


Purdue University
Center for Education and Research in
Information Assurance and Security



Why Open Source Software Only Seems More Secure

<<http://www.cerias.purdue.edu>>

Eugene H. Spafford
Professor and Director



Center for Education and Research
in Information Assurance and Security

Credits to...

- Gene Kim
- Dan Farmer
- Tom Daniels
- Diego Zamboni
- Ben Kuperman
- Michele Crabb
- Alan Paller
- Assorted FIRST Teams



First off

What is secure?

- Does not disclose information
- Does not allow unauthorized access
- Does not allow unauthorized change
- Maintains QoS despite input and load
- Preserves audit, authenticity, control



Outline

- ➔ Claims
 - Some datapoints
 - Realities
 - Security Design
 - Conclusions




"Cathedral vs. Bazaar"

- Open source is inherently a better way to write secure software.
- Security flaws are more likely to be found and fixed quickly.
- More people looking at code makes it more secure.
- Collaboration makes stronger code.



"Good" Examples


- Linux
- OpenBSD, FreeBSD, NetBSD, BSDI
- GNU software
- Perl



CERIAS
Center for Education and Research
in Information Assurance and Security

"Bad" Examples

- Solaris
- HP/UX
- Windows



CERIAS
Center for Education and Research
in Information Assurance and Security

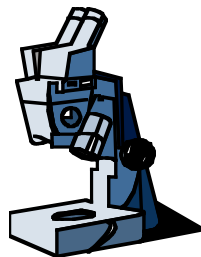
Outline

- Claims
- ➔ Some datapoints
- Realities
- Security Design
- Conclusions



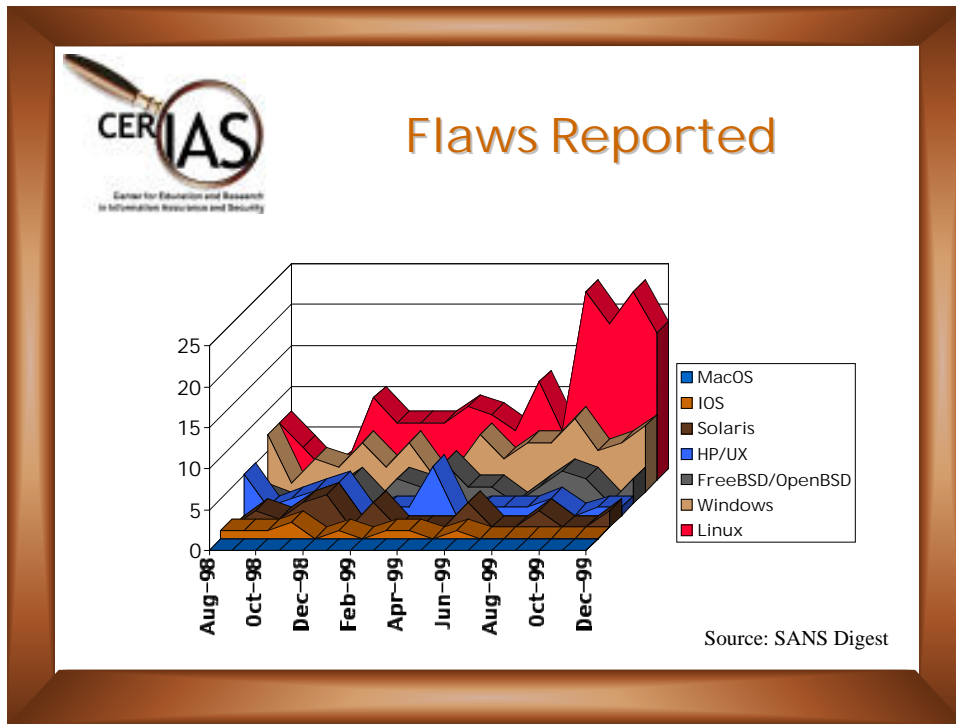
Datapoints

Let's examine some data to see if we can support all these claims.



Examples

- *BSD largely design by small group
- Linux kernel vetted through small group
- Perl largely the work of one person
- Most GNU software done by one person (each)



Documentation

Documented Commands:

- Solaris 7 1159 / 94%
- FreeBSD 3.3 700 / 96%
- Redhat 6.1 1939 / 59%

- Linux has more commands, implying greater complexity
- Competing documentation format makes this worse.

Users are more likely to make mistakes

Source: Dan Farmer



Break-ins Reported by Selected FIRST Teams

- Linux compromises dominate
 - Nearly 4 to 1 over Windows
- Commercial Unix compromises usually rare
- Windows/Unix compromises are 2 to 1
- MacOS compromises do not occur

Computer viruses are a different issue



Experience at Major Hosting Sites

- Windows
 - Too unstable
 - Can't handle load well
- Linux
 - Too "hackable"
 - Requires too many updates
- *BSD – system of choice

(Source: article at upside.com about CaveCreek)



Experience with Tripwire ASR

- Eight years of Tripwire ASR history.
- 10 contributors integrated into source code (builders), only 3 major ones.
- 50 contributors politely refused.
- Tens of thousands of users.
- Linux community had problems porting Tripwire, although relatively simple.



Experience with Kerberos 4

- Code had been in widespread release for years
- Code was used in instruction, basis for commercial products, research
- Code contained a serious cryptographic flaw discovered by S. Lodin and B. Dole, allowing near real-time access to session keys.




Software Engineering

- From a SoftEng point of view, the problems most often occur at the interfaces
 - A bigger problem in cooperative coding
- Use of dangerous languages (C, C++) does not remove problems
- No formal specification of Unix-like systems
- Testing tools are limited



Lines of Code

- There is no perfect code.
- Assume a conservative rate for serious bugs
 - 1 error per 1K LoC in unaudited code (20 pages)
 - 1 error per 5K LoC in audited code (100 pages)
- Kernels
 - OpenBSD 2.6
 - ~1874K lines, implying
 - ~375 bugs
 - Linux 2.2.121
 - ~ 1500K lines, implying
 - ~1500 bugs
 - HP/UX
 - ~2341K lines, implying
 - ~470 bugs
 - FreeBSD 3.4
 - ~ 698K lines, implying
 - ~700 bugs


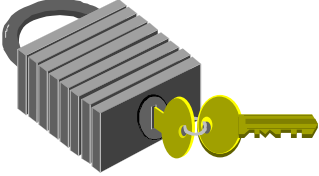


CERIAS
Center for Education and Research
in Information Assurance and Security

Secure Systems

- S/COMP
 - TCSEC/Orange Book A1 system
- Trusted VMS
- CMWS Ultrix and SunOS

...all closed source systems.



CERIAS
Center for Education and Research
in Information Assurance and Security

Outline

- Claims
- Some datapoints
- ➔ Realities
- Security Design
- Conclusions



Cathedral vs. Bazaar

- Not necessarily “closed vs. open”
- Reality: The distinguishing characteristic is control
 - Over people
 - Over design
 - Over methods
 - Most of all, over quality assurance.



Problems with the Bazaar

- You can control who comes into the cathedral.
- Productivity of coders varies:
 - Top 10% are 10x (or more) more productive than average coder.
 - Organizations invest in raising average skill level.
- Bazaar attendees include many with lower skills, especially in QA & security.



Problems with the Bazaar

- The cathedral can be built to a unified, planned blueprint.
- There may be no building code applied to the bazaar.
- Maintenance of the cathedral will not suddenly result in a new nave full of sporting goods.
- Novices have some idea about how to find their way around the cathedral.



Bazaar Facts

- Code does not magically fix itself.
- Code does not magically write itself.
- Code does not magically test itself.
- Average bazaar attendee contributes to none of the above.
 - Browsers
 - Users
 - Builders
- Builders with good skills in shorter supply than we realize




Lifecycles

- Commercial products:
identify, justify, implement, sell, sell, sell.
- Open source projects:
fire, aim, fire, aim. (Guy Kawasaki)
- Disruptive product life cycle:
ship on shoestring, but allow enough budget to improve as data comes in. (Innovator Dilemma)



Testing

- Testing is an economic consideration
- The bazaar undergoes testing based on time available, with skill and tools as are available
- The cathedral can pay for better tools, and hire experts. In the bazaar, everyone wants to produce, no one wants to sweep.



Center for Education and Research
in Information Assurance and Security

Outline

- Claims
- Some datapoints
- Realities
- ➔ Security Design
- Conclusions



Center for Education and Research
in Information Assurance and Security

Secure Design Principles

- Least privilege
- Economy of mechanism
- Complete mediation
- Open design
- Separation of privilege
- Least common mechanism
- Psychological acceptability

J. H. Saltzer & M. D. Schroeder 1975



Least privilege

- Concept is to limit:
 - Protection domains
 - Access
- No superuser
- Fine-grained ACLs, real capabilities or similar
- Role-based authentication
- Confinement



Economy of Mechanism

- Privileged code should be
 - Small
 - Simple
 - Easy to verify
 - Security built in instead of added on
- Use proven methods



Complete mediation

- Every access checked
- Security kernel approach and/or real capabilities



Open design

- Should not depend on the design being kept secret
 - Note: *Not* the same as “no security through obscurity”
- Primarily applies to cryptographic modules
- Open design is *not* the same as open source!



Separation of privilege

- Access should require more than one authorization
- Superuser is a bad concept
- Model of administrator is also the security officer violates this principle



Least common mechanism

- To reduce information flow
- To reduce race conditions
- Reuse of verified code is good, up to a point



Psychological acceptability

- Easy to use
 - Should be as easy to use as to not use
- False alarms should be avoided
- Frequent changes and updates are bad
- Should not require great expertise to get correct



Outline

- Claims
- Some datapoints
- Realities
- Security Design
- ➔ Conclusions



Revisiting the Claims

- Open source is inherently a better way to write secure software.
- Security flaws are more likely to be found and fixed quickly.
- More people looking at code makes it more secure.
- Collaboration makes stronger code.



Claim #1

- Open source is inherently a better way to write secure software.

Experience has not shown this to be true.

No current open source system is based on secure design principles.

No current open source system undergoes formal V&V testing.



Claim #2

- Security flaws are more likely to be found and fixed quickly.

True to some extent, but fixes are not always correct.

There is the question of the skill level to apply the fixes

How many flaws get out?



Claim #3

- More people looking at code makes it more secure.

Unless the people know what to look for, this is not true.

Paradoxically, more people looking may lull users into complacency.



Claim #4

- Collaboration makes stronger code.

Collaboration works only when the best collaborators are involved.

Collaboration without communication or continuity can lead to problems.

Collaboration in a controlled environment can lead to better quality over time.



Open Source More Secure

- Most secure systems ever developed were proprietary source, using formal processes and formal V&V methods.
 - Some of the poorest security has also resulted from proprietary source.
- Recent experience shows less security for many definitions of "secure"



Open Source Has Some Advantages

- Quicker response to new platform needs
- Allows experimentation
- Can make use of multiple approaches
- Allows the expert user to make changes and patches quickly



A Comment on Patches

- Fixes for flaws that require an expert to install are not a good fix.
- Fixes that break something else are not a good fix.
- Frequent fixes may be ignored.
- Exploits are not necessary to supply working fixes
- Goal should be design, not patch



Open Source Could be More Secure Than It Is

- Understand typical user
 - Changing from expert to novice
- Understand balance between features and security
- Employ better testing
- Manage complexity and change
- Build in security from the start
- Understand policy differences.



... So Could Some Closed Source Systems

- 28-30 million lines of code for an OS !?
- Consumers need to start demanding quality and security instead of new features.
- Security needs to be part of every design
- Hacking into systems is not security ("penetrate and patch" is not design)



Next Steps

- Better security education
 - Need designers and not hackers
 - Need to understand the role of policy and psychology
- Security metrics
- Inexpensive but reliable V&V testing
- Revisit some design and use goals



Thank you!