Free-Libre / Open Source Software (FLOSS) and Software Assurance / Software Security

David A. Wheeler
December 11, 2006

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.

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

- Free-Libre / Open Source Software (FLOSS) programs 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

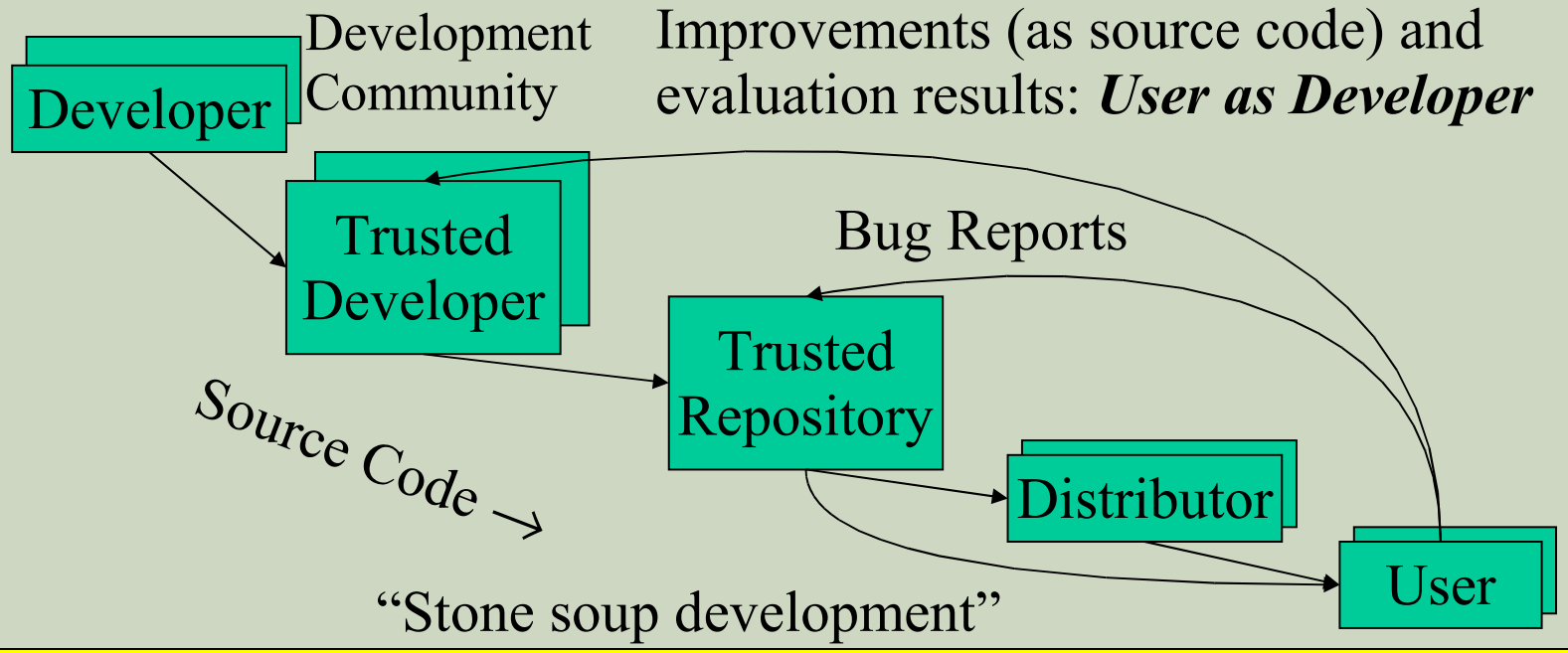
Definition: Software Assurance

- “Software assurance (SwA) relates to the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software.”

Outline

- Introduction: FLOSS development
- Extreme claims
- Unintentional vulnerabilities
  - Statistics on security & reliability
  - Open design: A security fundamental
  - FLOSS security preconditions (unintentional)
  - Proprietary advantages… not necessarily
- Intentional/malicious vulnerabilities
- High assurance
- FLOSS bottom line
- Backup: High assurance (details), Common Criteria, formal methods

Typical FLOSS Development Model



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

Extreme claims

- Extreme claims
  - "FLOSS is always more secure"
  - "Proprietary is always more secure"
- Reality: Neither FLOSS nor proprietary always better
  - Some *specific* FLOSS programs *are* more secure than their competing proprietary competitors
- Include FLOSS options when acquiring, then evaluate

FLOSS 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	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)

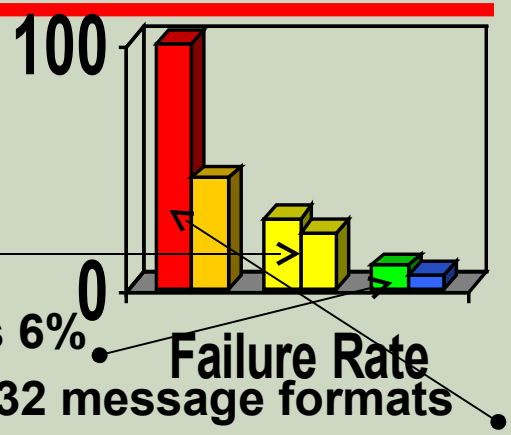
FLOSS Security (2)

- 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 FLOSS

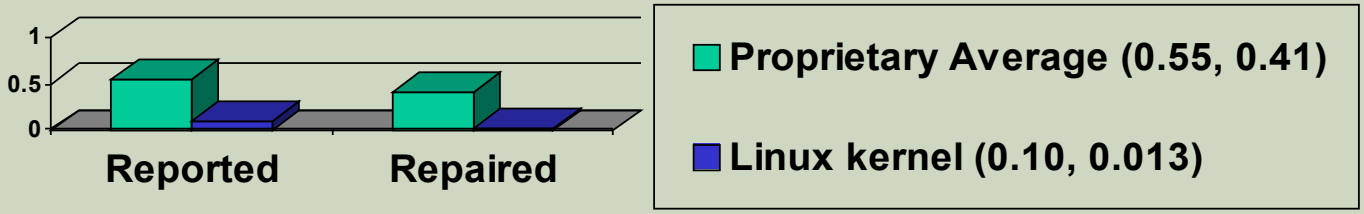
FLOSS Security (3)

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

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
- GNU/Linux vs. Windows NT 10 mo study [ZDNet]
  - NT crashed every 6 weeks; both GNU/Linuxes, never
- IIS web servers >2x downtime of Apache [Syscontrol AG]
- Linux kernel TCP/IP had smaller defect density [Reasoning]



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 denegrated
- 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
- ***Useless*** in most cases!
  - Incompatible with proprietary off-the-shelf model
- Proprietary software *can* be secure – but not this way

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! (next)
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)
- Many independents (see Bugtraq, etc.)
- Business case: Must examine to change (*reason* to review)
- Users' increased transparency encourages 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

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

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

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

Examples: Malicious code & FLOSS

- 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
  - 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

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

High Assurance

- High assurance (HA) 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.* CC EAL 6+
  - Significant use of formal methods, high test coverage
  - High cost – requires deep pockets at this time
  - A few OSS & proprietary tools to support HA dev
  - Few proprietary, even fewer OSS HA at this time
- Theoretically OSS should be better for HA
  - In mathematics, proofs are often wrong, so only peer review of proofs valid [De Millo,Lipton,Perlis]. OSS!
- HA developers/customers *very* conservative & results often secret, so rarely apply "new" approaches like OSS... yet
  - Cannot easily compare in practice... yet

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 FLOSS, with a custom configuration for your problem
  - Do risk/benefit analysis before proceeding

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
  - Can be a challenge because of radically different assumptions & approach
- Include FLOSS options when acquiring, then evaluate

Appendix: High Assurance

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.* 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**

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, Superversion, Arx, Codeville...
- Testing: 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, PTTP, 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

Test coverage

- **Many high assurance projects must meet measurable requirements on their tests. Common measures:**
  - **Statement (line) coverage: % program statements exercised by at least one test.**
  - **Branch coverage: % of "branches" from decision points (if/then/else, switch/case, ?:) exercised by at least one test**
  - **Some argue unit testing (low-level tests) should achieve 100% in both; most say 80%-90% okay**
  - **Many other measures**
- **FLOSS tools available (e.g., gcov)**

Formal methods: Introduction

- Formal methods: application of rigorous mathematical techniques to software development
- Three levels (& often focused application):
  - 0: Formal specification (using mathematics)
  - 1: Go deeper, e.g., (a) refine specification to a mathematically-defined design or more detailed model, and/or (b) prove important properties of the specification and/or model
  - 2: Fully prove that the code matches the design specification. Very expensive, if done at all, often done with only the most critical portions
- Tool types: Specification handling, theorem provers, model checkers, ...

Formal Method Specification Languages

- Z: ISO/IEC 13568:2002
  - Community Z tools (CZT) project, ZETA, ProofPower
- VDM
  - Overture
- B
  - RODIN Project
- UML Object Constraint Language
  - KeY

Sample formal method tools

- **ACL2**
  - **Industrial-strength theorem prover using LISP notation, Boyer-Moore family (ACM Software System Award)**
  - **Used for processor designs, microcode, machine code, Java bytecode, ... found 3 defects in BSD C string library**
- **Otter**
  - **High-performance general-purpose (math) prover**
- **Isabelle, HOL 4**
  - **Theorem prover, "drive" via ML**
- **Spin**
  - **Model-checking to verify distributed software systems. Used for Netherlands flood control barrier, Deep Space 1, Cassini, Mars Exploration Rovers, Deep Impact. ACM Software System Award**

High assurance components rare; OSS especially

- Closest: GNAT Pro High-Integrity Edition (HIE), EROS, few library routines
- FLOSS strong in medium assurance
- Why few high assurance?
  - Too expensive to do using FLOSS? Unlikely, if there's an economic incentive
  - No possibility of a rational economic model? Unlikely; components exist where cost avoidance sensible
  - Expertise too specialized? Probably partly true
  - Few considered the possibility? Yes; many potential customers/users have failed to consider an FLOSS option at all (e.g., consortium)

High assurance conclusions

- Many FLOSS tools available for creating high assurance components
- Few FLOSS high assurance components
  - Decision-makers failing to consider FLOSS-based approaches... they *should* consider them, so can choose when appropriate
- Government-funded software development in academia should normally be released under a FLOSS license
  - Many tools lost forever if not released as FLOSS
  - Build on it vs. start over. GPL-compatible
- FLOSS developers should consider developing or improving high-assurance components or tools

Backup Slides

Common Criteria & FLOSS

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

Formal Methods & FLOSS

- Formal methods applicable to FLOSS & proprietary
- Difference: FLOSS 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 FLOSS
    - Not easy for proprietary either

MITRE 2003 Report

"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. It would also 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. Finally, it would 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, http://www.egovos.org/

Acronyms

- COTS: Commercial Off-the-Shelf (either proprietary or FLOSS)
- DoD: Department of Defense
- HP: Hewlitt-Packard Corporation
- JTA: Joint Technical Architecture (list of standards for the DoD); being renamed to DISR
- OSDL: Open Source Development Labs
- FLOSS: Open Source Software
- RFP: Request for Proposal
- RH: Red Hat, Inc.
- U.S.: United States

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

Free-Libre / Open Source Software (FLOSS) and Software Assurance / Software Security

David A. Wheeler
December 11, 2006

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.

1

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

- Free-Libre / Open Source Software (FLOSS) programs 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

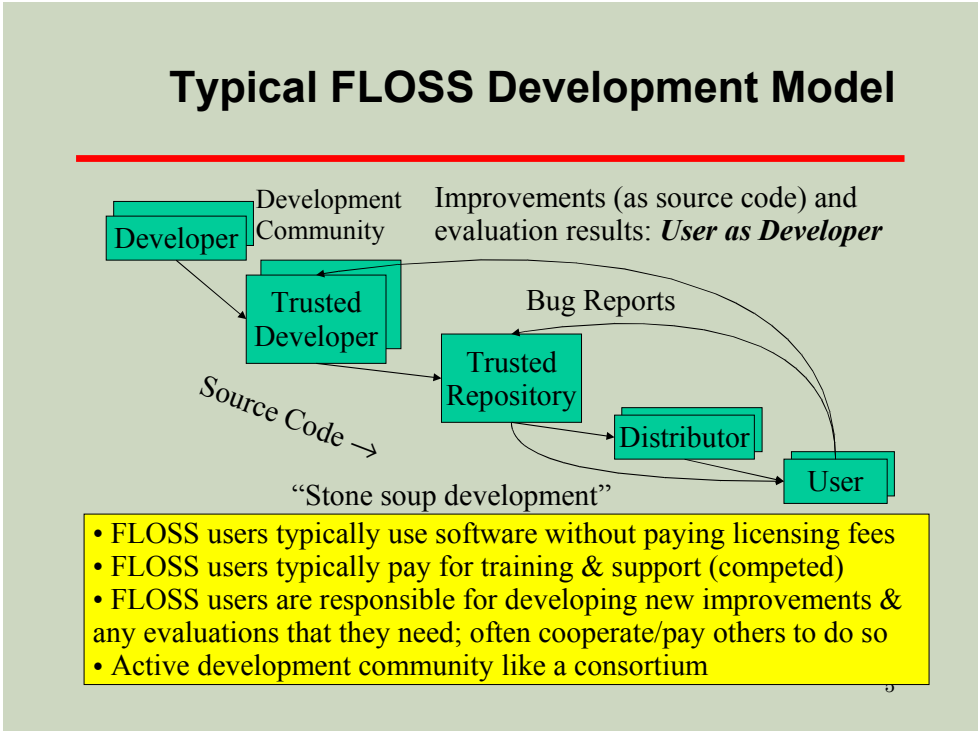
Definition: Software Assurance

- “Software assurance (SwA) relates to the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software.”

Outline

- Introduction: FLOSS development
- Extreme claims
- Unintentional vulnerabilities
  - Statistics on security & reliability
  - Open design: A security fundamental
  - FLOSS security preconditions (unintentional)
  - Proprietary advantages… not necessarily
- Intentional/malicious vulnerabilities
- High assurance
- FLOSS bottom line
- Backup: High assurance (details), Common Criteria, formal methods

4



Development model

User

User as developer

Developer

Trusted developer

Trusted repository

Distributor

Stone soup development

Source code

Licensing fees

Training

Support

Evaluations

Improvements

Cooperate

Extreme claims

- **Extreme claims**
  - **“FLOSS is always more secure”**
  - **“Proprietary is always more secure”**
- **Reality: Neither FLOSS nor proprietary always better**
  - **Some *specific* FLOSS programs *are* more secure than their competing proprietary competitors**
- **Include FLOSS options when acquiring, then evaluate**

FLOSS 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	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)

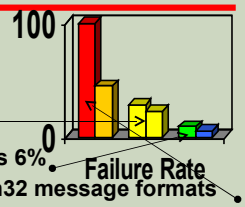
FLOSS Security (2)

- 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 FLOSS

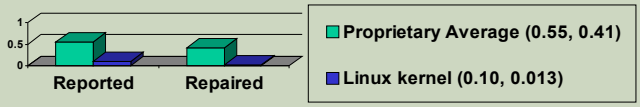
FLOSS Security (3)

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

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
- GNU/Linux vs. Windows NT 10 mo study [ZDNet]
  - NT crashed every 6 weeks; both GNU/Linuxes, never
- IIS web servers >2x downtime of Apache [Syscontrol AG]
- Linux kernel TCP/IP had smaller defect density [Reasoning]



10

Reliability

Fuzz

GNU/Linux

Windows

Crash

Downtime

TCP/IP

Defect

Defect density

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 denegrated
- 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
- ***Useless*** in most cases!
  - Incompatible with proprietary off-the-shelf model
- Proprietary software *can* be secure – but not this way

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! (next)**
3. **Problems must be fixed**
   - **Far better to fix *before* deployment**
   - **If already deployed, need to deploy fix**

15

Username “politically” passowrd “correct” was a major Interbase backdoor, hidden for years when it was proprietary, and found quickly soon after it was releaesd as FLOSS. It’s not known if, or how widely, this was exploited before it was revealed.

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)
- Many independents (see Bugtraq, etc.)
- Business case: Must examine to change (*reason* to review)
- Users' increased transparency encourages 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

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

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

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

Examples: Malicious code & FLOSS

- 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
  - 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

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

High Assurance

- High assurance (HA) 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.* CC EAL 6+
  - Significant use of formal methods, high test coverage
  - High cost – requires deep pockets at this time
  - A few OSS & proprietary tools to support HA dev
  - Few proprietary, even fewer OSS HA at this time
- Theoretically OSS should be better for HA
  - In mathematics, proofs are often wrong, so only peer review of proofs valid [De Millo,Lipton,Perlis]. OSS!
- HA developers/customers *very* conservative & results often secret, so rarely apply "new" approaches like OSS... yet
  - Cannot easily compare in practice... yet

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 FLOSS, with a custom configuration for your problem
  - Do risk/benefit analysis before proceeding

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
  - Can be a challenge because of radically different assumptions & approach
- Include FLOSS options when acquiring, then evaluate

Appendix: High Assurance

- Click to add an outline

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.* 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**

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, Superversion, Arx, Codeville...
- Testing: 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, PTTP, 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

Test coverage

- Many high assurance projects must meet measurable requirements on their tests. Common measures:
  - Statement (line) coverage: % program statements exercised by at least one test.
  - Branch coverage: % of "branches" from decision points (if/then/else, switch/case, ?:) exercised by at least one test
  - Some argue unit testing (low-level tests) should achieve 100% in both; most say 80%-90% okay
  - Many other measures
- FLOSS tools available (e.g., gcov)

Formal methods: Introduction

- Formal methods: application of rigorous mathematical techniques to software development
- Three levels (& often focused application):
  - 0: Formal specification (using mathematics)
  - 1: Go deeper, e.g., (a) refine specification to a mathematically-defined design or more detailed model, and/or (b) prove important properties of the specification and/or model
  - 2: Fully prove that the code matches the design specification. Very expensive, if done at all, often done with only the most critical portions
- Tool types: Specification handling, theorem provers, model checkers, ...

Formal Method Specification Languages

- **Z: ISO/IEC 13568:2002**
  - **Community Z tools (CZT) project, ZETA, ProofPower**
- **VDM**
  - **Overture**
- **B**
  - **RODIN Project**
- **UML Object Constraint Language**
  - **KeY**

Sample formal method tools

- **ACL2**
  - **Industrial-strength theorem prover using LISP notation, Boyer-Moore family (ACM Software System Award)**
  - **Used for processor designs, microcode, machine code, Java bytecode, ... found 3 defects in BSD C string library**
- **Otter**
  - **High-performance general-purpose (math) prover**
- **Isabelle, HOL 4**
  - **Theorem prover, "drive" via ML**
- **Spin**
  - **Model-checking to verify distributed software systems. Used for Netherlands flood control barrier, Deep Space 1, Cassini, Mars Exploration Rovers, Deep Impact. ACM Software System Award**

High assurance components rare; OSS especially

- Closest: GNAT Pro High-Integrity Edition (HIE), EROS, few library routines
- FLOSS strong in medium assurance
- Why few high assurance?
  - Too expensive to do using FLOSS? Unlikely, if there's an economic incentive
  - No possibility of a rational economic model? Unlikely; components exist where cost avoidance sensible
  - Expertise too specialized? Probably partly true
  - Few considered the possibility? Yes; many potential customers/users have failed to consider an FLOSS option at all (e.g., consortium)

High assurance conclusions

- Many FLOSS tools available for creating high assurance components
- Few FLOSS high assurance components
  - Decision-makers failing to consider FLOSS-based approaches... they *should* consider them, so can choose when appropriate
- Government-funded software development in academia should normally be released under a FLOSS license
  - Many tools lost forever if not released as FLOSS
  - Build on it vs. start over. GPL-compatible
- FLOSS developers should consider developing or improving high-assurance components or tools

Backup Slides

- Click to add an outline

Common Criteria & FLOSS

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

Formal Methods & FLOSS

- Formal methods applicable to FLOSS & proprietary
- Difference: FLOSS 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 FLOSS
    - Not easy for proprietary either

MITRE 2003 Report

"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. It would also 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. Finally, it would 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, http://www.egovos.org/

Acronyms

- COTS: Commercial Off-the-Shelf (either proprietary or FLOSS)
- DoD: Department of Defense
- HP: Hewlitt-Packard Corporation
- JTA: Joint Technical Architecture (list of standards for the DoD); being renamed to DISR
- OSDL: Open Source Development Labs
- FLOSS: Open Source Software
- RFP: Request for Proposal
- RH: Red Hat, Inc.
- U.S.: United States

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**