JSA Workshop: FOSS in Military Computing
Ottawa, CA — 20-22 April 2005

Security Implications of Using FOSS in Military Applications

Terry Bollinger
The MITRE Corporation

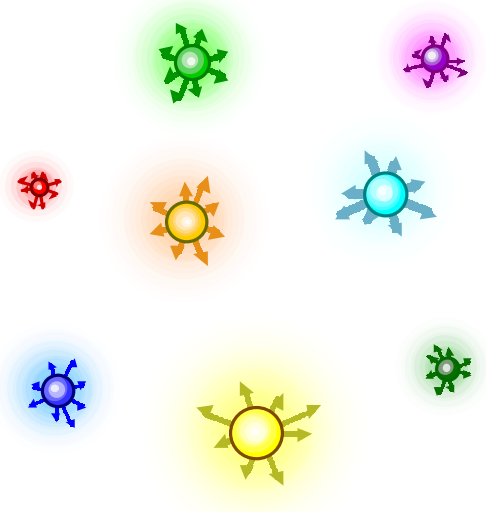
April 20, 2005

Note: *The author's affiliation with The MITRE Corporation is provided here for identification only, and does not imply MITRE concurrence with or support for the positions, opinions or viewpoints expressed by the author.*

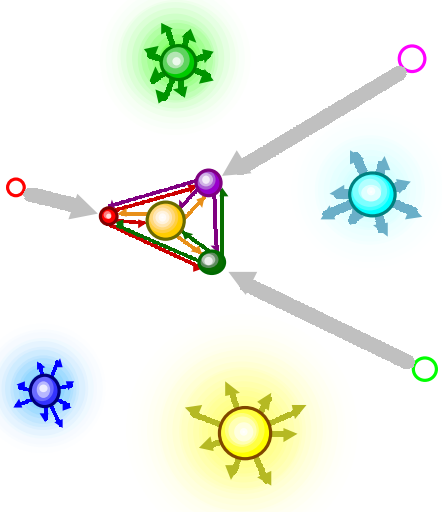
Why Does Open Source Software Exist?

1970-80s: Era of the Software Firm
(costly data transport drives structure)

Stranded Resources

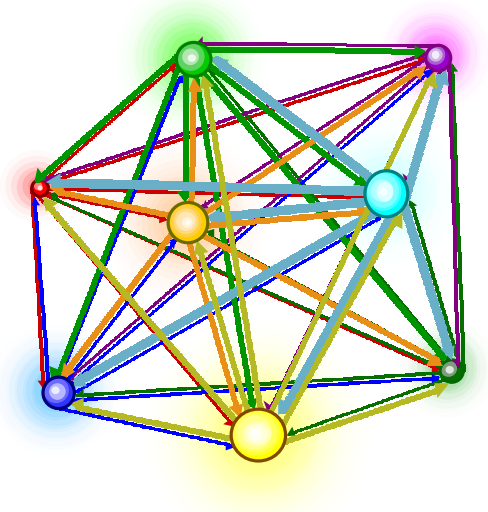


Coase Localization



RESULT: Innovation is enabled, but "invisible hand" is limited

1990s-on: Free Market
(cheap transport dominates)



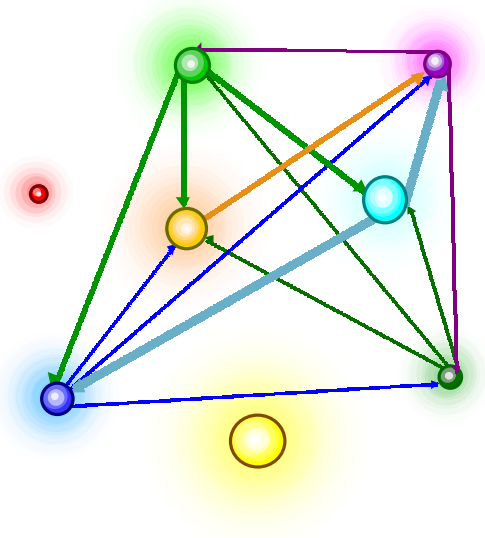
RESULT: "Invisible hand" is unleashed

Source: "Software Cooperatives" by Terry Bollinger (http://www.terrybollinger.com/)

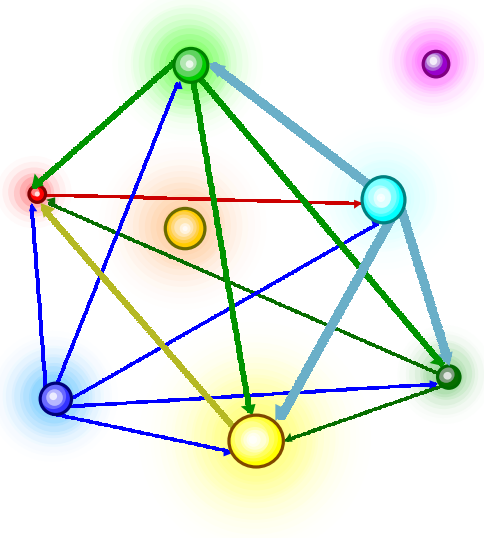
What are the Business Consequences?

FUTURE: Cooperatives (OSS, barter-based) and eventually, Consortia (fee-based) jointly dominate the market:

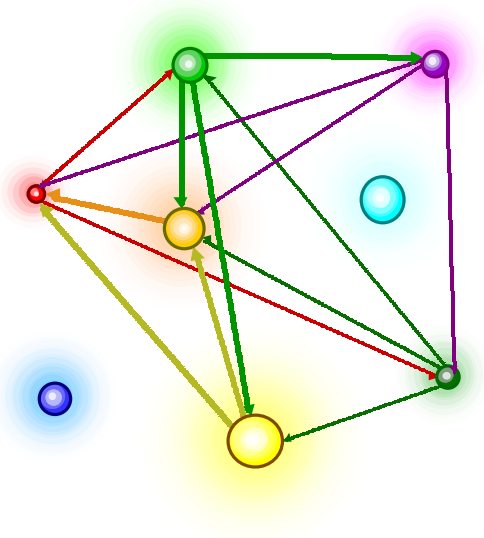
Global Project A



Global Project B



Global Project C

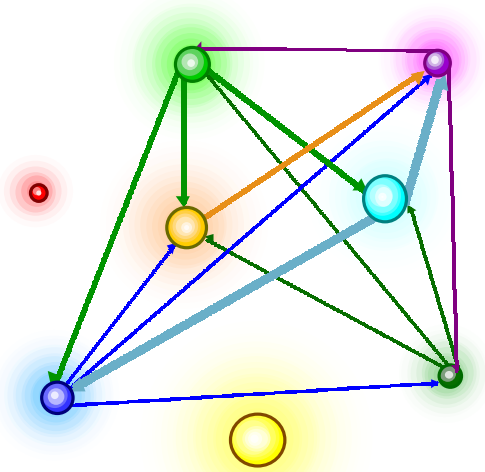


REASON: "self-selecting" groups retain free-market innovation & speed

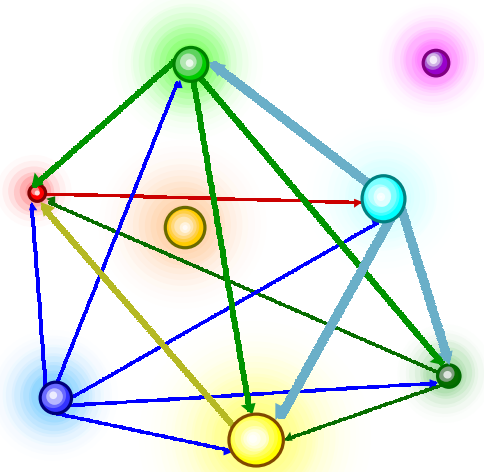
What are the Security Consequences?

- **Self-selecting groups** **with high *internal cohesion* dominate**
- **Infiltration is harder** **than for traditionally managed groups**
- **Filtering effect** **speeds innovation while slowing infiltration**

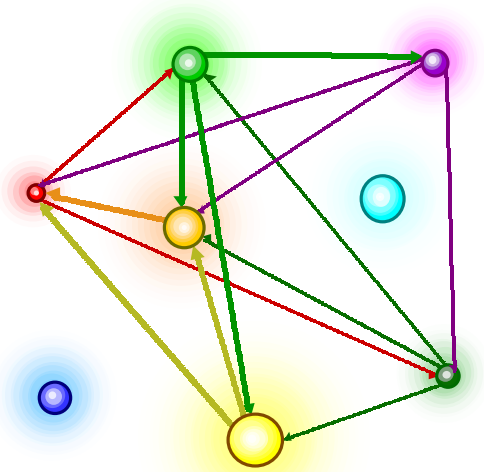
Self-Selected Group A



Self-Selected Group B



Self-Selected Group C



IMPLICATION: ***Self-selection of groups can directly benefit security***

How Does Ownership Work in Open Source?

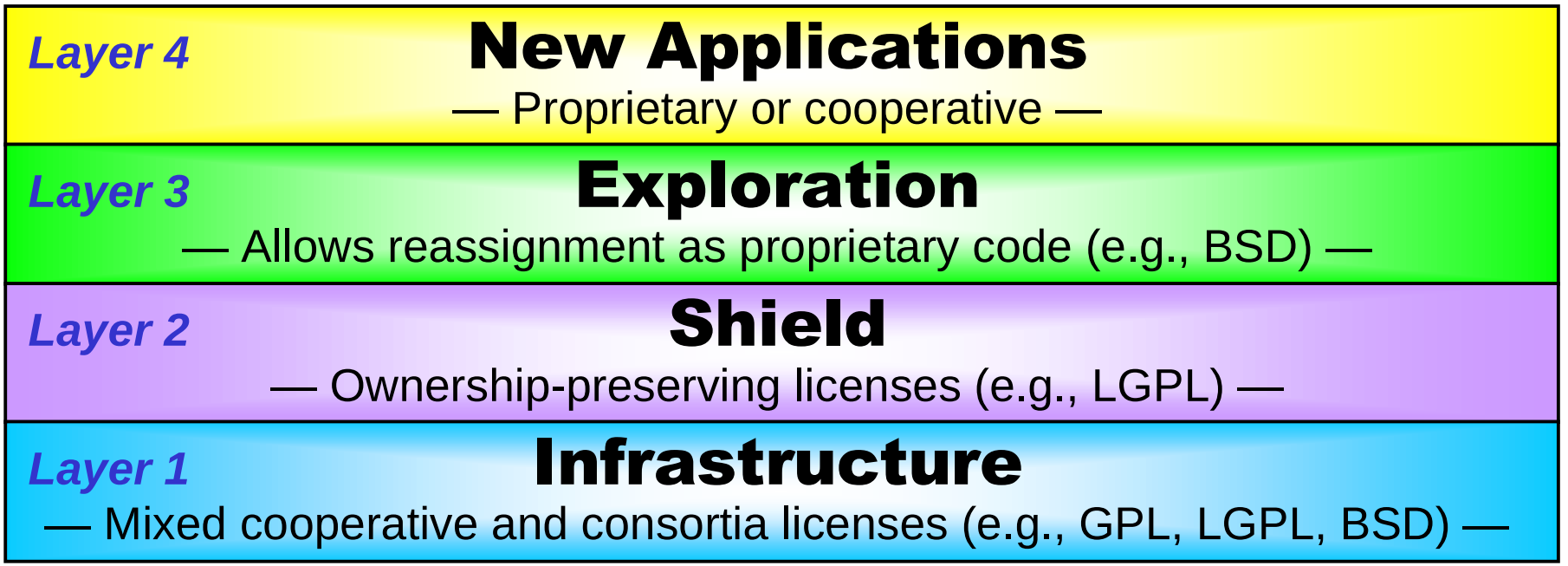
- **Schoolhouse** (e.g., GPL)
  - Jointly & voluntarily built. All may use it, but no one person or group owns it.
  - *"Once a schoolhouse, always a schoolhouse"*: Parts may be reused, but only to build more schoolhouses.
- **Public Service** (e.g., BSD, Artistic)
  - Jointly built using voluntary donations, but allows reassignment as private property (e.g., Apple OS X)
  - The most popular alternative to the GPL License
- **Liberal Lease** (e.g., LGPL)
  - Parts remain "property of the school," but can be freely reused to enhance the value of private property
  - Popular with small businesses that rely on open source

What About Traditional Software Firms?

- **The profit incentive remains intact!**
  - Consortia "flatten the playing field"…
  - … but they do *not* remove classic profit incentives
  - Ironically, companies that refuse to use consortia are the ones most likely to suffer competitively:
    - Coase-localized (traditional) software companies cannot easily compete with free-market consortia working the same problem
    - Lack of participation in global consortia limits employee abilities to understand and apply viable low-cost consortium options
- **Refocusing and restructuring is needed**
  - The maximum-value software business structure:
    - Maximize use of, and participation in, consortia
    - Discourage attempts to compete with of consortium-based software
    - Focus non-shared work and creativity primarily on difficult, unique, and high-payoff innovations

Example of a Maximum-Value Architecture

New Applications: Software that is unexpected, or solves a hard problem



Infrastructure: Software whose value increases as it is more widely shared

How Does Maximum-Value Affect Networks?

- **Assertion**

  The most economical design for a global network is to use cooperatively developed software for those parts that are the most widely shared, and proprietary software only for those parts that must remain unique.

- **Why?**
  - **Cost:** Using global communities to support globally shared components keeps support costs linear
  - **Stabilization:** Competing interests of global network users create massive resistance to arbitrary changes
  - **Security:** Distributing even trivial secrets in globally available software components dramatically increases risk of discovery. Using only cooperatively developed software helps enforce open design for all participants.

The Dark Side

- **Networking also works for the bad guys!**
- **Self-assembling groups of attackers can:**
  - Learn more rapidly when earlier ploys are uncovered
  - Explore and develop new attacks methods more quickly
  - Operate effectively on very small budgets
  - Co-opt naïve regions of the Internet for more power
  - Automate attack modes to devastate slow responders
- **The result is an ongoing arms war**
  - Groups that accept only traditional “turtle tactics” will be marginalized and become about as relevant as… turtles.
  - Groups that fully embrace the competitive advantages of using cooperative development can continue to thrive

Software Selection as Ecology

- **Angler fish…**
  - Are extremely dangerous to certain classes of fish
  - Capture those fish by dangling highly realistic baits
  - Are an unavoidable consequence of a complex ecology
  - Can *mostly* be avoided by increasing sensors & analysis
- **Malicious component creators…**
  - Are extremely dangerous to certain classes of users
  - Capture those users by dangling highly realistic baits
  - Are an unavoidable consequence of a complex ecology
  - Can *mostly* be avoided by increasing sensors & analysis
- **Are other solutions viable?**

Options for Avoiding Malicious Fish (1 of 2)

- **Option 1: Stop eating**
  - Consequences: Starvation
  - Conclusion: Not a very good solution
- **Option 2: Eat only food you have grown**
  - Consequences: Severely restricted, nutrient-poor diet
  - Expansion helps, but too much expansion recreates the original problem of uncertain origins of food supply
  - Conclusion: Possible, but costly if done right
- **Option 3: Certify your food**
  - Consequences: Good solution for *non-malicious* food
  - Problem: Malicious users can almost always circumvent
  - Conclusion: Helpful, but not a complete solution

Options for Avoiding Malicious Fish (2 of 2)

- **Option 4: Increase sensors & analysis**
  - Certification alone does not address real-time threats
  - Active sensing ("sight" and "smell") looks for subtle real-time clues to malicious intent (e.g., does that tasty fish have a long line attached to it?)
  - Active analysis (memory, alertness to subtle changes, and looking for unexpected consequences) allows leads from senses to alert users in real-time
- **Conclusion:**
  - Dropping out fails (results in "capabilities starvation")
  - Certification helps, but is *not* enough when truly malicious threats exist within the software ecosystem
  - Only improvements in real-time sensors and analysis provide a viable route for keeping up in the long term

Why Not Develop Using All Cleared People?

- **This is a variant of "grow your own"**
- **Problem:**
  - The single biggest source of security vulnerabilities in new software is unintended errors, *not* malicious acts
  - Developing in a pure classified environment can actually result in a dramatic increase in vulnerabilities due to:
    - Relative inexperience of the coders (selected for clearability, rather than for maximum expertise)
    - Far too few threads in the development "fabric" (too easy for a minor error to have major consequences, and to go unnoticed)
    - Attractiveness of such singular sites to malicious intrusion (the act of coding securely makes the site appear unusually "tasty"
- **Conclusion:**
  - Not a good idea!

An Alternative Approach to Military Software

- **An alternative approach:**
  - Use cleared people to *select, validate, and structure* components captured from the broader ecology
  - Keep new, unique coding to a minimum
  - Wherever possible, rely on strengths of the big ecology
- **Advantages:**
  - Components selections hugely increase the capabilities of the developers, without revealing their intent
  - Selection and evaluation become critical, but also can be reused (the “King’s taster” approach)
  - Small prototyping first (“cautious tasting”) lowers risk
  - Availability of source code becomes a huge advantage in this strategy, as it allows better real-time analysis

Nine Open Source Security Issues

(1) Mutual Software Trust (MST)
(2) Rapid Responses to Novel Cyber Attacks
(3) James Madison "Balance of Developers"
(4) Competitive Pressure (Riding the Wave)
(5) Practical Second-Sourcing of Software
(6) Network and Enterprise Self-Auditing
(7) Better Use of Security Research Dollars
(8) Market Survival of Security Applications
(9) Appliances: "Hardened" Open Source

(1) Mutual Software Trust (MST)

- **The problem:**
  - When groups with varying level of trust of each other must work together, how can they share infrastructure?
- **A lesson from history:**
  - The simple handshake developed first as a way of proving that neither side carried a weapon
  - For software, similar “open inspection” principles apply
- **A partial solution: Mutual Software Trust**
  - Mutual Software Trust (MST) means that all software resources shared by all parties must be fully exposed for potential inspection by any of those parties
  - Open source groups are inherently trust based, so they provide a good starting point for building MST

(2) Rapid Responses to Novel Cyber Attacks

- **The problem:**
  - Closed repair processes: ***Identify => describe => transmit => prioritize => interpret => repair => redistribute***
  - It is **difficult to accelerate** a closed repair processes
  - Each process step has a significant **risk of added error**
- **The open source response option:**
  - For critical software, develop in-house source expertise
  - Reduce repair process to: ***Identify => repair => redistribute***
- **The *potential* for rapid response exists if:**
  - The expert team is skilled at rapid response
  - The team was trained on the right source code
  - Rapid software redistribution processes also exist

(3) James Madison “Balance of Developers”

- **Question: Who controls your security?**
  - Would you trust your security to a single individual?
  - Would you trust your security to a single company?
  - Would you give up the right to question your overseers?
- **James Madison & Balance of Developers**
  - The James Madison principle of Balance of Power is based on the inevitable tendency of nearly all people to try to maximize their power over others
  - Sharing power limits abuse of power by any one group
  - In software, individual companies and programmers can suddenly wield enormous power over information, and thus over people. *(Example: Electronic-only elections)*
  - Consortia development extends the Madison principle

(4) Competitive Pressure (Riding the Wave)

- **The problem:**
  - Cooperative methods increase development speed:
    - Free-market “invisible hand” increases effective IQ of groups
    - Inherent incentives to build adaptable software reduce waste
    - Self-assembling specialty groups minimize fossilization risks
  - Pure closed-coding cannot match free-market speeds
  - The danger: *Don’t build piers while others ride waves.*
- **The solution:**
  - Keep *all* software solutions flexible and adaptable
  - Move to open standards to support rapid migration
  - Don’t fritter security on trying to perform mathematically impossible validations of huge software systems
  - Instead, concentrate closed security efforts on linchpin points of the overall distributed suite of software

(5) Practical Second-Sourcing of Software

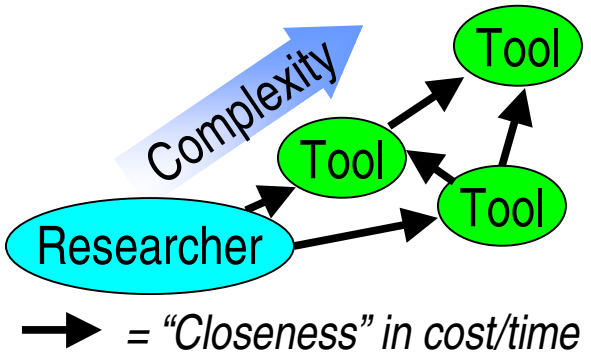
- **The problem:**
  - In hardware, second sources helps control costs & risks
  - DoD has largely abandoned second-sources in software
  - Reason: Interfaces are often closed & hard to replicate
- **Open source and adaptability**
  - Cooperative methods encourage adaptable solutions
  - Consequence: Low-cost emulation ability rises over time
  - Example: It is now estimated that 1/3 of all office users could be switched to open source *without realizing it.* (Wade Roush, *Technology Today*, Sept 2004, p. 50-56)
- **Implications for security:**
  - Provides alternatives & legitimizes legacy sole-source

(6) Network and Enterprise Self-Auditing

- **The problem:**
  - Noise-level cyber attack rates are accelerating rapidly
  - Serious cyber attacks are mutating at alarmingly speeds
  - Enterprises must respond rapidly to such changes
- **Open source and self-auditing**
  - Open source developers are strongly motivated by self-interest (*personal use* of *jointly developed* software)
  - Such self-interest translates into a keen interest in both self-testing and mutual testing of cyber security
- **Implication**
  - Open source auditing tools are important resources for identifying new examples and classes of cyber attack

(7) Better Use of Security Research Dollars

- **One of the four largest uses of open source for the DoD is research**
- **Open source in research provides:**
  - Cost-effective access to prerequisite infrastructure (e.g., Beowulf supercomputers)
  - Easy adaptation of critical components to new uses
  - A powerful way to communicate research results ("executable research papers")
  - Easier cross-training of researchers in software design
- **At a deeper level, OSS parts provide a lattice for new concept exploration:**



(8) Market Survival of Security Applications

- **The problem:**
  - Functionality-obsessed commercial markets can drive security-focused tools and languages out of the market
  - The result: Networks that lack the tools needed to create secure, highly reliable local and distributed applications
- **The solution:**
  - Cooperative development allows communities with strong interest in security and reliability to exist and even thrive, even when overall markets are functionality-obsessed. (An example: Rural electric cooperatives).
  - Self-selection of the supporting cooperatives further enhances security by creating highly cohesive groups
- **Examples:** **OpenBSD** (security), **GNAT**(Ada)

(9) Appliances: “Hardened” Open Source

- **The problem:**
  - All forms of software are… well… *soft*
  - Security tries to “harden” software through techniques such as encryption, but…
  - …encryption is ultimately more a form of hiding than securing. It’s like putting diamonds in bigger and bigger mud puddles when what you really want is a steel safe
  - Where can an enterprise find the cyber equivalent of a cost-effective, physically hardened, truly secure safe?
- **The solution:**
  - Dropping hardware costs are creating an explosion of *appliances* — specialized, physically secure “ility boxes”
  - Appliances require extreme operating system reliability
  - Guess what operating system most appliances use?

Conclusions

- **Open source software is part of security**
  - *Not* an antagonistic relationship
  - Complex and synergistic — not a simple either/or choice
- **Open source is useful for building trust**
  - Trust is a necessary component of the security equation (part of the cyberspace equivalent of the “rule of law”)
  - Building trusted infrastructure refocuses security efforts
  - Failures of trust in cyber infrastructure can have major (and negative) real-world economic consequences
- **Goal: Synergistic use of open and closed**
  - Open source helps establish trusted infrastructure
  - Closed source helps push innovation forward

For more info (& spyware help) see http://terrybollinger.com

Bollinger Literary Terms (BLT)