

---

# **Open Source Software (OSS/FLOSS) and Security**

## **International Workshop on Free/ Open Source Software Technologies Riyadh, Saudi Arabia**

**Dr. David A. Wheeler  
April 21, 2010**

*This presentation contains the views of the author and does not indicate endorsement by IDA, the U.S. government, or the U.S. Department of Defense.*

# Outline

---

- **Open Source Software (OSS): Definition, commercial**
- **Typical OSS development model**
- **Security: Extreme claims on OSS**
- **Unintentional vulnerabilities**
  - **Statistics on security & reliability**
  - **Open design: A security fundamental**
  - **Proprietary advantages... not necessarily**
  - **FLOSS security preconditions (unintentional)**
  - **How to evaluate OSS for security**
- **Intentional/malicious vulnerabilities**
- **Open proofs – using OSS to mature research**

# Definition: Free-Libre / Open Source Software (FLOSS/OSS)

---

- **Free-Libre / Open Source Software (FLOSS or OSS) have licenses giving users the freedom:**
  - to run the program for any purpose,
  - to study and modify the program, and
  - to freely redistribute copies of either the original or modified program (without royalties, etc.)
- ***Not* non-commercial, *not* necessarily no-charge**
  - Often supported via commercial companies
- **Synonyms: Libre software, FLOS, OSS/FS**
- **Antonyms: proprietary software, closed software**

# Why would organizations use or create OSS (value proposition)?

---

- **Can evaluate in detail, lowering risk**
  - Can see if meets needs (security, etc.)
  - Mass peer review typically greatly increases quality/security
  - Aids longevity of records (governments: aids transparency)
- **Can copy repeatedly at no additional charge (lower TCO)**
  - Support may have per-use charges (compete-able)
- **Can share development costs with other users**
- **Can modify for special needs & to counter attack**
  - Even if you're the only one who needs the modification
- ***Control own destiny*: Freedom from vendor lock-in, vendor abandonment, conflicting vendor goals, etc.**

In many cases, OSS approaches have the *potential* to increase functionality, quality, and flexibility, while lowering cost and development time

# OSS is commercial

---

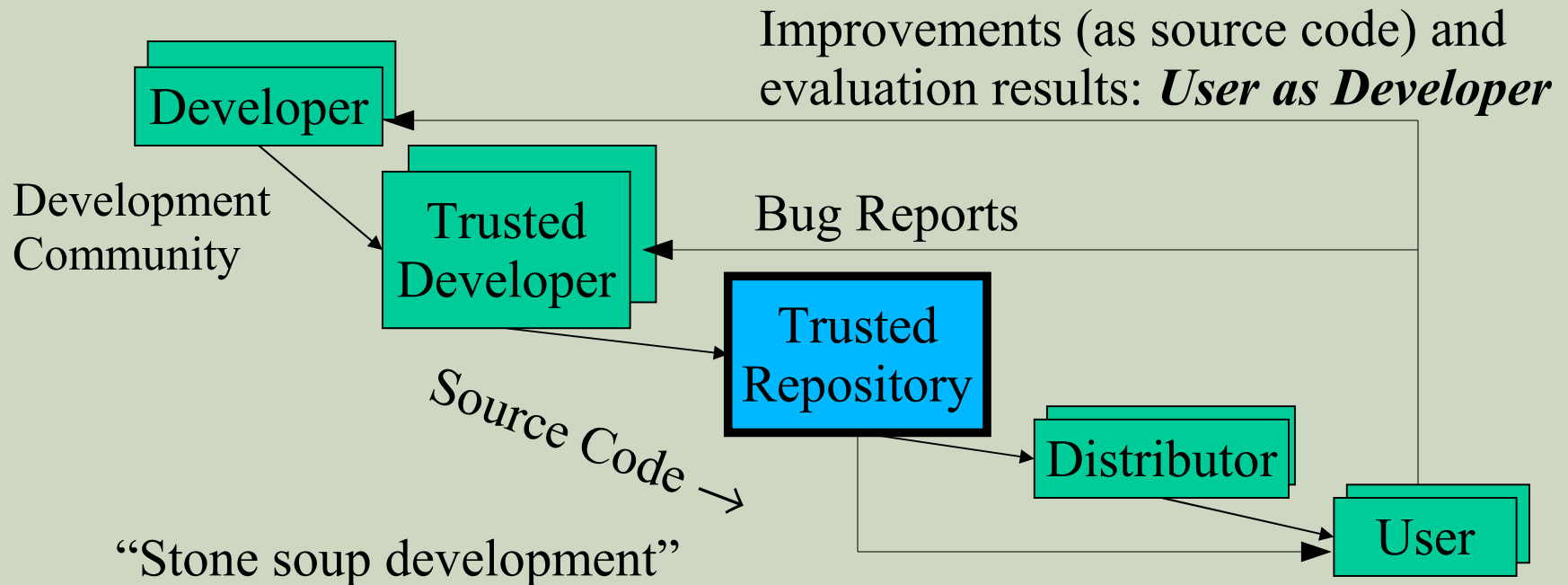
- **Many OSS projects supported by commercial companies**
  - IBM, Sun, Red Hat (solely OSS, market cap \$4.3B), Novell, Microsoft (WiX, IronPython, SFU, Codeplex site)
- **Big money in OSS companies**
  - Citrix bought XenSource (\$500 million), Sun bought MySQL (\$1 billion), Red Hat bought JBoss (\$350 million; *OSS buys OSS*)
  - IBM reports invested \$1B in 2001, made it back in 2002
  - Venture capital invested \$1.44B in OSS 2001-2006 [InfoWorld]
- **Paid developers**
  - Linux: 37K/38K changes in 2004; Apache, X Window system
- **OSS licenses/projects approve of commercial support**
- **Models: Sell service/hw, commoditize complements, avoid costs, ...**
- **Users use commercial off-the-shelf (COTS) because they share maintenance costs – OSS does!**

# OSS is commercial in United States by law and regulation

---

- **U.S. Law (41 USC 403), FAR, & DFARS: OSS is commercial!**
  - Commercial item is “(1) Any item, other than real property, that is of a type customarily used by the general public or by non-governmental entities for purposes [not government-unique], and (i) Has been sold, leased, or licensed to the general public; or (ii) Has been offered for sale, lease, or license to the general public... (3) [Above with] (i) Modifications of a type customarily available in the commercial marketplace; or (ii) Minor modifications... made to meet Federal Government requirements...”
  - Intentionally broad; "enables the Government to take greater advantage of the commercial marketplace" [DoD AT&L]
- **Dept. of the Navy “OSS Guidance” (June 5, 2007) confirms**
- **17 USC 101: OSS projects’ improvements = financial gain**
  - 17 USC 101: “financial gain” inc. “receipt, or expectation of receipt, of anything of value, including the receipt of other copyrighted works.”
- **OMB Memo M-03-14 (Commercial software): OSS support**
- **Important: U.S. Law (41 USC 403), FAR, DFARS require U.S. gov’t contracts prefer commercial items (inc. COTS) & NDI:**
  - Agencies must “(a) Conduct market research to determine [if] commercial items or nondevelopmental items are available ... (b) Acquire [them] when... available ... (c) Require prime contractors and subcontractors at all tiers to incorporate, to the maximum extent practicable, [them] as components...”

# Typical OSS development model



- OSS users typically use software without paying licensing fees
- OSS users typically pay for training & support (competed)
- OSS users are responsible for paying/developing new improvements & any evaluations that they need; often cooperate with others to do so
- Goal: Active development community (like a consortium)

**“A hand by itself cannot clap”**

---

**Working together can accomplish much more.**



# Security: Extreme claims

---

- **Extreme claims**
  - “FLOSS is always more secure”
  - “Proprietary is always more secure”
- **Reality: Neither FLOSS nor proprietary always better**
  - FLOSS does have a *potential* advantage (the “open design”) principle
  - Some *specific* FLOSS programs *are* more secure than their competitors
- **Include FLOSS options when acquiring, then evaluate**

# Some FLOSS security statistics

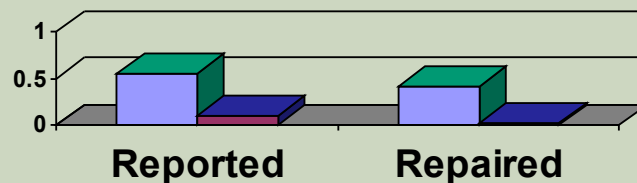
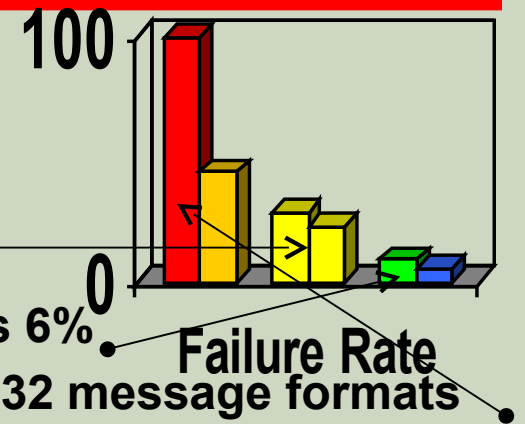
---

- **FLOSS systems scored better on security [Payne, Information Systems Journal 2002]**
- **Survey of 6,344 software development managers April 2005 favored FLOSS [BZ Research]**
- **IE 21x more likely to get spyware than Firefox [U of Wash.]**
- **Faster response: Firefox 37 days, Windows 134.5 days**
- **Browser “unsafe” days in 2004: 98% Internet Explorer, 15% Mozilla/Firefox (half of Firefox’s MacOS-only)**
- **Windows websites more vulnerable in practice**

Category	Proprietary	FLOSS
Defaced	66% (Windows)	17% (GNU/Linux)
Deployed Systems	49.6% (Windows)	29.6% (GNU/Linux)
Deployed websites (by name)	24.81% (IIS)	66.75% (Apache)

# Reliability

- Fuzz studies found FLOSS apps significantly more reliable [U Wisconsin]
  - Proprietary Unix failure rate: 28%, 23%
  - FLOSS: Slackware Linux 9%, GNU utilities 6%
  - Windows: 100%; 45% if forbid certain Win32 message formats
- IIS web servers >2x downtime of Apache [Syscontrol AG]
- Linux kernel TCP/IP had smaller defect density [Reasoning]



Proprietary Average (0.55, 0.41)

Linux kernel (0.10, 0.013)

# DoD cyber security requires OSS

---

- “One unexpected result was the degree to which Security depends on FOSS. Banning FOSS would
- remove certain types of infrastructure components (e.g., OpenBSD) that currently help support network security.
  - ... limit DoD access to—and overall expertise in—the use of powerful FOSS analysis and detection applications that hostile groups could use to help stage cyberattacks.
  - ... remove the demonstrated ability of FOSS applications to be updated rapidly in response to new types of cyberattack.

Taken together, these factors imply that banning FOSS would have immediate, broad, and strongly negative impacts on the ability of many sensitive and security-focused DoD groups to defend against cyberattacks.” - *Use of Free and Open Source Software in the US Dept. of Defense* (MITRE, sponsored by DISA), Jan. 2, 2003

“In cyberspace, coding is maneuver” - Jim Stogdill; see <http://www.slideshare.net/jstogdill/coding-is-maneuver>

# FLOSS Always More Secure?

---

- **No: Sendmail, bind 4**
- **Must examine case-by-case**
  - **But there *is* a principle that gives FLOSS a *potential* advantage...**

# Open design: A security fundamental

---

- **Saltzer & Schroeder [1974/1975] - Open design principle**
  - the protection mechanism must not depend on attacker ignorance
- **FLOSS better fulfills this principle**
- **Security experts perceive FLOSS advantage**
  - Bruce Schneier: “demand OSS for anything related to security”
  - Vincent Rijmen (AES): “forces people to write more clear code & adhere to standards”
  - Whitfield Diffie: “it’s simply unrealistic to depend on secrecy for security”

# Problems with hiding source & vulnerability secrecy

---

- **Hiding source doesn't halt attacks**
  - Presumes you can keep source secret
    - Attackers may extract or legitimately get it
  - Dynamic attacks don't need source or binary
    - Observing output from inputs sufficient for attack
  - Static attacks can use pattern-matches against binaries
  - Source can be regenerated by disassemblers & decompilers sufficiently to search for vulnerabilities
  - Secrecy inhibits helpers, while not preventing attackers
  - "Security by Obscurity" widely denigrated
- **Hiding source slows vulnerability response**
- **Vulnerability secrecy doesn't halt attacks**
  - Vulnerabilities are a time bomb and are likely to be rediscovered by attackers
  - Brief secrecy works (10-30 days), not months/years

# Can “Security by Obscurity” be a basis for security?

---

- “Security by Obscurity” can work, but iff:
  - Keeping secret actually improves security
  - You can keep the critical information a secret
- For obscurity itself to give significant security:
  - Keep source secret from all but a few people. Never sell or reveal source. E.G.: Classify
  - Keep binary secret; never sell binary to outsiders
    - Use software protection mechanisms (goo, etc.)
    - Remove software binary before exporting system
  - Do not allow inputs/outputs of program to be accessible by others – *no* Internet/web access
- Useless in most cases!
  - Incompatible with off-the-shelf model
- Proprietary software *can* be secure – but not this way <sup>16</sup>



# Proprietary advantages... not necessarily

---

- Experienced developers who understand security produce better results
  - Experience & knowledge *are critical*, but...
  - FLOSS developers often very experienced & knowledgeable too (BCG study: average 11yrs experience, 30 yrs old) – often same people
- Proprietary developers higher quality?
  - Dubious; FLOSS often higher reliability, security
  - Market rush often impairs proprietary quality
- No guarantee FLOSS is widely reviewed
  - True! Unreviewed FLOSS may be very insecure
  - Also true for proprietary (rarely reviewed!). *Check it!*
- Can sue vendor if insecure/inadequate
  - Nonsense. EULAs forbid, courts rarely accept, costly to sue with improbable results, you want sw not a suit

# FLOSS Security Preconditions (Unintentional vulnerabilities)

---

1. **Developers/reviewers need security knowledge**
  - Knowledge more important than licensing
2. **People have to actually *review* the code**
  - Reduced likelihood if niche/rarely-used, few developers, rare computer language, not really FLOSS
  - More contributors, more review
    - Is it *truly* community-developed?
  - Evidence suggests this really happens!
3. **Problems must be fixed**
  - Far better to fix *before* deployment
  - If already deployed, need to deploy fix

# Is FLOSS code ever examined?

## Yes.

---

- **Most major FLOSS projects have specific code reviews; some have rewards**
  - **Mozilla Security Bug Bounty Program (\$500)**
  - **Linux: hierarchical review, “sparse” tool**
- **Disseminated review groups (second check):**
  - **OpenBSD (for OpenBSD)**
  - **Debian-audit (for Debian Linux)**
- **Static analysis tool vendors test using FLOSS**
  - **Vulnerability Discovery and Remediation, Open Source Hardening Project (DHS/Coverity/Stanford)**
  - **Fortify’s “Java Open Review Project”**
- **Many independents (see Bugtraq, etc.)**
- **Users' increased transparency -> examination & feedback**

# Evaluating FLOSS?

## Look for evidence

---

- **First, identify your security requirements**
- **Look for evidence at FLOSS project website**
  - **User's/Admin Guides: discuss make/keep it secure?**
  - **Process for reporting security vulnerabilities?**
  - **Cryptographic signatures for current release?**
  - **Developer mailing lists discuss security issues and work to keep the program secure?**
  - **Active community**
- **Use other information sources where available**
  - **E.G., CVE... but absence is not necessarily good**
  - **External reputation (e.g., OpenBSD)**
- **See [http://www.dwheeler.com/oss\\_fs\\_eval.html](http://www.dwheeler.com/oss_fs_eval.html)**

# Inserting malicious code & FLOSS: Basic concepts

---

- **“Anyone can modify FLOSS, including attackers”**
  - **Actually, you can modify proprietary programs too... just use a hex editor. Legal niceties not protection!**
  - **Trick is to get result into user supply chain**
  - **In FLOSS, requires subverting/misleading the trusted developers or trusted repository/distribution...**
  - ***and* no one noticing the public malsource later**
- **Different threat types: Individual...nation-state**
- **Distributed source aids detection**
- **Large community-based FLOSS projects tend to have many reviewers from many countries**
  - **Makes attacks more difficult**
  - **Consider supplier as you would proprietary software**
  - **Risk larger for small FLOSS projects**

# FLOSS tends to deal with malicious code *at least* as well

---

- Linux kernel attack – repository insertion
  - Tried to hide; = instead of ==
  - Attack failed (CM, developer review, conventions)
- Debian/SourceForge repository subversions
  - Countered & restored by external copy comparisons
- Often malicious code made to look like unintentional code
  - Techniques to counter unintentional still apply
  - Attacker could devise to work around tools... but won't know in FLOSS what tools are used!
- Borland InterBase/Firebird back door: FLOSS *helped*
  - user: politically, password: correct
  - Hidden for 7 years in proprietary product
  - Found after release as FLOSS in 5 months
  - Unclear if malicious, but has its form

# Malicious attack approaches: FLOSS vs. proprietary

---

- **Repository/distribution system attack**
  - **Traditional proprietary advantage: can more easily disconnect repository from Internet, not shared between different groups**
    - **But development going global, so disconnect less practical**
  - **Proprietary advantage: distribution control**
  - **OSS advantage: Easier detection & recovery via many copies**
- **Malicious trusted developers**
  - **OSS slight advantage via review, but weak (“fix” worse!)**
  - **OSS slight advantage: More likely to know who developers are**
  - **Reality: For both, *check who is developing it!***
- **Malicious untrusted developer**
  - **Proprietary advantage: Fewer untrusted developers**
    - **Sub-suppliers, “Trusted” developers may be malicious**
  - **OSS long-term advantages: Multiple reviewers (more better)**
- **Unclear winner – No evidence proprietary always better**

# Security Preconditions (Malicious vulnerabilities)

---

- **Counter Repository/distribution system attack**
  - Widespread copies, comparison process
  - Evidence of hardened repository
  - Digitally signed distribution
- **Counter Malicious trusted developers**
  - Find out who's developing your system (*always!*)
- **Counter Malicious untrusted developer**
  - Strong review process
    - As with unintentional vulnerabilities: Security-knowledgeable developers, review, fix what's found
  - Update process, for when vulnerabilities found




# Problem: Need to speed formal methods research

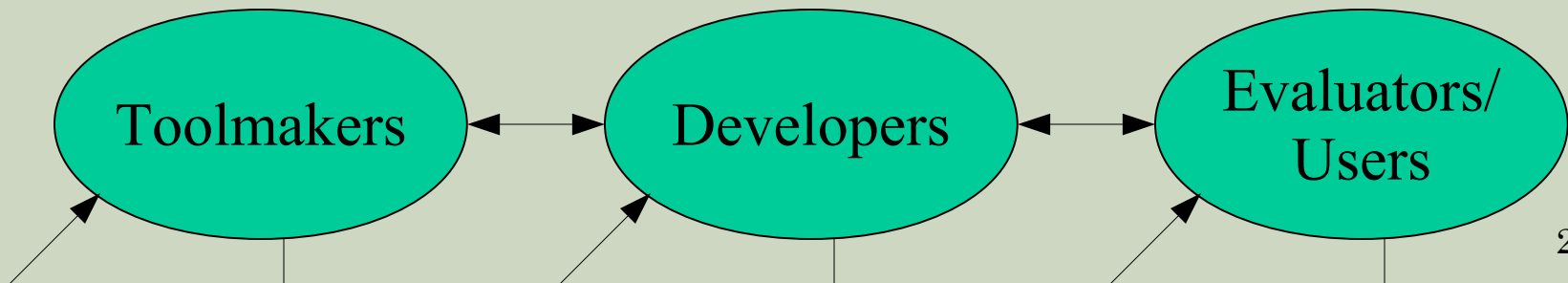
---

- **Wanted: “High assurance (HA) software”**
  - Provably always/never does something
- **Requires formal methods (FM) – progress slow**
  - Stovepiped & stalled research because FM researchers often do not share implementations (code), often forbid industry use, and almost no public examples can be reused
  - “From the publications alone, without access to the source code, various details were still unclear... what we did not realize, and which hardly could be deduced from the literature, was [an optimization] employed in GRASP and CHAFF [was critically important]... Only [when CHAFF's source code became available did] our unfortunate design decision become clear... important details are often omitted in publications and can only be extracted from source code... making source code ... available is as important to the advancement of the field as publications” [Biere, “The Evolution from LIMMAT to NANOSAT”, Apr 2004]

# Solution: Use OSS approaches to create an ecosystem

---

- **Solution: Encourage development of “Open proofs”:**
  - Source code, proofs, and required tools are OSS
- **Anyone can examine/critique, improve upon, collaborate with others for improvements**
  - Not just software, but what’s proved & tools
  - Example for training, or as useful component
- **Enables collaboration and learning**
  - By toolmakers, developers, and users
- **For more info/join, see: <http://www.openproofs.org>** 



# Bottom Line

---

- **Neither FLOSS nor proprietary always better**
  - But clearly many cases where FLOSS *is* better
- **FLOSS use increasing industry-wide**
  - In some areas, e.g., web servers, it dominates
- **Policies must not ignore or make it difficult to use FLOSS where applicable**
  - Challenges: radically different assumptions & approach
- **Include FLOSS options when acquiring, then evaluate**
  - Including development & use of existing FLOSS

# Backup slides

---

# Securing OSS

---

- **Secure OSS component's environment**
  - I.E., change what's outside that OSS component
- **Secure OSS component itself**
  - Select secure OSS component
  - Build (new) secure OSS component
  - Improve OSS component for operational environment (combine above)
- **Examples follow, but are necessarily incomplete**
  - OSS is software; all techniques for any software apply
  - *Don't need to do all of them*; merely ideas to consider
    - Rigor will depend on criticality of use
  - “OSS:” identifies OSS-unique items (read/modify/redistribute)

# Secure OSS component's environment (1 of 2)

---

- **Limit component's privileges / data rights**
  - ACLs, diff. user, SELinux privs, sandbox, jails, virtual machines
  - OSS: View to determine more precisely what privileges needed, modify so easier to limit (e.g., make finer-grain resources)
- **Limit external access to component & its data**
  - Authenticated users, encrypted connections
- **Filter input/output data (e.g., app firewalls, wrappers)**
  - Only permit validated content. OSS: More knowledge→precision
- **Subset data sent to component (can't expose/exfiltrate)**
  - Pseudonyms, DB views, etc.

# Secure OSS component's environment (2 of 2)

---

- **Add common error countermeasures**
  - Attempt to detect & counter attacks that exploit common implementation errors by observing program behavior
  - Typically embedded in OS or low-level libraries
  - Examples: StackGuard, Exec-Shield, MALLOC\_CHECK\_
  - Encourage development of these!
  - Some cause performance degradation; sometimes worth it
- **Auto patch management**
- **Backups/checkpoints/recovery**
- **Intrusion detection/prevention systems**
- **Don't forget physical & personnel security**
- **Security Controls: NIST 800-53, DODI 8500.2 (E4)**

# Select secure OSS component (1 of 2)

---

- Both OSS & proprietary: Evaluate compared to need
  - *Identify* candidates, *Read Reviews*, *Compare* (briefly) to needs through criteria, *Analyze* top candidates
- Basic evaluation criteria same, though data sources differ
  - Functionality, total cost of ownership, support, maintenance/longevity, reliability, performance, scalability, flexibility, legal/license (inc. *rights and responsibilities* – *OSS always gives right to view/ modify/ redistribute*), market share, other
  - Most OSS developed publicly – more development info available
- Examine evidence of security review
  - Often more users→more contributors→more review
    - Community-developed with many different organizations?
  - Project-specific: Mozilla bounty, etc. Check mailing list!
  - Review projects: OpenBSD, Debian Security Audit, ...
  - Tool vendors: Coverity (with DHS), Fortify, etc.
  - Common Criteria/FIPS evaluation
  - What's project's reputation? Why? Is it widely trusted?



# Select secure OSS component (2 of 2)

---

- **Examine product for yourself (or hire someone to do so)**
  - **User/admin documentation: Discuss make/keep it secure?**
  - **OSS: Min. privileges, simplicity, carefully checks input**
  - **OSS: Scanning tools/peer review sample - no/few real problems**
  - **Do own penetration testing**
- **Examine development process**
  - **OSS: Developer mailing lists discuss security issues?**
  - **OSS: Is there an active community that reviews changes?**
  - **How are security vulnerabilities reported? Are they fixed?**
  - **Cryptographic signatures for current release?**
- **Who are the developers (*not* just the company; pedigree)?**
  - **Evidence that developers understand security (ML discussion)**
  - **OSS: Developers trustworthy? Id key developers: Criminal record? Incentive to insert malicious code? Treat proprietary** <sup>33</sup>
- **Use blind download/staging** [See [http://www.dwheeler.com/oss\\_fs\\_eval.html](http://www.dwheeler.com/oss_fs_eval.html)]

# Build (new) secure OSS component

---

- **Requirements: Identify security requirements, flow down**
- **Design: Limit privileges, etc.; see securing environment**
  - **Make modular so can (1) remove/replace subcomponents and (2) give different components different privileges**
- **Implementation**
  - **Be aware of & avoid common mistakes (e.g., CWEs)**
  - **Prefer implementation tools that prevent errors (vs. C, C++)**
  - **Turn on warning flags, use coding style that avoids mistakes**
  - **“So simple it’s obviously right”; OSS: Embarrassment factor**
- **Test**
  - **Use static & dynamic analysis tools to find vulnerabilities**
  - **Peer review. OSS: Mass peer review – enable it (e.g., common CM tools/languages/licenses; broad usefulness; incentives)**
  - **Develop & include automated regression test suite**
  - **Penetration testing**

# Building secure OSS component?

## *Smartly* start the OSS project

---

- **Check usual project-start requirements**
  - Is there a need, no/better solution, TCO, etc.
  - Examine OSS approach; similar to GOTS, with greater opportunity for cost-sharing, but greater openness
- **Cost-sharing: Remove barriers to entry**
  - Use common license well-known to be OSS (GPL, LGPL, MIT/X, BSD-new) – *don't write your own license, it's a common road to failure & very hard to overcome*
  - Establish project website (mailing list, tracker, source)
  - Document scope, major decisions
  - Use typical infrastructure, tools, etc. (e.g., SCM)
  - Maximize portability, avoid proprietary langs/libraries
  - *Must run* - Small-but-running better than big-and-not
  - Establish vetting process(es) before organization use
    - Organization-paid lead? Testing? Same issues: proprietary
- **Many articles & books on subject**

# Improve OSS component for operational environment (1 of 2)

---

- **See “Secure OSS component environment”**
  - OSS: Can apply inside component to give finer-grain protection
    - Embed (finer-grained) input filtering, drop (more) privileges (sooner), divide into smaller modules (different privileges)
- **OSS: Modify with added protective measures**
  - Add taint checking, compiler-inserted protective measures, switch to libraries with added defensive measures, etc.
- **Remove unneeded (can’t subvert what’s not there)**
  - Disable run-time flexibility: read-only media, embed plug-ins
  - OSS: Reconfig/comment out, recompile (e.g., embed modules)
- **OSS: Modify to use more secure components**
- **Intentional diversity (warning: Do not depend on these)**
  - OSS: modify & recompile for unusual processors (possibly simulated), unusual memory layouts, unusual compiler / compiler options, misleading headers, nonstandard protocols

# Improve OSS component for operational environment (2 of 2)

---

- **OSS: Use tools / peer review / pen testing / (competitor) CVE to find vulnerabilities (then repair component)**
  - **Tools: Use static or dynamic analysis to find likely problems (compiler flags, splint, flawfinder, fuzzers, etc.)**
    - **Examine reports to remove false alarms**
  - **Look for mailing list discussions on vulnerabilities.. or start one**
  - **Check that past CVEs on component have been *fully* addressed**
    - **Sometimes patch only partly fixes, or only fixes one of many**
  - **Check similar-component CVEs; may have same problem**
  - **Find code written by people you don't trust (via CM system)**
  - **Find components with high probable-defect density or high complexity (it may be best to rewrite them)**
- **After repair, contribute change to OSS project**
  - **So releases will include your repair with others' improvements**
  - **Use their process; initial security discussions often private**

# OSS as security-enabling strategy

---

- **Enables rapid change**
  - “FOSS [makes] it possible to change and fix security holes quickly... [allowing] rapid response to new or innovative forms of cyberattack... [this is] generally impractical in closed source products.” [MITRE 2003]
  - “in the Cyber Domain, technological agility will matter even more, because there will be no compensating physical assets” [Jim Stogdill’s “Coding is Maneuver”]
- **Frees from control by another (vendor lock-in / embargo)**
  - Proprietary software risks: vendor lock-in → high prices, poor product, abandonment, undesirable changes (DRM, data loss)
  - OSS: Can independently maintain code: modify for specific needs and/or retain desirable functionality, even if original developer uninterested

# Acquisition: Same (OSS vs. Proprietary)

---

- **Negotiate best options with all parties, *then* select**
- **Evaluation criteria**
  - **Functionality, cost, market share, support, maintenance/longevity, reliability, performance, scalability, flexibility, legal/license (inc. *rights and responsibilities*), other**
- **Warranty & indemnification**
  - **Often disclaimed by *both* proprietary & OSS licenses**
  - **Indemnification exception: Linux (OSDL, HP, RH, Novell)**
- **Developer trustworthiness usually unknown**
  - **Can review OSS code & sometimes proprietary**

# Acquisition: Different (OSS vs. Proprietary)

---

- **Process&code openness means more&different sources of evaluation information**
  - Bug databases, mailing list discussions, ...
  - Anyone can review/comment on design
  - Anyone (inc. you) can evaluate source code for security
  - See [http://www.dwheeler.com/oss\\_fs\\_eval.html](http://www.dwheeler.com/oss_fs_eval.html)
- **Proprietary=pay/use, OSS=pay/improvement**
  - In either case pay for installation, training, support
  - In OSS, pay can be time and/or money
- **Support can be competed & changed**
  - OSS vendors, government support contracts, self
- **OSS can be modified & redistributed**
  - New option, but need to know when to modify
  - Forking usually fails; generally work with community



# Evaluating OSS

---

- **Steps for evaluating OSS and proprietary are essentially the same. My process:**
  - *Identify* candidates
  - *Read reviews*
  - *Compare* (briefly) to needs
    - **Functionality, cost, market share, support, maintenance, reliability, performance, scalability, useability, security, flexibility, legal/license**
  - *Analyze* top candidates in more depth
- **But many differences in the details**
  - **Different information available**
  - **Support can be competed**
  - **Can be modified, redistributed**
  - **Outlay costs per improvement, not per seat**

# Can FLOSS be applied to custom systems?

---

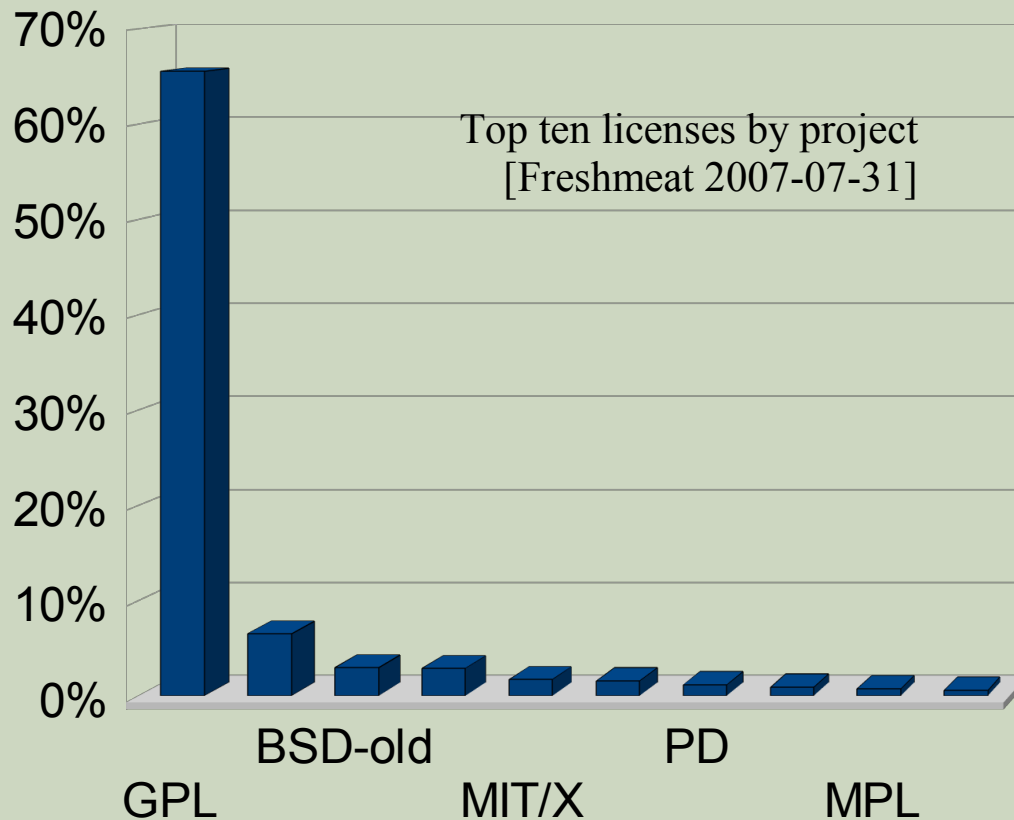
- **Effective FLOSS systems typically have built a large development community**
  - Share costs/effort for development & review
  - Same reason that proprietary off-the-shelf works: Multiple customers distribute costs
- **Custom systems can be built from FLOSS (& proprietary) components**
- **If no pre-existing system, sometimes can create a generalized custom system**
  - Then *generalized* system/framework FLOSS, with a custom config/plug-ins for your problem
  - Do risk/benefit analysis before proceeding

# Comparing GOTS, COTS Proprietary, and COTS OSS

Support Strategy	Flexibility	Cost	Risks
Government-owned / GOTS	High	High	Become obsolescent (government bears all costs & can't afford them)
COTS – Proprietary	Low	Medium*	Abandonment, & high cost if monopoly
COTS – OSS	High	Low*	As costly as GOTS if fail to build development community

OSS is not always the right answer...  
but it's clear why it's worth considering  
(both reusing OSS and creating new/modified OSS)

# Most Popular OSS Licenses



See <http://www.dwheeler.com/essays/gpl-compatible.html>

- Most OSS projects GPL
- GPL incompatibility foolish (MPL, BSD-old)
- Over 3/4 OSS projects use a top 10 license
- "Do not write a new license if it is possible to use [an existing common one]... many different and incompatible licenses works to the detriment of OSS because fragments of one program can not be used in another..." - Bruce Perens

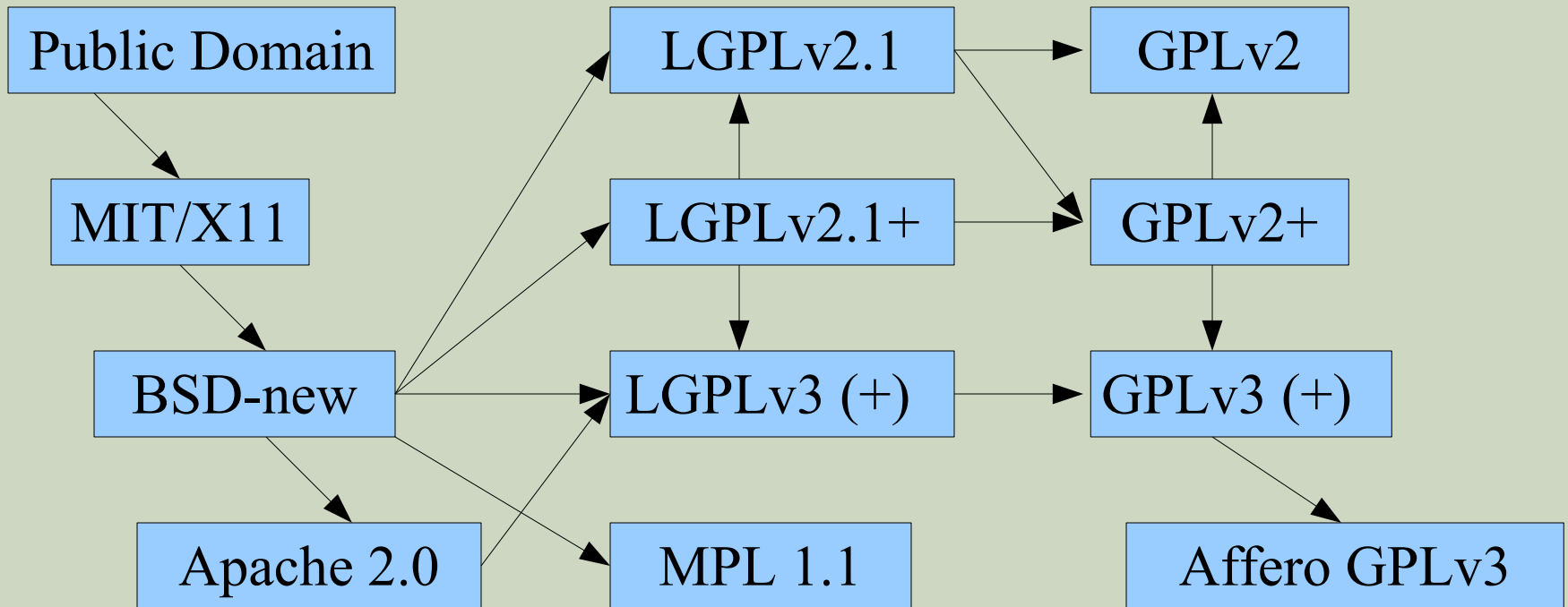
# FLOSS License Slide: Determining License Compatibility

---

*Permissive*

*Weakly  
Protective*

*Strongly  
Protective*



A→B means A can be merged into B

See <http://www.dwheeler.com/essays/floss-license-slide.html><sub>45</sub>

# Criteria for picking OSS license (If new/changed software)

---

1. **Actually OSS: Both OSI & FSF approved license**
2. **Legal issues**
  - Public domain (PD) if US government employee on clock
  - Otherwise avoid PD; use “MIT” for same result (lawsuits)
3. **If modification of existing project code, include its license**
  - Otherwise cannot share costs with existing project
4. **Encourage contributions: Use common existing license**
5. **Maximize future flexibility/reuse: Use GPL-compatible one!**
6. **Best meets goal:**
  - Use of new tech/standard: Permissive (MIT – alt., BSD-new)
  - \$ savings/longevity/common library: Weakly protective (LGPL)
  - \$ savings/longevity/common app: Strongly protective (GPL)
7. **Meets expected use (Mix with classified? proprietary?)**

# OSS licensing suggestions (if new/changed software)

---

- **Recommended short list: MIT/X, LGPL, GPL**
- **Avoid (unless modifying pre-existing software):**
  - **Artistic: Old version too vague, others better**
  - **MPL: GPL-incompatible**
  - **BSD-old: GPL-incompatible, obsolete (BSD-new replaces)**
- **Prefer MIT/X over BSD-new**
  - **MIT license simpler & thus easier to understand**
  - **BSD-new adds “can’t use my name in ads”, unclear legal value**
- **Caution: Apache 2.0 license compatible GPLv3, not GPLv2**
- **GPL: Version 2 or version 3?**
  - **Widest use is “GPL2+”; projects slowly transitioning to 3**
  - **Auto-transition (“GPLv2+”) - at least establish upgrade process**
- **Sometimes: Copyright assignment, dual-license**
- **To control just name/brand, use trademark (not copyright)**

# OSS challenges

---

- **Ensuring OSS fairly considered in acquisitions**
  - Some acquisition processes/policies not updated for OSS
  - Policy noncompliance (FAR's market research, "OSS neutral")
  - Many PMs unfamiliar with OSS: don't consider using or creating
  - Many OSS projects ignore solicitations & RFPs
  - Favor proposals with OSS – more rights
- **Different economics: Pay-up-front for improvements**
  - Some policies presume proprietary COTS' pay-per-use model
  - Can pay in \$ or time, can compete, can cost-share with other users
- **Transition costs if pre-existing system**
  - Especially if dependent on proprietary formats/protocols/APIs
  - Use open standards so can switch (multi-vendor, no 'RAND' patents)
    - Emphasize web-based apps/SOA/platform-neutral – & test it!
  - Vendor lock-in often increases TCO; transition may be worthwhile<sub>48</sub>



# Quick Aside: “Intellectual Rights”

---

- **Laws on software often called “intellectual property rights” (IPR)**
  - Copyright, trademark, patent, trade secret, ...
- **IPR term extremely misleading**
  - If I take your car, you have no car
  - If I copy your software.. you still have the software
  - Formal term: non-rivalrous
  - Failure to understand differences leads to mistaken thinking, especially regarding OSS
- **Knowledge & physical property fundamentally different**
  - U.S. Constitution permits exclusive rights only for limited times, solely “to promote the progress of science and useful arts”
- **Use term “intellectual rights” instead**
  - Avoids mis-thinking & clarifies that all parties have rights

# Many FLOSS tools support high assurance development

---

- **Configuration Management:** CVS, Subversion (SVN), GNU Arch, git/Cogito, Bazaar, Bazaar-NG, Monotone, Mercurial, Darcs, svk, Aegis, CVSNT, FastCST, OpenCM, Vesta, Supersversion, Arx, Codeville...
- **Testing:** [opensourcetesting.org](http://opensourcetesting.org) lists 275 tools Apr 2006, inc. Bugzilla (tracking), DejaGnu (framework), gcov (coverage), ...
- **Formal methods:** Community Z tools (CZT) , ZETA, ProofPower, Overture, ACL2, Coq, E, Otter/MACE, PTPP, Isabelle, HOL4, HOL Light, Gandalf, Maude Sufficient Completeness Checker, KeY, RODIN, Hybrid Logics Model Checker, Spin, NuSMV 2, BLAST, Java PathFinder, SATABS, DiVinE, Splint (as LCLint), ...
- **Analysis implementation:** Common LISP (GNU Common LISP (GCL), CMUCL, GNU CLISP), Scheme, Prolog (GNU Prolog, SWI-Prolog, Ciao Prolog, YAP), Maude, Standard ML, Haskell (GHC, Hugs), ...
- **Code implementation:** C (gcc), Ada (gcc GNAT), ...
  - **Java/C#:** FLOSS implementations (gcj/Mono) maturing; gc issue

# Acronyms (1)

---

**BSD: Berkeley Software Distribution**

**COTS: Commercial Off-the-Shelf (either proprietary or OSS)**

**DFARS: Defense Federal Acquisition Regulation Supplement**

**DISR: DoD Information Technology Standards and Profile Registry**

**DoD: Department of Defense**

**DoDD: DoD Directive**

**DoDI: DoD Instruction**

**EULA: End-User License Agreement**

**FAR: Federal Acquisition Regulation**

**FLOSS: Free-libre / Open Source Software**

**FSF: Free Software Foundation (fsf.org)**

**GNU: GNU's not Unix**

**GOTS: Government Off-The-Shelf (see COTS)**

**GPL: GNU General Public License**

**HP: Hewlett-Packard Corporation**

**IPR: Intellectual Property Rights; use "Intellectual Rights" instead**

**IT: Information Technology**

**LGPL: GNU Lesser General Public License**

# Acronyms (2)

---

**MIT: Massachusetts Institute of Technology**

**MPL: Mozilla Public License**

**NDI: Non-developmental item (see COTS)**

**OMB: Office of Management & Budget**

**OSDL: Open Source Development Labs**

**OSI: Open Source Initiative ([opensource.org](https://opensource.org))**

**OSJTF: Open Systems Joint Task Force**

**OSS: Open Source Software**

**PD: Public Domain**

**PM: Program Manager**

**RFP: Request for Proposal**

**RH: Red Hat, Inc.**

**ROI: Return on Investment**

**STIG: Security Technical Implementation Guide**

**TCO: Total Cost of Ownership**

**U.S.: United States**

**USC: U.S. Code**

**V&V: Verification & Validation**

Trademarks belong to the trademark holder.

# Interesting Documents/Sites

---

- **“Why OSS/FS? Look at the Numbers!”**  
[http://www.dwheeler.com/oss\\_fs\\_why.html](http://www.dwheeler.com/oss_fs_why.html)
- **“Use of Free and Open Source Software in the US Dept. of Defense”** (MITRE, sponsored by DISA)
- **President's Information Technology Advisory Committee (PITAC) -- Panel on Open Source Software for High End Computing, October 2000**
- **“Open Source Software (OSS) in the DoD,”** DoD memo signed by John P. Stenbit (DoD CIO), May 28, 2003
- **Center of Open Source and Government (EgovOS)**  
<http://www.egovos.org/>
- **OpenSector.org** <http://opensector.org>
- **Open Source and Industry Alliance** <http://www.osaia.org>
- **Open Source Initiative** <http://www.opensource.org>
- **Free Software Foundation** <http://www.fsf.org>
- **OSS/FS References**  
[http://www.dwheeler.com/oss\\_fs\\_refs.html](http://www.dwheeler.com/oss_fs_refs.html)