Security and Open Source Software

Terry Bollinger
The MITRE Corporation
November 6, 2004

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)

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/)

Copyright 2004 by Terry Bollinger. BLT 1.1 applies (see final slide).
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

**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

<table>
<thead>
<tr>
<th>Layer 4</th>
<th>New Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>— Proprietary or cooperative —</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Layer 3</th>
<th>Exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>— Allows reassignment as proprietary code (e.g., BSD) —</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Layer 2</th>
<th>Shield</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>— Ownership-preserving licenses (e.g., LGPL) —</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>— Mixed cooperative and consortia licenses (e.g., GPL, LGPL, BSD) —</td>
</tr>
</tbody>
</table>

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

---

Copyright 2004 by Terry Bollinger. BLT 1.1 applies (see final slide).
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
How Does All This Impact Network Security?

Eight open source network 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
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: \textit{Identify} => \textit{describe} => \textit{transmit} => \textit{prioritize} => \textit{interpret} => \textit{repair} => \textit{redistribute}
  - It is \textbf{difficult to accelerate} a closed repair processes
  - Each process step has a significant \textbf{risk of added error}

- **The open source response option:**
  - For critical software, develop in-house source expertise
  - Reduce repair process to: \textit{Identify} => \textit{repair} => \textit{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

Copyright 2004 by Terry Bollinger. BLT 1.1 applies (see final slide).
(5) Practical Second-Sourcing of Software

- **The problem:**
  - In hardware, second sources help 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:

---

Copyright 2004 by Terry Bollinger. BLT 1.1 applies (see final slide).
(8) Market Survival of Security Applications

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

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)
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
Bollinger Literary Terms (BLT)

Copyright 2004 by Terry Bollinger. Bollinger Literary Terms (BLT) License v1.1 applies: (1) Any reader of this work, including any incorporated entity, is granted the right to unlimited redistribution of this work in any medium or language, provided only that the meaning of the entire work and of this license remain complete and accurate in the best judgment of the redistributor. (2) Readers may at their own discretion redistribute this work either for profit or free. (3) If this work is bundled with other materials, the redistribution rights granted by the BLT license apply only to this work. (4) If brief, fair-use quotations of text or figures from this work are clearly attributed to this work by title and author name, the redistribution rights of the BLT license do not apply to them and do not need to be mentioned. (5) The original author of this work retains the right to repackaging this work and subsets of this work under non-BLT licenses, with the provision that such secondary author licenses must not attempt to remove or diminish the redistribution rights granted to readers by the original BLT version of this work. Secondary author licenses that attempt to remove such rights are invalid and cannot be enforced.