Securing Open Source Software (OSS or FLOSS)

David A. Wheeler

May 5, 2008

Outline

- Introduction to Open Source Software (OSS)
  - Terminology, Why develop/use OSS
  - Typical development model
- OSS is commercial, extant OSS is COTS
- Fundamental Security Principle: Open Design
- Securing OSS
  - Secure OSS component's environment
  - Secure OSS component itself
    - Select, build (new), or improve (existing)
- OSS as security-enabling strategy
- Malicious code & OSS

What is Open Source Software (OSS)?

- OSS: software licensed to users with these freedoms:
  - 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.)
- Original term: “Free software” (confused with no-price)
- Other synonyms: libre sw, free-libre sw, FOSS, FLOSS
  - OSS most common in DoD (I often use “FLOSS” to non-DoD)
- Antonyms: proprietary software, closed software
- Widely used; OSS #1 or #2 in many markets
  - “… plays a more critical role in the DoD than has generally been recognized.” [MITRE 2003]
- <u>Released OSS is commercial software</u>

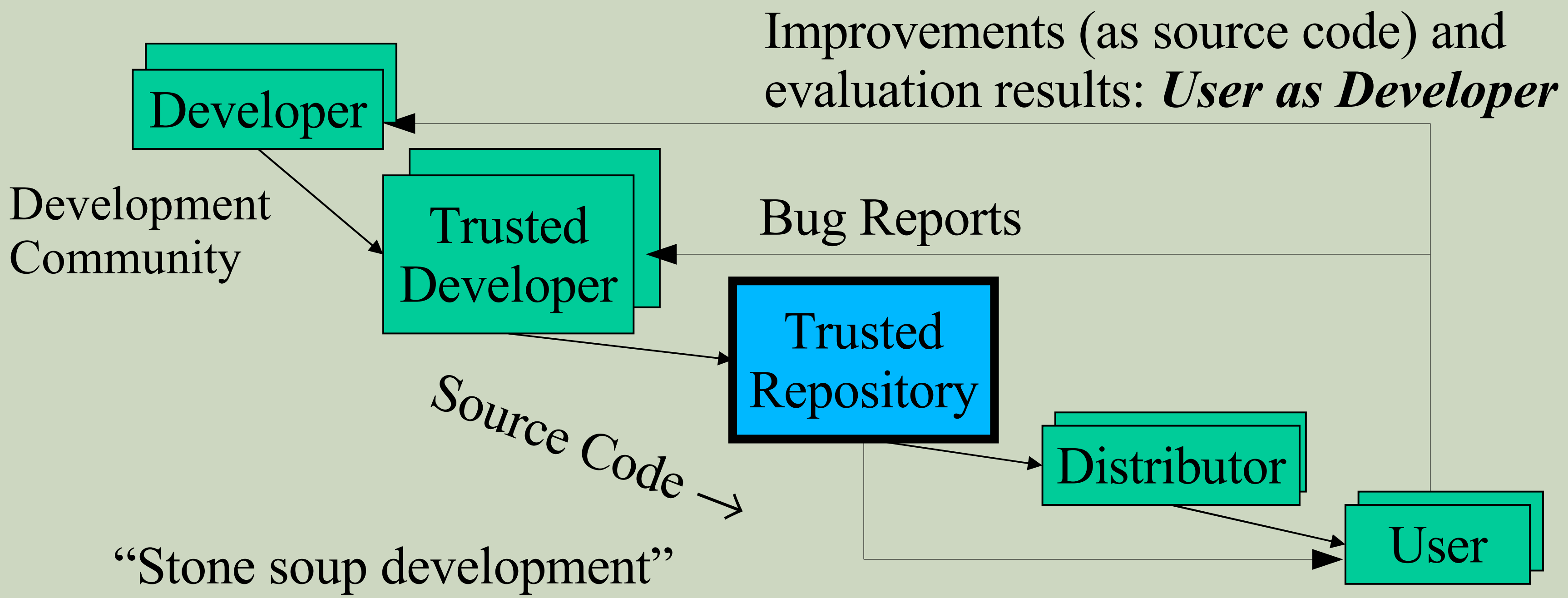
[For details see “Free Software Definition” & “Open Source Definition”]

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

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)

OSS is commercial, extant OSS is COTS

- **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 [<u>not government-unique</u>], and (i) Has been <u>sold</u>, leased, or <u>licensed</u> to the general public; or (ii) Has been <u>offered</u> for <u>sale</u>, lease, or <u>license</u> 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, <u>including the receipt of other copyrighted works</u>."**
- **OMB Memo M-03-14 (Commercial software): OSS support**
- **Important: U.S. Law (41 USC 403), FAR, DFARS <u>require</u> 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) <u>Acquire</u> [them] when… available … (c) Require prime contractors and subcontractors <u>at all tiers</u> to incorporate, to the <u>maximum extent practicable</u>, [them] as components..."**

OSS is clearly commercial by other measures too

- 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
- Sell service/hw, commoditize complements, avoid costs
- Users use COTS/NDI because they share costs – OSS does!
- Even GPL'ed software is commercial

[See http://www.dwheeler.com/essays/commercial-floss.html]

Extreme security claims for OSS

- Extreme claims
  - “OSS is always more secure”
  - “Proprietary is always more secure”
- Reality: Neither OSS nor proprietary always better
  - Some *specific* OSS programs *are* more secure than their competing proprietary competitors
  - Include OSS options when acquiring, then evaluate
- There *is* a principle that gives OSS a *potential* advantage…

Open design:
A security fundamental

- Saltzer & Schroeder [1974/1975] - Open design principle
  - the protection mechanism must not depend on attacker ignorance
- OSS better fulfills this principle
- Security experts perceive OSS 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”

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**
  - ***<u>Don't need to do all of them</u>*; 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 =
- Use blind download/staging

[See 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

[See NDIA guidebook, http://www.dwheeler.com/secure-programs]

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**

Malicious code & OSS (1 of 2)

- "Anyone can modify OSS, 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 OSS, requires subverting/misleading the trusted developers or trusted repository/distribution…
  - *and* no one noticing the public malsource later
- Different threat types: Individual...nation-state
- Large OSS projects more difficult to undetectably subvert
  - Tend to have many reviewers from many countries
  - Distributed source – easy to detect/determine changes
  - Consider supplier as you would proprietary software
  - Risk often larger for small/niche OSS projects
- Detection & repair in OSS much easier than proprietary
- Can often determine developer identity for OSS

Malicious code & OSS (2 of 2)

- OSS repositories demonstrate great resilience vs. attacks
  - Linux kernel (2003); hid via “= instead of ==”
    - Attack failed (CM, developer review, conventions)
  - SourceForge/Apache (2001), Debian (2003)
    - Countered & restored using external copies
- Malicious code can be made to look unintentional
  - Techniques to counter unintentional still apply
  - Attacker could try to work around tools... but for OSS won't know what tools will be used!
- Borland InterBase/Firebird Back Door
  - user: politically, password: correct
  - Hidden for 7 years in proprietary product
  - Found after release as OSS in 5 months
  - Unclear if malicious, but has its form

Bottom line

- Neither OSS nor proprietary always better
  - But clearly many cases where OSS *is* better
  - OSS better meets “open design” principle
  - By definition, OSS gives more rights to its user community
- Include OSS options when acquiring, then evaluate
  - Acquisitions for government contracts are <u>required</u> to consider existing OSS
  - Consider both reusing existing <u>and</u> developing new OSS
- Many ways to secure OSS components
  - Secure OSS component’s environment
  - Secure OSS component itself
    - Select, build (new), or improve (existing)

Backup slides

OSS Security Preconditions (Unintentional vulnerabilities)

1. Developers/reviewers need security knowledge
   - Knowledge more important than licensing
2. Developers/reviewers have to actually *use* that knowledge in development & *review* the code
   - Reduced likelihood of review if niche/rarely-used, few developers, rare computer language, not really OSS
   - More contributors, more review
     - <u>Is it *truly* community-developed?</u>
   - Review really does happen
     - Tool vendors: Coverity, Fortify, etc.
     - Review projects: OpenBSD, Debian Security Audit, ...
     - Project-specific: Mozilla bounty, etc.
3. Problems must be fixed
   - Far better to fix *before* deployment
   - If already deployed, need to deploy fix

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
  - “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 to many. 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
- Incompatible with off-the-shelf development approaches
  - Fine for (custom) classified software, but that’s costly
- Proprietary software *can* be secure – but not this way

Proprietary advantages? Not really

- Experienced developers who understand security produce better results
  - Experience & knowledge *are critical*, but...
  - OSS developers often very experienced & knowledgeable too (BCG study: average 11yrs experience, 30 yrs old) – often same people
- Proprietary developers higher quality?
  - Dubious; OSS often higher reliability, security
  - Market rush often impairs proprietary quality
- No guarantee OSS is widely reviewed
  - *True!* Unreviewed OSS 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

Evaluating Existing COTS: What's the Same? (OSS vs. Proprietary)

- Negotiate best options with all parties, *then* select
- Evaluate by winnowing out top candidates for your needs
  - *Identify* candidates, Read *Reviews, Compare* (briefly) to needs through criteria, *Analyze* top candidates
- 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
- Warranty & indemnification ("who do you sue?")
  - Generally disclaimed by *both* proprietary & OSS licenses
  - Red Hat, HP, Novell offer Linux system indemnification
- Pay for installation, training, support (time and/or money)
- Developer trustworthiness usually unknown
  - Mitigation: Can review OSS code & sometimes proprietary
  - Mitigation: Supplier due diligence; often main OSS developers and integrators determinable
  - Remember: Selling company often not developer

Evaluating Existing COTS: What’s Different? (OSS vs. Proprietary)

- Process/code openness means more & different sources of evaluation information for COTS OSS
  - Bug databases, mailing list discussions, detailed documentation, CM changes, source
  - Anyone (inc. you) can evaluate in detail (or pay to)
  - See http: //www.dwheeler.com/oss_fs_eval.html
- Proprietary=pay/use, OSS=pay/improvement
  - 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 COTS OSS: Some specific differences

- Functionality
  - If doesn't meet needs, determine cost to add what's missing
- Cost: Include all costs over long period, for all options
  - Transition, training, support, additional proprietary licenses, etc.
  - Long-term thinking critical; OSS may be more expensive at first (from transition & changes) yet be less expensive long-term (from upgrade & proprietary license fees from additional units)
- Support: May be >1 viable option!
  - Project-focused/sponsor, OSS support, industry-specific support (e.g., government contractor), community+self support
  - If >1 viable option, treat as separate options
- Reliability: Automatic test suite provided?
- Security: Coverity/Fortify scan, OpenBSD/Debian audit...
- License: Is it really OSS? (OSI & FSF approved license)

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)

DoD Open Technology Development

- **"Open Technology Development Roadmap Plan"**
  - **Apr 2006**
  - **http://www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf**
- **Three goals:**
  1. **Leverage open source infrastructure and technologies**
  2. **Apply open source collaborative technologies**
  3. **Change the default acquisitions and development behavior to default to technology services vs. products**
- **Implementation strategy:**
  - **Crawl: Open standards, interfaces, data**
  - **Walk: Open source & concept methodology**
  - **Run: Service/DoD/Industry source repositories**

OSS non-challenges

1. **COTS support if no traditional vendor**
   - Compete-able in traditional fashion
2. **License compliance**
   - Usually much easier than proprietary: Illegal becomes legal
     - Distributing unchanged program *encouraged* by OSS
     - Distributing changed program *encouraged* by OSS... but may require some actions (attribution, re-release)
   - Different, so need education (PMs, developers, lawyers)
3. **OSS in classified systems**
   - Unchanged programs: non-issue, use as-is
   - Privately modify if permitted by license
     - Ok if (1) permissive or (2) protective & don't "distribute"*
     - Usually unwise - bear large maintenance costs
   - Put classified material in data tables / plug-ins
   - Layer/modularize into separate unlinked pieces
   - Either better for confidentiality - reduce need for access

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

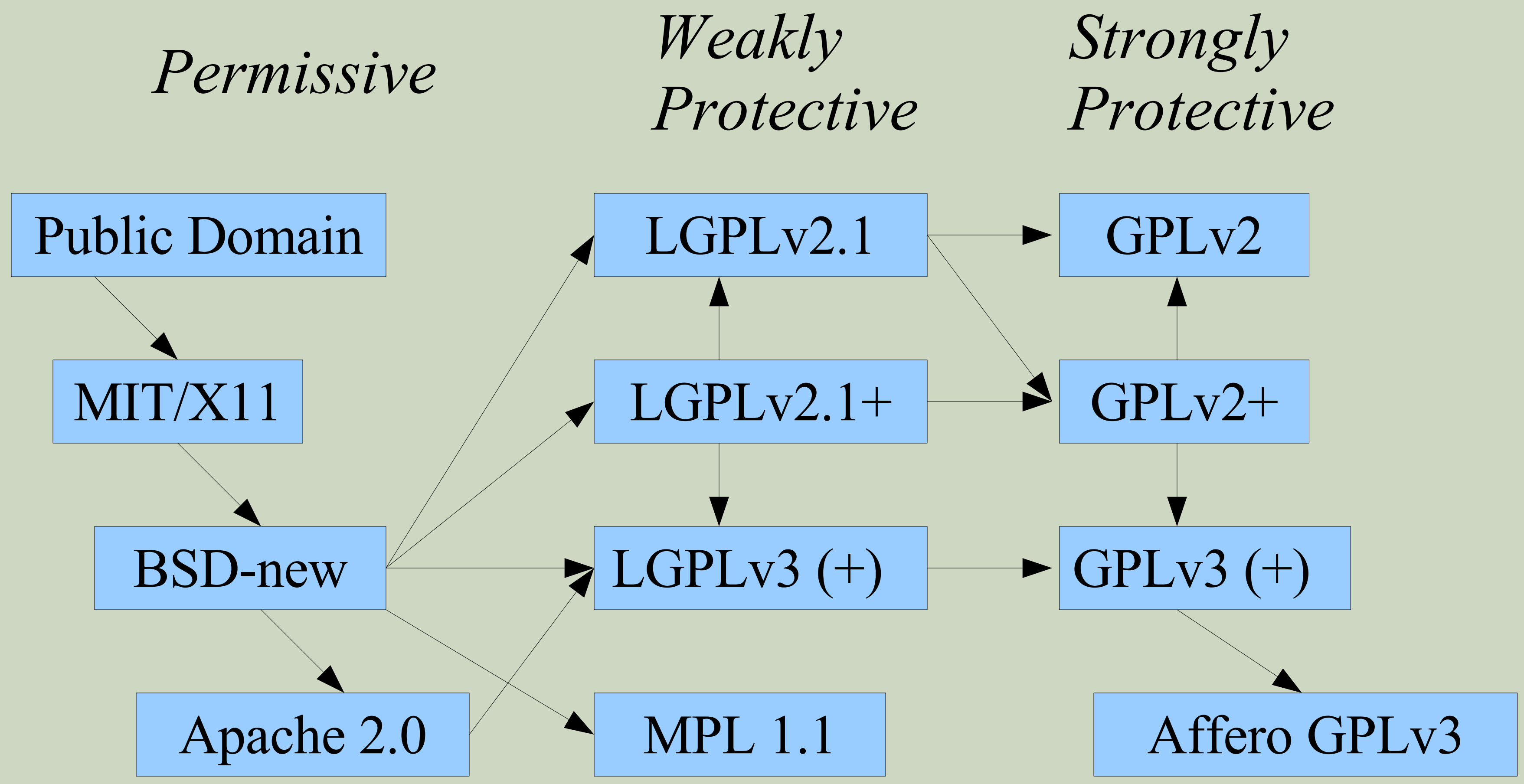
Quick Aside: “Intellectual Rights”

- 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

Types of OSS licenses

- Copyright law: Must have permission to copy software
  - Permission is given by a license
  - Proprietary software: Pay for a license to use a copy/copies
  - OSS licenses grant more rights, but still conditional licenses
- Over 100 OSS licenses, but only a few widely used
- Can be grouped into three categories (differing goals):
  - Permissive: Can make proprietary versions (MIT, BSD-new)
  - Weakly protective: Can't distribute proprietary version *of this component,* but *can* link into larger proprietary work (LGPL)
  - Strongly protective: Can't distribute proprietary version *or* directly combine (link) into proprietary work (GPL)
- The most popular OSS licenses tend to be compatible
  - Compatible = you can create larger programs by combining software with different licenses (must obey all of them)

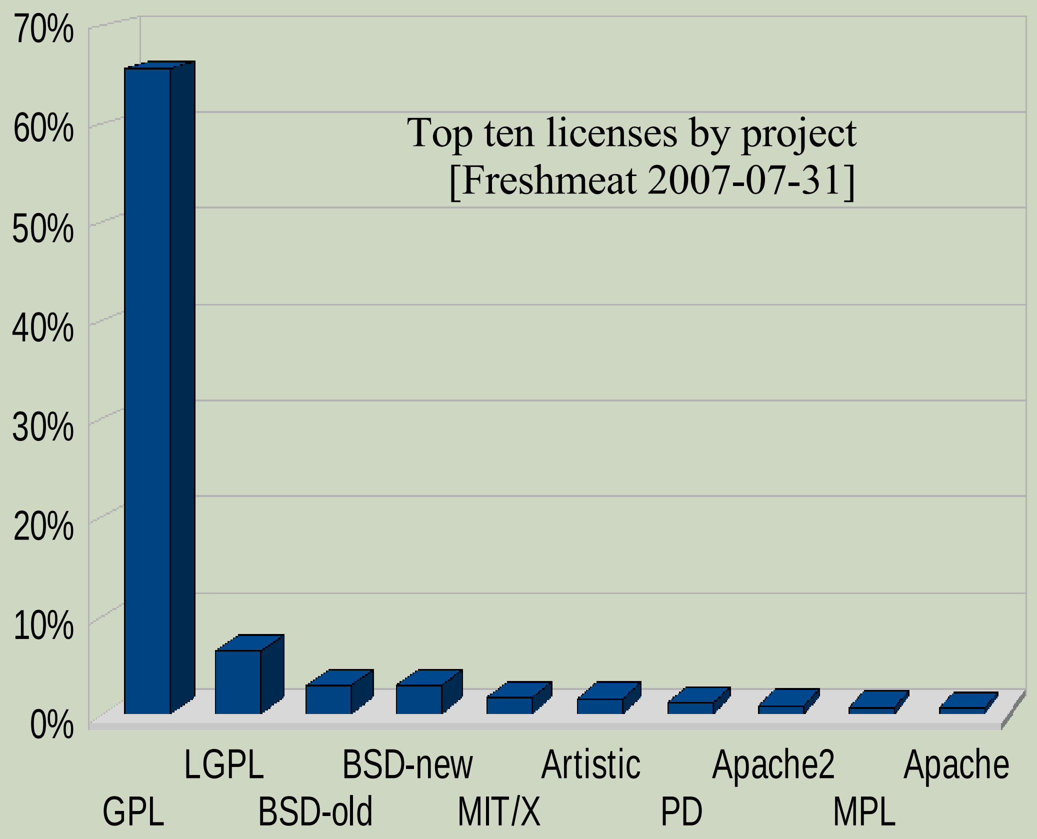
FLOSS License Slide: Determining License Compatibility



A→B means A can be merged into B

See http://www.dwheeler.com/essays/floss-license-slide.html

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

Examples of OSS in U.S. government

- Use – ***pervasive***
  - OSS "plays a more critical role in the DoD than has generally been recognized"; inc. Linux, Samba, Apache, Perl, GCC, GNAT, XFree86, OpenSSH, bind, and sendmail. [MITRE 2003]
  - "devIS saves its clients a minimum of $100,000 per contract by using OSS" [NewsForge]
  - Often unaware it's OSS
- Government-paid improvements of OSS
  - OpenSSL (CC evaluation), Bind (DNSSEC), GNAT, …
- Government-developed OSS
  - BSD TCP/IP suite, Security-Enhanced Linux (SELinux), OpenVista, Expect, EZRO, Evergreen (Georgia), …
- U.S. federal policies explicitly neutral: OSS, or not, is fine
  - OMB memo M-04-16, DoD memo "OSS in DoD"
  - Examine *all* licenses before commit (GPL fine)

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

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 develop-ment 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)

What are open standards?

Not just “open mouth”. Merged Perens'/Krechmer's/EC's definition:

1. Availability: available for all to read and implement
2. Maximize End-User Choice: Create a fair, competitive market for implementations; NOT lock the customer in. Multiple implementors
3. No Royalty: Free for all to implement, with no royalty or fee
4. No Discrimination: Don't favor one implementor over another (open meeting, consensus/no domination, due process)
5. Extension or Subset: May be extended or offered in subset form
6. Predatory Practices: May employ license terms that protect against subversion of the standard by embrace-and-extend tactics
7. One World: Same standard for the same capability, world-wide
8. On-going Support: Supported until user interest ceases
9. No or nominal cost for specification (at *least*; open access?)

See http://www.dwheeler.com/essays/opendocument-open.html

Most popular OSS licenses

- Many licenses, but most use GPL, and over 3/4 projects use top 10
- "Do not write a new license if it is possible to use [an existing common license]... many different and incompatible licenses works to the detriment of OSS because fragments of one program can not be used in another program with an incompatible license." - Bruce Perens

<u>Top Ten OSS Licenses</u>

GPL:	65.50%
LGPL:	6.53%
BSD-old:	2.93%
BSD-new:	2.86%
MIT:	1.67%
Artistic:	1.55%
Public Domain:	1.15%
Apache 2.0:	0.86%
MPL:	0.72%
Apache (orig.):	0.56%

[Freshmeat 2007-07-31]

What's high assurance software?

- **"High assurance software": has an argument that could convince skeptical parties that the software will *always perform or never perform* certain key functions *without fail*... convincing evidence that there are *absolutely no software defects.***
  - **Formal methods, deep testing. CC EAL 6+**
  - **Today, extremely rare. Critical safety/security**
- **Medium assurance software: not high assurance, but significant effort expended to find and remove important flaws through review, testing, and so on. CC EAL 4-5**
  - **No proof it's flawless, just effort to find and fix**

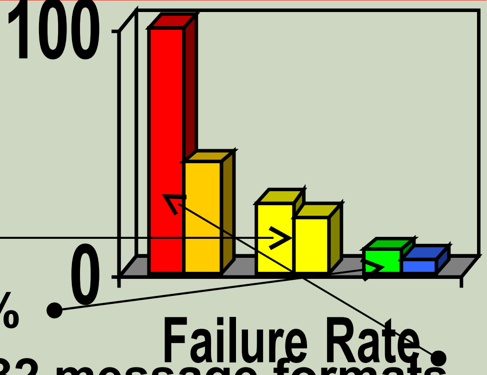
High assurance

- Many OSS tools that support developing HA
  - CM: CVS, Subversion (SVN), git, Mercurial, ...
  - Testing: opensourcetesting.org lists 275 tools Apr 2006, inc. Bugzilla (tracking), DejaGnu (framework), gcov (coverage), ...
  - Formal methods: ACL2, PVS, Prover9/Mace4, Isabelle, Alloy, ...
  - Analysis implementation: Common LISP (GNU Common LISP (GCL), CMUCL, GNU CLISP), Prolog (GNU Prolog, SWI-Prolog, Ciao Prolog, YAP), Standard ML, Haskell (GHC, Hugs), ...
  - Code implementation: C (gcc), Ada (gcc GNAT), ...
- HA OSS: Almost never tried (proprietary rare too)
- OSS should be better for HA – hope to see in future
  - In mathematics, proofs are often wrong, so only peer review of proofs valid [De Millo,Lipton,Perlis]. OSS!

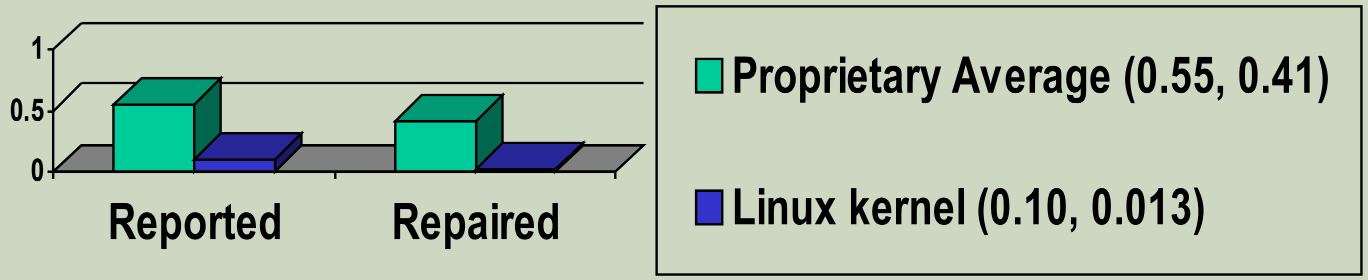
Formal methods & OSS

- Formal methods applicable to OSS & proprietary
- Difference: OSS allows public peer review
  - In mathematics, peer review often finds problems in proofs; many publicly-published proofs are later invalidated
  - Expect true for software-related proofs, even with proof-checkers (invalid models/translation, invalid assumptions/proof methods)
  - Proprietary sw generally forbids public peer review
- Formal methods challenges same
  - Few understand formal methods (*anywhere)*
  - Scaling up to “real” systems difficult
  - Costs of applying formal methods—who pays?
    - *May* be even harder for OSS
    - Not easy for proprietary either

OSS often very reliable



- **Fuzz studies found OSS apps significantly more reliable [U Wisconsin]**
  - **Proprietary Unix failure rate: 28%,23%**
  - **OSS: 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]**



[See http://www.dwheeler.com/oss_fs_why.html]

OSS security (1)

- **Browser “unsafe” days in 2004: 98% Internet Explorer, 15% Mozilla/Firefox (half of Firefox’s MacOS-only)**
- **IE 21x more likely to get spyware than Firefox [U of Wash.]**
- **Faster response: Firefox 37 days, Windows 134.5 days**
- **Evans Data: Linux rarely broken, ~virus/Trojan-free**
- **Serious vulnerabilities: Apache 0, IIS 8 / 3yrs**
- **J.S. Wurzler hacker insurance costs 5-15% more for Windows than for Unix or Linux**
- **Bugtraq vulnerability 99-00: Smallest is OpenBSD, Windows largest (**Don't quintuple-count!)
- **Windows websites more vulnerable in practice**

Category	Proprietary	OSS
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)

OSS security (2)

- Unpatched networked systems: 3 months Linux, hours Windows (variance minutes ... months) [Honeynet.org, Dec 2004]
  - Windows SP2 believed to be better than previous versions of Windows
- 50% Windows vulnerabilities are critical, vs. 10% in Red Hat [Nicholas Petreley, Oct 2004]
- Viruses primarily Windows phenomenon
  - 60,000 Windows, 40 Macintosh, 5 for commercial Unix versions, 40 for Linux
- 91% broadband users have spyware on their home computers (proprietary OS) [National Cyber Security Alliance, May 2003] vs. ~0% on OSS

OSS security (3)

- OSS systems scored better on security [Payne, Information Systems Journal 2002]
- Survey of 6,344 software development managers April 2005 favored OSS [BZ Research]

Common Criteria & OSS

- Common Criteria (CC) can be used on OSS
  - Red Hat Linux, Novell/SuSE Linux, OpenSSL
- CC matches OSS imperfectly
  - CC developed before rise of OSS
  - Doesn’t credit mass peer review or detailed code review
  - Requires mass creation of documentation not normally used in OSS development
- Government policies discriminate against OSS
  - Presume that vendor will pay hundreds of thousands or millions for a CC evaluation (“big company” funding)
    - Presumes nearly all small business & OSS insecure
  - Presume that “without CC evaluation, it’s not secure”
  - Need to fix policies to meet real goal: secure software
    - Government-funded eval in exchange for free use?
    - Multi-agency/government funding?
    - Alternative evaluation processes?

Evaluating OSS? Look for evidence

- First, identify your security requirements
- Look for evidence at OSS 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

Concluding remarks

- **OSS options should always be considered**
  - **Both choosing COTS OSS & creating new OSS project**
  - **Components or even whole project (depending on need)**
  - **Not always best choice, but foolish to ignore**
- **OSS can be very flexible & often lowers costs**
  - **Directly and as competition to non-OSS (keep options open!)**
- **OSS raises strategic questions for governments**
  - **How pool users to start OSS projects when appropriate?**
  - **Educating PMs on OSS, deploying fully open architectures**
  - **Research: default to OSS (with some common OSS license)**
  - **Eliminating software patents**
- **Projects should change to consider OSS approaches:**
  - **PM education: OSS differences, fears, *<u>always consider option</u>***
  - **Classified systems: separate data & program, layer programs**
  - **Open standards so can change later (e.g., browser-neutral)**
    - ***Require & operationally demonstrate* that can switch components**

OSS consistent with DoD policy

- DoD memo "Open Source Software (OSS) in the Department of Defense (DoD)" (2003)
  - OSS is fine if meets usual requirements
  - Must comply with licenses (OSS ≠ proprietary, GPL ok)
  - Tries to counter bias against “new” approach
- OMB M-04-16 “Software Acquisition” (July 1, 2004)
- Dept. of the Navy “OSS Guidance” (June 5, 2007)
- Some misunderstand DoDD 8500.1/DoDI 8500.2 DCPD-1 as forbidding OSS...

DoDD 8500.1/DoDI 8500.2 DCPD-1 does not apply to OSS

- **DoDD 8500.1/DoDI 8500.2 DCPD-1 "Public Domain Software Controls" does *<u>not</u>* apply to OSS**
  - **"Binary or machine executable ... software products and other software products with limited or no warranty such as those commonly known as freeware or shareware are not [to be] used in DoD information systems ..." <u>don't stop here!</u>**
  - **"[because they're] difficult or impossible to review, repair, or extend, <u>given that the Government does not have access to the original source code and there is no owner who could make such repairs on behalf of the Government</u>."**
  - **Clearly doesn't apply to OSS – source code <u>is</u> available**
    - **this is abandoned binary-only**
  - **Confirmed by General Desktop Application STIG**
    - **DoDI 5200.2 section E3.2.6 references DISA/NSA guides**
    - **STIG Version 3, Release 1 (09 March 2007), section 2.4**

Examples of U.S. government-sponsored OSS development (1 of 2)

- **BSD TCP/IP implementation: BSD (-old, later -new)**
  - Maximize use of new tech/standard (TCP/IP, basis of Internet)
- **Expect: Public Domain**
  - Legally required - government (NIST) employee, on clock
- **SELinux: GPL**
  - Reuse existing components (Linux kernel)
- **GNAT: GPL (Ada compiler), GPL+exception (library)**
  - Library: weakly protective license – clearly permits use by proprietary/classified apps, yet keeps library itself OSS
  - Reuse past components (gcc compiler)
  - Encourage use of standard (Ada)
  - Cost savings/longevity of app - previous Ada compilers $$$$$
- **Workforce Connections/EZRO: GPL**
  - Probably for cost savings/longevity of app

Examples of U.S. government-sponsored OSS development (2 of 2)

- **Evergreen (library management): GPL**
  - U.S. state government (Georgia Public Library Service)
  - Probably for cost savings/longevity of app
  - Could not find existing application with needed functionality
- **Delta3D (display/simulation engine): Mostly LGPL**
  - MOVES Institute - Naval Postgraduate School
  - Cost savings/longevity/max use of library
  - Proprietary $300K..$1000K/application, follow-on requires another fee - could not afford to field developed simulations
    - "[by] owning the IP... if [customers] want to do a version 2, they have to come back to you. It guarantees... downstream revenue." - Doug Whatley, CEO Breakaway Games [JDMS, July 2006, http://www.scs.org/pubs/jdms/vol3num3/JDMSIITSECvol3no3McDowell143-154.pdf]
  - Flexibility important - enables modification as needed

A few other myths...

- Myth: OSS unsupported
  - Businesses support OSS. Red Hat, Novell, HP, Sun, IBM, DMSolutions, SourceLabs, OpenLogic, Carahsoft, ...
  - Community support often good; 1997 InfoWorld “Best Technical Support” award won by Linux User Community
- Myth: Only programmers care about software licenses
  - Bob Young: “Would you buy a car with the hood welded shut?... We demand the ability to open the hood... because it gives us, the consumer, control over [what] we’ve bought ... [if a dealer] overcharges us, won’t fix the problem... or refuses to install [something, others] would be happy to have our business”
- Myth: Developers just (inexperienced) college students
  - BCG study: Average OSS developer 30yrs old, 11yrs experience
- Myth: OSS is no cost
  - Training, support, transition, etc. are not free-of-cost
  - Competition often produces lower TCO & higher ROI for OSS

Interesting Documents/Sites

- "Why OSS/FS? Look at the Numbers!" 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

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)
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