (Multics)
- OS changes come in waves
  - One OS replaced by another, even if a step back in some areas
  - Then as demands come in, old features are reinvented



Retro trend: compartments

- Multics invented B2-level compartments
- Unix copied Multics but more simply (chroot vs. labels)
  - Unix was simpler, cheaper, spreads widely
- DOS appeared on new hardware, no protection
  - Unix fades as expensive/high-end
- Windows added interaction and some authentication (password-based login)
  - Linux appears as a continuation of Unix (but cheaper)
- NT starts over from VMS with microkernel leanings
  - Much of this went unused as code ported from Windows
  - XP adds usable RunAs
  - Linux adds multi-level SELinux
  - Vista adds trusted path for drivers

See also: http://www.multicians.org/

Case study: sendmail

- Consider the security architecture when qmail first appeared
- Functionality
  - One program does daemon, forwarding, local delivery, aliases, lists, rewriting
  - Full configuration language
  - Many extra capabilities (identd, etc.)
- Privileges
  - Daemon runs as root, drops privileges to match the appropriate user's mailbox
  - Setuid root executable, drops privileges after initializing and selecting the requested functionality
- Platform security mechanisms
  - fork(), setuid()
- History: dozens of remote root holes, often in features you've never heard of
- Sendmail later adopted some of the least privilege approaches from qmail (e.g. separate UIDs per function)

Most recent sendmail hole(s)

- Timeout signal handler calls setjmp/longjmp which are not async-safe
- Impact: stack data can be corrupted and manipulated, leading to remote code execution
- setjmp/longjmp
  - Save current registers in context struct
  - Allow jump back to previous state based on saved context
  - Problem: stack is in indeterminate state when signal delivered
- Initial response
  - "… ISS explained it to us and told us that they had managed to craft an exploit in their lab, but frankly we don't see how it can be practical. This literally requires nanosecond precision in the millisecond world of networking." (Allman, Bugtraq mailing list)
  - Opinion misunderstands a common trait of race conditions and timing attacks:
    - Use the speed of a computer to try millions of times
    - One successful attempt is sufficient

Case study: qmail

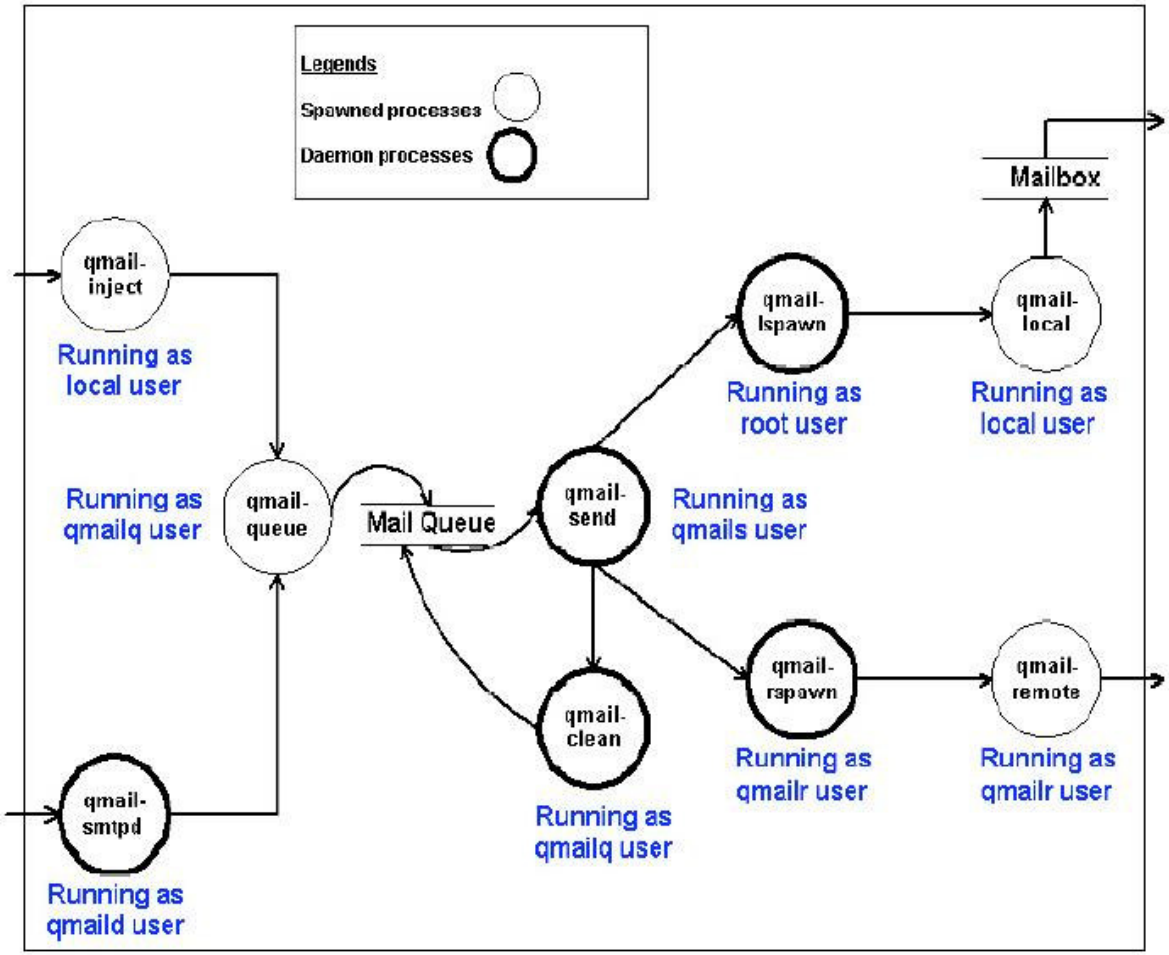


Diagram from "The Security Architecture of qmail"; Hafiz, Johnson, Afandi; 2004

Case study: qmail

- Functionality
  - Separate program for each feature
  - Separate UID/GID for each program
  - File is IPC mechanism (written to disk in some cases)
  - Custom C library (counted strings)
- Platform security mechanisms
  - fork(), setuid(), setgid()
  - chroot() used extensively
- History: 1 hole, not exploitable on any existing configuration

More info on qmail's security approach: http://cr.yp.to/qmail/guarantee.html

Most recent qmail hole (1 of 1)

- Classic integer overflow
  - Default promotion type (int) is signed but used in conjunction with unsigned int
  - Comparison operation doesn't match use of value
    - if ((u_int) -1 > 1) printf("oops!\n");
- Limitations
  - Requires 64-bit arch with large virtual address space (> 4 GB)
  - Restricted privileges once component compromised
- But, sendmail still has a larger market share
  - Does security matter?

More of what's wrong

- Firewalls: wrong solution to several problems
  - It's hard to turn off services on a variety of networked systems
  - And most of those services are insecure and enabled by default
- Anti-virus: attempt to solve the wrong problem
  - Malicious code is executing on your system
  - Other code running at the same privilege level is supposed to combat that code
  - Real problem: why is an attacker running code on your system?
- Applications: reinventing the wheel
  - "And then I'll parse the config file with strtok()…"
  - Premature optimization is the root of most bugs

Conclusion

- You’re all doomed, go home and sleep well

Conclusion (2nd try)

- Build your application carefully
- Plan to fail
  - Make sure your design keeps your first failure from being fatal
- Learn from the past
- Remember to design systems that have these attributes:
  - Asymmetry: passing the security has to be much more difficult for the attacker than the defender
  - Analogy: build up the full protection from small parts
  - Avoid complexity: each part must be very simple
- Security is an interesting field, but there's no room for ego