Sourcing Security

five arguments in favor of commercial security solutions

A Certicom White Paper
November 2005

certicom™

Part I: An ongoing debate

In a previous Certicom white paper (*Sum Total: Determining the true cost of development for embedded security*) we identified many of the hidden—and sometimes startling—costs associated with the use of non-commercial open source code for embedded security applications.

While it is essential for developers to understand those costs, the truth is they represent just one set of considerations when choosing between open source code and a commercial security solution. With respect to 'sourcing security', there is a great deal to think about before one gets to the bottom line.

Practically speaking, there are five critical factors:

1. *Vulnerability*—security strengths and weaknesses
2. *Standards compliance*—how each keeps up with evolving expectations
3. *Liability mitigation*—how each affects your legal and competitive exposure
4. *Support*—what's available to aid your development and maintenance efforts
5. *Total costs of development and ownership*—the lifecycle price tag for each

This paper offers a comparison of open source and commercial code in the context of the above. It argues that competitive necessity tends to compel commercial solutions to meet the full range of development and security requirements—and typically makes them easier to work with—than the open source alternative.

certicom™

Part II: The Five Determinants

1: Vulnerability

It may seem strangely obvious to point out that security is a fundamental concern when evaluating the options for developing a *security application*. But the reality is that preconceptions about the convenience, flexibility and cost of open source code sometimes shift focus away from this most critical aspect.

And critical it is. The CERT Coordination Center of the CarnegieMellon Software Engineering Institute tracks statistics on reported vulnerabilities and security alerts throughout the computing domain: its figures show a general increase in each of these since 1995. By the end of the second quarter of 2005, for example, 76 percent of the previous year's total vulnerabilities had already been reported.[1]

Even if one considers that part of this increase may be due to the proliferation of available technologies (i.e. the ever-greater number of systems that could potentially become vulnerable), it seems clear that the general 'threat level' is rising.

In that context, it is important to appreciate that the very openness of open code makes it more vulnerable than commercial alternatives. Hackers have the same free access to the source code as legitimate developers, and hence have ample opportunity to discover ways of exploiting its weaknesses.

At the same time, because open source code is non-specialized (it is by nature multipurpose), developers must often sift through massive quantities of irrelevant or unnecessary code to find the pieces that suit their precise needs. This can be a long and tedious process that ultimately stretches out an application's time to market—something that is rarely desirable from a competitive point of view. While time can be saved by building an application without scrupulously weeding out extraneous code, doing so yields a 'fatter', less efficient end product. More seriously, it creates a security risk: unused portions of code lying dormant in an application are prime targets for viruses and other forms of attack.

Commercial solutions, on the other hand, are significantly less exposed, affording a greater degree of basic protection. On top of that, commercial solution providers are under intense competitive pressures to ensure that their offerings are as vulnerability-resistant as possible: a product that earns a reputation for weakness will die a quick death in the marketplace. The efforts commercial solution providers devote to staying on top of new threats and improving—continuously—the resilience of their offerings deliver a degree of 'vulnerability insurance' that is absent from open source alternatives.

Maintaining full freedom

Some developers may counter that the freedom and flexibility of having access to source code outweighs all other considerations. While in fact it is increasingly common for commercial solutions to provide licensed, authorized users with access to source code to create truly customized

[1] Total vulnerabilities in 2004: 3,780. Total vulnerabilities by the end of Q2 2005: 2,874. (*CERT/CCStatistics 1988 – 2005, www.cert.org/stats*)

applications, the truth is that this should be done with extreme caution in any circumstance. Most developers—by their own admission—are not security experts, and altering quality-assured source code is an excellent way to introduce bugs into an application.

Regarding inherent security, then, the arguments weigh heavily in favor of commercial code. Developers should be cautioned, however, not to put blind or passive faith in any solution. Even the strongest offerings require proactive integration and maintenance plans to ensure that their full security potential is realized. Recommendations for such plans—along with some additional development guidelines—are offered in Part III of this paper under the chapter heading: *Taking the proactive approach.*

2: Standards compliance

As businesses and governments continue to increase their reliance on electronic information and exchanges—and as concerns about personal privacy and public safety continue to intensify—solution providers are under ever-mounting pressure to comply and keep up to date with stringent, technically demanding security standards.

For commercial code providers who sell to solution developers, adhering to those standards is a primary business driver. Not so for the open source community. (This isn't to suggest that the open source community has no interest in established or emerging standards; it's simply to say that for commercial code providers compliance is essentially imperative, not elective.)

FIPS 140-2, the U.S. Federal Information Processing Standard established by the National Institute of Standards and Technology (NIST), is a commonly cited case in point. Throughout North America, solutions adopted by government departments are required to meet the specifications of FIPS 140-2. Developers have two choices: they can try to create their own FIPS-validated code from an open source base, or they can acquire pre-validated commercial code.

The latter is usually regarded as being the more expensive option, as open source code can be acquired for 'free'. And yet it can take many months and cost thousands of dollars to develop an application that qualifies for FIPS validation. To date, no open source algorithm or platform has achieved FIPS validation—and indeed, the difficulty of doing so is demonstrated by the fact that not even all commercial code offerings are certified. Yet there are commercially available security-specific code solutions that include FIPS-approved algorithms and which are FIPS-validated on more than one platform.

Quite often, customer agreements obligate code vendors to support a prescribed set of standards for a given period. The open source community, on the other hand, has greater freedom to switch spontaneously from support for one to support for another—with the consequence that code users who have developed products based on one standard are

NSA Suite B: You can't sell into government without it.

The National Security Agency (NSA) recently established cryptographic specifications to protect classified and unclassified information within the United States Government. These specifications, known as Suite B, deal with unclassified data, as well as classified and sensitive command-and-control information. It includes the following protocols: ECDH and ECMQV for key transport and agreement; ECDSA for digital signatures; AES for symmetric encryption; and SHA-2 for hashing.

Vendors who want to sell into the government must ensure that, for a given application, their solutions include these approved algorithms. While commercial solutions built for security may include the Suite B requirements, developers working with open source code must somehow acquire and build them in on their own: an immensely complex and challenging process. But there's no real choice in the matter. Without Suite B, the door is closed.

suddenly faced with maintaining compliance entirely on their own. New and changing standards—Transport Layer Security (TLS), for example—require the development of wholly new ciphersuites; while it is incumbent on commercial solution providers to keep up, open-source users are essentially left to their own devices or the whims of the community they belong to.

Debunking a popular misconception

Open source code is indeed subject to licensing requirements: it's not an 'open bar'. Under the General Public License agreement, manufacturers may be required to share application-specific code modifications with the open-source community. Developers frequently try to avoid this by 'wrapping' the code in software or keeping changes secret. But this complicates the whole question of intellectual property ownership and increases the liability of open source users.

3: Liability mitigation

Patent indemnification is an issue of great concern for solution developers today, and understandably so. The financial repercussions of violating patent protections—whether intentionally or accidentally—can be crippling.

In the case of commercial security solutions, the solution provider is legally liable for any patent infringement or contravention of intellectual property rights. It is up to that company to ensure it has its proverbial 'ducks in a row': that the chain of intellectual property is well-defined and that any third-party components in the solution are properly licensed.

With open source code, that liability shifts to the developer, and a legal review is often necessary to ensure that no patent-protected elements are being used unacknowledged. If third-party intellectual property is included in the code, the onus is on the developer to negotiate and follow through with licensing.

As well, commercial code vendors generally provide some guarantee of performance as well as bug-fixing and ongoing maintenance services. Certainly, the open source community devotes considerable effort to bug-fixing, but not in any prescribed (and therefore commercially reliable) way. And yet open source code may in fact be more vulnerable to bugs due to its multi-contributor development. The content of commercial code, on the other hand, is controlled by the vendor.

4: Support

Because commercial code vendors are business enterprises, their success depends on upholding a certain level of customer satisfaction. Consequently, they offer a range of training, documentation, maintenance, upgrade and professional services to support their customers' solutions. These supports are not quite so readily available to users of open source code. The open source community is simply not structured to deliver stable, customer-centric support over the long term.

Yet—to revisit some earlier remarks—even with the most supportive commercial solution there is a requirement for the customer to establish a proactive integration and maintenance plan that accounts for contingencies such as patch requirements and unexpected upgrades. The realm of security is particularly fluid; change can come quickly, and preparation is essential to making a decisive, effective response.

This, of course, is true whether one chooses open source or commercial code. However, those who work with open source code tend to take on much more responsibility in this regard; quite often, being proactive when dealing with a commercial solution involves maintaining good communication with the vendor. Doing so allows developers to meet customer needs by adding new features and functions without having to concern themselves with adhering to the GPL or going it alone.

Again, Part III of this paper will review some of these considerations in greater detail.

5: Total costs of development and ownership

Taken together from a lifecycle perspective, all of the abovementioned factors (vulnerability, standards compliance, liability and support) expose greater-than-expected costs of development and ownership with respect to open source code, and by the same token reveal the true value of some of the perceived 'expense' associated with commercial solutions.

Undeniably, open source code is 'free' in the sense that it requires no up-front fees for developers to begin using it. And as has been discussed, it affords completely unrestricted access to source code. But in the first instance, as has been shown, no fees does not mean no cost with respect to licensing intellectual property and related issues. And in the second, there is a definite security cost that derives from the very fact that anyone can look into the very heart of the code being used for a particular application.

There is also an engineering cost associated with the tailoring of all-purpose open code to suit a specific application. This, in fact, can be significant when optimizing and integrating generic open-source components within an embedded environment where security is the goal. Commercial products, alternatively, are typically purpose-built and designed for easy integration.

Included with this paper, as an appendix, is a TCD/TCO calculator developed for the earlier Certicom white paper, *Sum Total: Determining the true cost of development for embedded security*. As that tool reveals, there are development-cost savings to be realized when using open source code for well-supported open source projects. However, for developers aiming to get new products to market expeditiously, commercial code offers irrefutable practical advantages.

The licensing landscape

There are in fact many approaches to licensing, GPL (mentioned above) being just one. The common types include:

General Public Licenses (GPLs)—These guarantee user freedom to distribute copies of software, receive source code, modify software or re-use it in other programs.

Berkeley Software Distribution Licenses (BSDs)—These have even fewer restrictions than GPLs, allowing for largely unconditional sharing, copying and reuse.

Mozilla Public Licenses (MPLs)—These serve as the main licenses for the Mozilla Application Suite, and also as the basis of the Common Development and Distribution License for OpenSolaris; they require that source code copied or changed under an MPL must stay under the terms of the MPL.

certicom™

Part III: Taking the proactive approach

To speak frankly, many developers with primary expertise outside of the security domain often fail to grasp the risks associated with deploying a security component in their applications. Security demands vigilance—not only in execution, but also in design. Security software cannot be treated in exactly the same way as other third-party components. When it is, developers often end up contradicting their companies' own published security policies. And unfortunately, this conflict doesn't become apparent until a significant vulnerability arises and they find themselves unable to react in a timely manner.

The following recommendations will help any developer working with a commercial security solution avoid such consequences and get the most out of the code they've chosen. Well-designed security can actually reduce the cost of upgrades and accelerate integration time.

Plan for upgrades

To begin, a good integration plan is one that deals not only with initial development and integration but also provides a reliable upgrade path as well. For security components, such plans must consider two important scenarios:

1. *Planned upgrades*—through which the security application is upgraded routinely as part of the regular release cycle. This requires an appropriate allocation of time in the ongoing project plan.

2. *Unplanned upgrades*—essentially, emergency patch releases. Time is typically of the essence in these cases because developers and users are aware that a vulnerability exists and may be concerned about the risk to the application.

Many developers choose to implement upgrades only within their planned release cycles. Yet this leaves them unprepared to deal with a required upgrade in the event a vulnerability is identified. And of course a product that is long out of maintenance may not be eligible for the most recent upgrade should such an event arise: at some point, old code releases do become unsupportable.

Keep security separate

By minimizing the 'integration' of security within an application, developers have more freedom to maintain it separately and ensure its currency. Some strategies for achieving this include:

- **Minimizing application exposure to security interfaces**—for example, by isolating all callback definitions within a single library or module. Consider placing security within a single shared or static library and limiting the number of APIs exported from this library. (The benefit of isolating security this way is that it can be upgraded independently of the application. Using a static library probably requires a re-release of the application, but the changes to security are unlikely to require significant reworking of the code.)

- **Maintaining weak coupling between the application and security**—for example, by limiting the scope of security-related header files and data types within the application. If security data types need to be exported, consider doing

so in a manner that makes their content opaque: this will ensure that changes to the data structures don't affect the application. Isolating security header files and data types limits the number of features available to the application. This might be any great concern for a single application, but when multiple applications use the same security service, they may need to conform to a consistent security policy.

- **Enforcing information hiding between the application and the security functions.** This becomes particularly relevant whenever it is necessary to create a global security policy or to globally change the memory allocation used by security.

Part IV: Final thoughts

Because security is so specialized (and so critical) a function, it requires extra consideration and planning. Due to its complexities and the rigors of the standards that must be upheld to reach certain markets, using open source code can be prohibitively expensive and time-consuming—not to mention excessively demanding upon engineers whose core competencies lie elsewhere. Commercial security solutions tend to be less vulnerable, easier to use, and tidier with respect to ensuring compliance with reigning standards and patent protections. All of these factors, plus the matter of long-term support, rationalize much of the cost associated with choosing commercial code over open source.

Once a commercial option is selected, it remains incumbent upon the developer to devote some clear thinking to the long-term maintenance of the security component being developed. Perhaps more so than with any other element, security requires constant and proactive attention.

Establishing a clear integration plan and minimizing the interconnections between security and the main application being developed are key ways of ensuring that the chosen solution can be depended upon to deliver the greatest results over the long term.

certicom™

Appendix

To project the total cost of development for a given project, estimated costs for each of the following should be calculated:

Consideration	Value
RESEARCH AND PLANNING COSTS	
All preparation, specification, code-sifting, etc. MEASURE Time invested (developer person days)	
SECURITY FAMILIARIZATION COSTS	
Development team security training, including cost of creating a customized training course. MEASURE Time invested (developer person days)	
CODE REVIEW COSTS	
All costs for reviewing code. MEASURE Time invested (developer person days)	
CODE DEVELOPMENT COSTS	
All costs for building, adding to optimizing, and adapting code to suit product and platform. MEASURE Time invested (developer person days)	
TROUBLESHOOTING AND BUG-FIXING COSTS	
All costs for ongoing troubleshooting and code-related bug fixing. MEASURE Time invested (developer person days)	
DOCUMENTATION COSTS	
If documentation is required for security code, include estimated time for document-creation. MEASURE Time invested (developer person days)	
Total Developer Person Days	
COST PER DEVELOPER PER DAY (Average annual salary of developers + benefits and overhead expenses) / working days per year	

certicom™

Consideration	Value
STANDARDS-COMPLIANCE COSTS	
All costs for certifying code as standards-compliant. MEASURE Dollar value (certification fees) + developer costs (developer-person days * cost per developer)	
LEGAL COSTS	
Patent Protection: any investment in open source requires a legal review to rule out possible patent infringements MEASURE Dollar value (Legal fees)	
A) Total Development Effort = (Developer-person days * Cost per Developer per day) + Standard-Compliance costs + Legal costs	
SOFTWARE LICENSE	
Cost of software licenses.	
ROYALTIES	
Percentage based on product revenue.	
MAINTENANCE AND SUPPORT	
Cost of all ongoing support and maintenance.	
B) Total Software Costs = Software license + Royalties + Support	
TCD = (A) + (B)	

certicom™

Certicom White Papers

To read other Certicom white papers, visit www.certicom.com/whitepapers.

The Inside Story

Many Happy Returns: The ROI of Embedded Security

Wireless Security Inside Out (authored by Texas Instruments and Certicom)

Welcome to the Real World: Embedded Security in Action

Sum Total: Determining the True Cost of Security

The Elliptic Curve Cryptosystem for Smart Cards

Elliptic Curve DSA (ECDSA): An Enhanced DSA

Formal Security Proofs for a Signature Scheme with Partial Message Recovery

Postal Revenue Collection in the Digital Age

An Elliptic Curve Cryptography Primer

ECC in Action: Real World Applications of Elliptic Curve Cryptography

Using ECC for Enhanced Embedded Security: Financial Advantages of ECC over RSA or Diffie-Hellman

Good Things Come in Small Packages: Certicom Security Architecture for Mobility

Meeting Government Security Requirements: The Difference Between Selling to the Government and Not

FAQ: The National Security Agency's ECC License Agreement with Certicom Corp.

Exercise Your Rights: DRM and security

certicom™

About Certicom

Certicom protects the value of your content, software and devices with government-approved security. Adopted by the National Security Agency (NSA) for classified and sensitive but unclassified government communications, Elliptic Curve Cryptography (ECC) provides the most security per bit of any known public-key scheme. As the undisputed leader in ECC, Certicom security offerings are currently licensed to more than 300 customers including General Dynamics, Motorola, Oracle, Research In Motion and Unisys. Founded in 1985, Certicom's corporate offices are in Mississauga, ON, Canada with worldwide sales headquarters in Reston, VA and offices in the US, Canada and Europe. Visit www.certicom.com

Contact Certicom

Corporate Headquarters

5520 Explorer Drive, 4th Floor
Mississauga, Ontario
L4W 5L1
Tel: +1-905-507-4220
Fax: +1-905-507-4230
E-mail: info@certicom.com

Sales Offices

Worldwide Sales Headquarters
1800 Alexander Bell Dr., Suite 400
Reston, Virginia 20190
Tel: 703-234-2357
Fax: 703-234-2356
E-mail: sales@certicom.com

Ottawa
84 Hines Road, Suite 210
Ottawa, Ontario
K2K 3G3
Tel: 613-254-9270
Fax: 613-254-9275

U.S. Western Regional Office
393 Vintage Park Drive, Suite 260
Foster City, CA 94404
Tel: 650-655-3950
Fax: 650-655-3951
E-mail: sales@certicom.com

Europe
Golden Cross House
8 Duncannon Street
London WC2N 4JF UK
Tel: +44 20 7484 5025
Fax: +44 (0)870 7606778

Engelska Huset
Trappv 9
13242 Saltsjo-Boo
SWEDEN
Tel: +46 8 747 17 41
Mobile: +46 70 712 41 61
Fax: +46 708 74 41 61

www.certicom.com