ON THE ARCHITECTURE OF SECURE SOFTWARE DEFINED RADIOS

John A. Davidson
SAIC
San Diego, CA

ABSTRACT

This paper examines the rationale, properties and shortcomings of the prevailing "red-black" architecture for secure software defined radios (SDR). To address shortcomings, a computer security process is formalized that leads to a provably secure SDR architectural framework. Although an example of a secure architecture is discussed, the focus is to explore an architectural design strategy to achieve guaranteed secure operation. The result is more of an architectural framework with remarkable potential relative to previous approaches, including enabling deterministic development, operational friendliness, high performance, and affordability through strategic use of hardware to strengthen and simplify the enforcement of the security policy instead of simply trusting software.

INTRODUCTION

An SDR is a computer system, and we know how to design a computer system based on requirements. We have that game plan down, like playing checkers. When we design a secure computer system the rules are different. Security is not simply another requirement; it is another game, more complex, like chess. Worse yet, the game board is deceptively similar. Conventional system engineering methodologies fail to produce results that meet security expectations because the sufficient set of security requirements is often vague or missing. This lack of clear security requirements leads to nondeterministic development, more like trial and error than the deterministic development process planned (and costed). To reveal reasons for this problem we must first lay some groundwork.

First we establish an understanding of just what an SDR is. An SDR is a computer, or in recent years, more typically a system of co-located networked computers that implement a function that happens to be a two-way radio. A secure SDR often implements a network radio, meaning that there are a number of radios cooperating in a network routing data among themselves. Each node in the network performs radio and networking functions, as well as providing land-line user interfaces. This secure SDR is also an application on a secure computer that protects, encrypts and decrypts the payload traffic as required. As shown in Figure 1, some operations can be applied to encrypted traffic, called ciphertext (CT), but other processes require access to decrypted traffic, known as plaintext (PT). An encryption/decryption device, known as a cryptographic device historically called a Key Generator (KG), performs the PT/CT conversions. In an SDR, these functions are synthesized by computers, making it convenient to define arbitrary radio signals, networking protocols and user interfaces with software programs. The collection of software programs that make an SDR emulate a specific radio is collectively referred to as waveform software. The programs that implement common waveform software commands are infrastructure software.

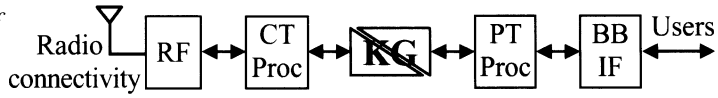

Figure 1. Typical Secure SDR Communications Channel Functions.

INFORMATION ASSURANCE VIEWPOINT

When cryptographic algorithms were implemented in hardware boxes, TEMPEST techniques were used to separate the PT side from the CT side from unintended leakage of PT (red) signals onto CT (black) cabling via conducted or radiated emanations. This tradition influenced early tendencies to organize radio functions by red and black sides [1], [2] illustrated in Figure 2. The crypto function designated KG protects information by substituting a key-dependent secret code for the clear text traffic. Based on that, the prevailing security architecture strategy [3] separates an SDR into two security domains, *red* and *black*. The control of the overall radio must be hosted in one of those domains. In practice, almost all control of the radio is done with unclassified parameters (e.g. channel assignments, protocol selections, RF power levels). The approach of hosting control on the black side shown in Figure 3 facilitates control of unclassified radio assets. The control must pass the red-black boundary, but in the direction allowable by secrecy if it is constrained by simple one-way flow. (Note: integrity is addressed later.) One problem with black-side control is that it places the control of the radio in the same domain that contains adversaries. De-

Figure 2. Generalized red-black architecture framework for secure SDRs.

978-1-4244-2677-5/08/$25.00 ©2008 IEEE

fense in depth [4] has been suggested [5] as the strategy for high assurance protection. This is extremely hard to accomplish without encountering the Cascading Problem [6] (cascading many three-foot hurdles fails to achieve the effect of a six-foot hurdle.) Too often, high-assurance is mistakenly [7] attributed to cascading a number of low-assurance mechanisms. The alternative for a red-black radio is to host the control with plaintext processing from the red-side as in Figure 4. Drawbacks of trying to protect control by moving it to the red side are more obscure.

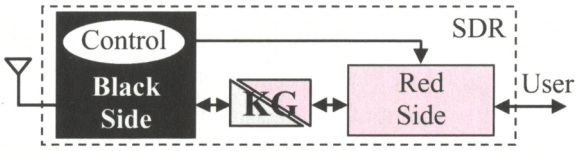

Figure 3. Black-Side Control.

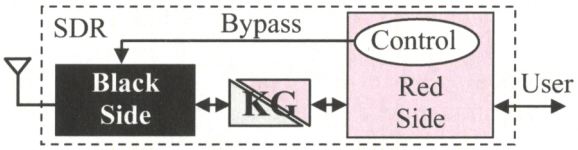

Figure 4. Red-Side Control.

How Control Bypass Becomes such a "Necessary Evil"

Red-side control locates control across the red-black boundary from the black-side assets it controls, so control flow goes in the wrong direction to preserve secrecy [8] of the red side. Control bypass is often expected to provide an acceptable way to secure such wrong-way flow. Since packet protocols have become pervasive, it is customary to bypass header address and service information between the PT packets to CT packets. Header bypass is not a rigorously secure solution, but it has become customary to view the vulnerability as "manageable" [9] because it is impractical to fix. The technical basis for its mitigation involves specialized schemes like state filters that discourage exploitation of this leak by modulating packet headers (not useful for control bypass). Justification of control bypass as an extension of header bypass is not a satisfying stance because external factors drives header bypass. Justification of control bypass as an inevitable covert channel is not supportable either. The term *covert channel* properly applies only to channels not intended for *any* data communication. [10] Bypass is "evil" because it amounts to spillage by what is known technically as a *non-secure state*.

The proliferation of bypass on SDR flow paths leads to examination of whether the red side – black side paradigm is doing the right job. It is not structured to separate unequal security levels. The bypass it relies on is contrary to the separation mechanisms that isolate the red side from the black side. Bypass not only undermines costly separation mechanisms, but will also be shown to lead to superficial bypass guards. Nagging concerns about the purity of the red-black paradigm are rooted in a lack of rigor about what *red* and *black* domains really meant since it was never based on a consensus of what *secure behavior* means, customarily expressed as a security policy. The paradigm originated [11] to address TEMPEST separation in analog circuits, but it is less useful for today's digital computers.

Complication of Dealing with Communications Plans

If coordinated control of the radio's classification-diverse channels were not needed, rigorous hardware channel separation, loosely known as Multiple Single Levels of Security (MSLS), might work. However channel set-up and control parameters must be derived from external communications plans, requiring additional radio planning/control interfaces. A communication plan is often conceived as a classified (secret) objective. Before using computers, unclassified elements of the plan were deduced by humans from the secret objective and used to generate unclassified radio set-up parameters. For an SDR these functions can be hosted (potentially externally) in a planner workstation. To avoid the inconvenience and cost of making this planner workstation trustworthy, it is tempting to regard it as "system high" secret. System high is a formal COMPUSEC term that results in all the parameters contained in the operating environment (OE), including unclassified, being re-classified at the high water mark, typically secret. This system high strategy produces a false sense of security by obscuring the fact that the planner must be trusted to separate security levels anyway, but it also adds a superficial but expensive trusted guard to the radio to restore the separation of unclassified and secret parameters. As illustrated by the upper arc in Figure 5, the radio must get the unclassified parameters to the black side. A control bypass channel in the radio supports this, with a guard preventing secret parameters from passing, as suggested by the leak in Figure 5. Because this bypass guard has no criteria on which to base a decision except to

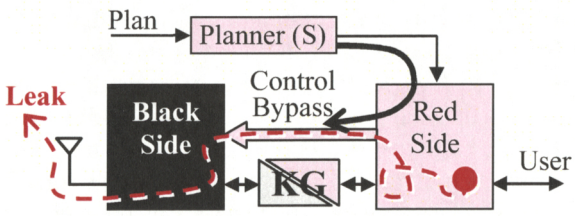

Figure 5. Risk imposed by bypass guards that rely on criteria originating in system high sources is bounded only by bandwidth constraints.

trust some format or content property produced by the planner, the planner might as well have been acknowl-

edged as trusted at the outset and given a direct interface to the black side, thus eliminating the high assurance guard.

WHAT CAN BE DONE TO IMPROVE

At the outset of this paper, it was suggested that the sufficient set of security requirements to guarantee secure behavior is critical to program success. It is the author's experience that a sufficient set of security requirements is almost never found in system performance requirements. But without them, the development process lacks criterion for design process termination and corrective design feedback is vague or wrong. This leads to a development that proceeds open-loop, in a design-by-trial-and-error manner. The schedule, cost and risks are unbounded for such a design process. Typical symptoms for this are chronic security architecture issues as cited recently for some SDR projects [12]. A set of security requirements sufficient to guarantee secure behavior for an SDR is critical to develop a secure architecture. Such a set will be referred to as a *sufficient set*.

It is widely acknowledged, at least among "old-guard" COMPUSEC subject matter experts, that there is one documented [13] process that can produce the sufficient set of security requirements. The process is analytical; it derives a Security Policy Model, and deduces this sufficient set of security requirements from that model. Defining the right security policy model is critical to that process. At a high level, the author generally follows five steps outlined here for the process diagrammed in Figure 6.

1. Define secure behavior for SDR (SDR Security Policy).
2. Create a correct data flow model of the SDR Security Policy, a Security Policy Model (also called Policy Model).
3. Create a data flow model of the SDR system that satisfies operational requirements. For this, a System Model expressed in the form of Data Flow Diagrams [14] captures the security-critical features, arguably not visible with UML because it hides flow/process distinction.
4. Converge the System Model to the Policy Model (Secure SDR System Model).
5. Verify that the implementation corresponds to the Secure SDR System Model.

The Security Policy expresses the fundamental definition of secure SDR behavior regardless of the approach. The

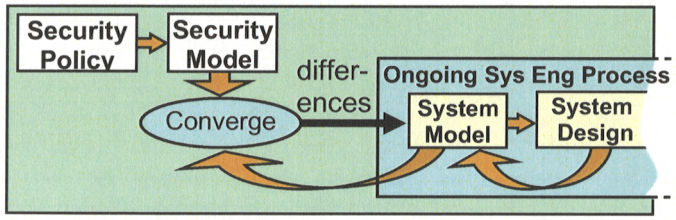

Figure 6. *Secure* System Engineering Process Flow Diagram.

Security Policy typically represents a consensus of all stakeholders at an operational level. Steps 1 and 2 depend on the system data environment, not the implementation.

The Policy Model is a behavior model, usually expressed mathematically (i.e. formal). The model precisely exhibits the security properties of the policy, but expressed in terms that can be logically validated. This is helpful because it is surprisingly difficult to validate *security* without some math. A case in point is the Red/Black model underlying the security architecture discussed earlier in this paper. The properties of that model are not consistent with each other. Consider that both secret traffic and control are hosted on the red side, loosely presumed secret; so secret control objects must flow to the unclassified black side, contradicting a presumed premise. That policy lacks internal consistency, and will need to be bypassed to make a radio work. Formalizing the model (math-based model) eliminates vague presumptions and assures internal consistency. A Policy Model also serves as a common link between the English expression of the Security Policy and the technical properties of a system model by integrating properties of both. It is also surprisingly difficult to establish whether or not a Security Policy is enforced by an implementation without using a model.

The policy model is a technical elaboration of only the definition of secure behavior, and avoids dependencies on design approach to achieve the operational requirements. While there are a variety of approaches that could be taken to produce a policy model for a given policy, there are substantial risks and technical pitfalls too. Informal approaches often produce models with internal inconsistencies. Formal models can be difficult to converge to a useful system model. Because policy modeling can go wrong [15], discretion is advised. The author believes that flow-type security policy modeling is among the least error-prone viewpoints since flow and domains correspond well to familiar architectural notions of real systems.

AN SDR SECURITY POLICY

An exemplary security policy that serves as a practical top-level definition of secure SDR behavior combines two essential dimensions of information assurance (IA): confidentiality and integrity. The objective is to strike an appropriate balance between a policy at a high enough level that it does not limit implementation options and approaches, while being specific enough to be complete in coverage of at least the minimal essential IA objectives.

Military SDR solutions are generally obligated to enforce the National Security Policy expressed by the Executive Order 12958 (EO) [16]. The EO establishes security levels

for data, clearance levels for people and mandatory access controls that prevent disclosure of the classified data to persons not appropriately cleared. Also, it is considered prudent by many stakeholders to protect the integrity of SDR control plane from corruption by elements in the traffic domain, a broader community with greater exposure to adversaries. Formal IA technology is available to guarantee near absolute enforcement of confidentiality and integrity. While availability may be no less important, it is acknowledged that there is no widely accepted IA technology to guarantee it with formal certainty, and so best-effort methods must be acceptable for this.

Derived requirements for mechanisms like discretionary access controls and key management can be inferred from the top level policy. They can be addressed as lower level design-specific derived requirements and are beyond the scope of this paper. The provisions proposed as an acceptable definition for secure behavior for this case of an SDR, can be condensed into three assertions as follows:

A1. System Absolutely Prevents Disclosure of Classified Data
A2. System Absolutely Prevents Corruption of Control by User Traffic
A3. System Substantially Prevents Service Denial with Best Efforts

These policy assertions (as is typical of security policy assertions amount to negative requirements and therefore are useless as design or test requirements. These need to be transformed into a form that can guide a design process. The security policy model process performs this transformation from policy assertions to policy model. It produces a model with minimal essential properties that the system must exhibit to enforce the security policy with mathematical precision. That means the model embodies the least set of security requirements that are sufficient to enforce the policy with certainty. That is a noteworthy guarantee. Such precise solutions raise concerns about cost effectiveness when so often the proverbial "80% solution" is expected to be more cost effective. Unfortunately, no known IA process can reliably achieve an 80% solution. Trial and error may eventually produce 80% solutions, but usually at a much higher cost (unbounded) than the 100% solution sought by the IA process.

PICKING A "LOW HANGING" SDR SECURITY POLICY MODEL FRUIT

Since there is no widely accepted IA technology to formally address assertion A3, service denial, we acknowledge that must be addressed, but with best effort means. So we set this admittedly serious vulnerability aside for another paper. Only the first two secrecy-critical assertions are formally modeled. The first assertion implements the hierarchical security classifications and access rules expressed by the EO. A substantial and widely vetted body of work already exists that applies this to a computing environment. The seminal work for this is the Bell-LaPadula (BLP) [8] formal security policy model. It has been formally proven to precisely model these hierarchical secrecy levels of the EO and interpreted briefly in common laymen's terms for computer operations (reads and writes to objects) by what has come to be known as the *Multics Interpretation*. So if we use BLP, we can include the existing proof as our supporting evidence without much risk of it being wrong. BLP can be viewed as a flow type model that establishes a domain for each security level, and prohibits flow from higher to lower domains. For the simple case of secret and unclassified, the BLP would model this as two domains, S and U, with unidirectional flow only from U to S. Biba [17] extended BLP to address integrity levels for cases where integrity can be represented by hierarchical levels analogous to security classifications. For integrity, Biba showed flow is inverted from low to high. These two models can be combined [18] with Cartesian products of their domains while preserving the flow constraints over the product. These formal security policy models are precisely expressed mathematically with set theory, but the math is not presented in this paper. Instead, Figure 7 presents an example graphic visualization of the underlying mathematical model discussed. It is well established that the IA properties of these models are equivalent to those of the policy.

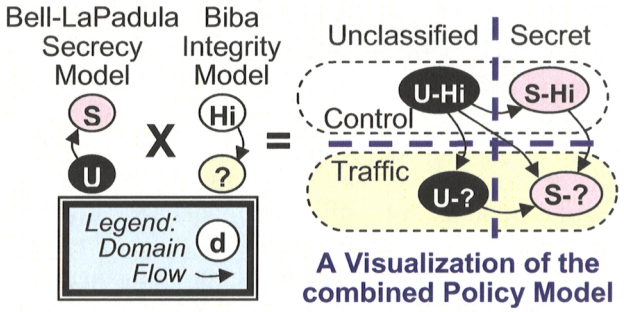

Figure 7. BLP and Biba models are combined to derive the necessary and sufficient domains and allowable flow.

There are at least two practical high-assurance approaches to enforce this policy. One is to use an MLS operating system (OS). A successful and relevant worked example of this approach, the BLACKER system, implements similar top secret through unclassified MLS requirements (except RF) with a similar security policy. As noted, [19] its Class A1 security "requirements are satisfied by a single logical processor managing the cryptographic component, a revolutionary approach." For integrity requirements [20] BLACKER leveraged the fact that "this kernel defines and

enforces an integrity policy [Biba]." They ended up "selecting GEMSOS, an off-the-shelf kernel from Gemini Computers" [19], for which current versions continue to be available as a COTS product [21].

For SDRs with multiple processors, an alternative high assurance approach offers the potential to enforce this policy model without an MLS OS. An MLS SDR can be composed using a single level processor for each domain and hardware one-way links in an Asymmetric Isolation [22] architecture. This requires more ingenuity to arrange the processors and one-way links to have a one-to-one correspondence to the domains and arcs, respectively, of the policy model, formalizing the system model. Formal correspondence of the system model to the security model amounts to establishing isomorphism between the nodes and directed arcs of a partially ordered lattice representation [23] of the policy model to corresponding physical components (the nodes and unidirectional links) of the system model (node mapped to machine; directed arc mapped to one-way link).

THE RESULTING ALTERNATE SDR ARCHITECTURE FRAMEWORK

It is not always possible to arrange processors and one-way links in a manner that has this lattice isomorphism property while substantially satisfying essential system operational requirements. Fortunately for an SDR, the process and data flow framework shown in Figure 8 accomplishes this. This organizes the traffic channel functions of Figure 1 and control functions of Figure 3, by classification. It differs from the red–black paradigm in two noteworthy ways. Instead of a *red* domain, it has a separate PT classified traffic domain for each classification level. Instead of a *black* domain, it has separate CT (unclassified) traffic and unclassified control domains distinguished by the heavy horizontal broken line. Also above that line to the right is a domain for secret control, should that be needed. If it is, *something* is forced into multilevel secure (MLS) mode by the logic of the Computer Security–Intermediate Value Theorem [24] and responsibility for that must be allocated. A practical solution is to consolidate this MLS responsibility[1] in the Planner, potentially outside the radio. This avoids the questionable expectation that the classification separation lost by avoiding the cost of trusting the planer can be recovered using a guard in the radio, itself MLS and costly, or using an operationally unfriendly manual downgrade, also costly and MLS. Using multiple processors sharing an HMI to implement the planner holds some potential for avoiding an MLS OS if the sources of planning data can be MLS.

Isomorphism between this framework and the policy model is straightforward except for the crypto and the switch (Sw). The crypto can be shown to reliably stop data flow by transforming the information to cover its intelligibility. The switch, enabling economy of sharing of baseband interface units, routes flow between incompatible domains. It must be shown robustly implemented and controlled to preserve the domains and flow of the policy model. This requires high assurance of the integrity (i.e. *mandatory integrity*) of the switch control, thus avoiding commands authenticated in untrusted sources. To avoid an MLS OS the commands from untrusted sources could be treated as requests if adequately validated with hardware. A promising approach compares crypto key labels with user port classification discrete signals via trusted paths (e.g. connector backshell strapping at the port interface.).

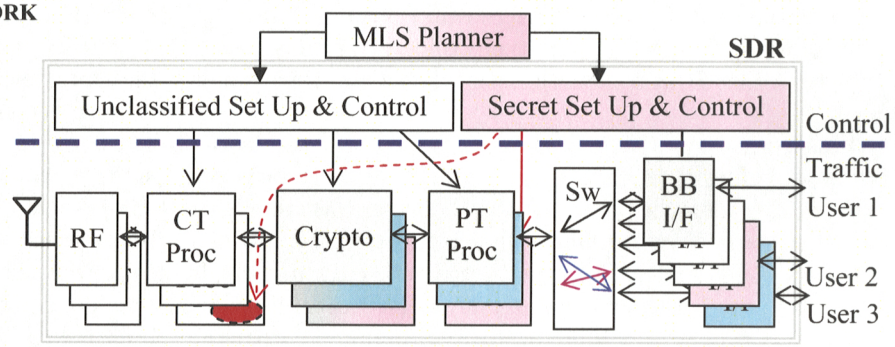

Figure 8. Alternate SDR Security Architecture Framework.

A PEEK AT THE DEVIL IN THE DETAILS

It is remarkable that simply by isolating the boxes and enforcing the unidirectional flow between them consistent with the policy model (illustrated in Figure 8), security policy enforcement can be guaranteed. The power of this framework stems from its ability to simplify the proof of a blanket high-assurance policy enforcement over all processes in the radio, substantially relieving the software of the security critical responsibility for *being secure*. That power is eroded each time it becomes necessary to punch a hole in that blanket to accommodate a "corner case." Corner cases are exceptions that can arise where processes need the privilege to violate these blanket mechanisms to do their job. When processes are given this privilege, they must be individually shown to be trustworthy to enforce the policy on themselves. They are known as trusted processes and are expensive to assure and vulnerable. If they

[1] Application of the Computer Security Intermediate Value Theorem also proves that the radio as a whole operates in MLS mode if a user provides classified traffic, since unintended users of the RF can be uncleared, making RF unclassified. A practical solution is allocating that MLS responsibility to functions in the trusted OE of the crypto.

are software processes they must be hosted in a protected OE. Trusting software at high assurance levels also implies either exhaustive "branch and path analysis" becomes unreasonably expensive on more than a few lines of code, or just accepting the risk as another case of "managed" risk.[9] Furthermore, trusting software often requires protections like encrypting it for storage that potentially affect performance and boot-up or instantiation timeframes. This is why simply trusting all the radio's software is considered unreasonable.

Comprehensive treatment of strategies to address such corner cases is beyond the scope of this paper. However it is prudent to offer some insight into their potential complexity. The most common example in secure packet switched systems is header bypass that remains an exploitable exception to the assumption that the crypto passes no cleartext classified information. A Catch 22 situation arises when an unclassified header arrives in a packet containing secret userdata and so is technically MLS, but its source is an untrusted secret system high machine. Due to this inconsistency, the header information (such as packet addressee) must be bypassed to an unclassified radio router process and consequently downgraded. This downgrade is customarily allocated to the crypto since it typically has a trusted OE and it is necessarily involved in that flow path.

Other corner cases arise, such as that of passing a classified waveform parameter to a modem signal processor which consumes the parameter to produce unclassified RF. By the CS-IVT, something in that modem must be MLS. This is suggested in Figure 8 by the thin curved dashed line from the Secret Set-up & Control function to the oval in the CT process. A solution involves defining a domain boundary between classified and unclassified inside the modem and showing the boundary is trustworthy. Assurance may include properties of the software or the hardware that preclude unintended leakage. By showing that there is no leakage along the flow path of the parameter or across that boundary, the need for a guard or trusted filter between the modem and the secret control processes to enforce the policy is eliminated.

This case illustrates a strategy of "redistricting" domain boundaries sometimes applicable to other corner cases. It essentially creates an island of one domain within what was otherwise a hardware delineated domain. Redistricting works when a domain boundary is easy to enforce (e.g. exploiting hardware modularity) around functions within a component that is otherwise a member of a different domain. Although other creative strategies for trusted processes exist, the default blanket mechanisms are inexpensive and more easily assured. These corner cases require creative allocation of processes to architectural elements in such a way as to never breach the integrity of the blanket mechanisms. Each additional case addressed as a trusted process undermines the blanket mechanisms a little more.

Whether this framework should support instances of verbose middleware across domain boundaries is debatable. However, even middleware-supported interprocess communication (including CORBA) with the intent to transfer a parameter one-way across a domain boundary in the direction allowed can require intervention to avoid the need to trust software. Although CORBA has security features, these are not trustworthy with high assurance, so trustworthy mechanisms must be used. Intervention comes at a cost of some of the intended spontaneity and flexibility of the middleware. A potential strategy is to synthesize backflow handshaking used for "reliable" communication locally in the high domain. Since this undermines the reliability objectives of that handshaking, communication reliability may be recovered by other strategies such as empowering the destination domain processes to address exception processing autonomously as much as possible.

Unfortunately, interaction crossing domain boundaries *against* the directed arcs of the policy model (middleware-supported or otherwise) must inevitably be addressed with the aforementioned expensive trusted processes. This may involve decomposing the command structure to communicate only the core information with minimal entropy. The strategy would avoid communicating any handshaking that can be synthesized without flow in the wrong way across boundaries. The trusted process would need to be hosted in a protected OE established by a trusted OS or isolated in a separate processor. There may be no strategy to manage high risk in these cases except to limit the bandwidth of the leak, so emphasis should be placed on avoiding them.

CONCLUSION

Besides presenting a provably secure SDR architecture framework and addressing some implementation considerations, this paper illustrates other remarkable things. It illustrates an economical scheme that uses hardware to eliminate, or substantially eliminate trusted software for an MLS SDR. It shows how a little bit of MLS awareness in the planner preserves classification separation to relieve the substantial operational burden of a manual downgrade and the MLS OS in the radio, improving user friendliness. This architecture framework encourages early architectural problem resolution by focusing attention on the cases of trusted functions and pinpointing domain boundaries where they exist. This focuses design energy on the important challenges. No less remarkable is the recognition of how easily informal reasoning can be deceived, illustrating the value of formal COMPUSEC methods.

The development approach benefits substantially from this policy-driven strategy. It produces clear, relevant and verifiable security requirements. This provides corrective feedback and termination criteria for the development process making it manageable and finite. Because it produces the sufficient set of requirements needed to *be secure*, the artifacts represent proof the system is secure. This provides an alternative to expecting certifiers for architectural guidance. It empowers the developer to accept responsibility for the security architecture and to precisely address the certification with certainty early in the development. It enables developers to avoid the only "dumb" question in computer security: "Why isn't my system secure? Instead it empowers the developer to assert: "Here is precisely why my system is secure: …" Inverted developer–certifier roles strain relations and interfere with the essential collaborative spirit that streamlines certification.

ACKNOWLEDGEMENTS

Of the many who assisted with this paper, the author particularly wishes to acknowledge the gentle encouragement of Dr. David Elliot Bell to preserve the formal foundations of this methodology, and the ever-surprising insight of Scott Finkelstein whose contributions enabled this strategy to be practical enough to actually work. Mark Altier and Raymond Moberly's help organizing and wording some of the concepts substantially improved readability of this paper.

REFERENCES

[1] Hayes, Neli, "Software Communications Architecture," Feb.1, 2005, Object Management Group: http://www.omg.org/docs/omg/05-02-01.ppt

[2] Kurdziel, M.; Beane, J.; Fitton, J.J., Military, "An SCA security supplement compliant radio architecture, Communications Conference," 2005. IEEE MILCOM 2005, 17-20 Oct. 2005 Page(s): 2244 - 2250 Vol. 4.

[3] Bard, J. & Kovarik, V., Software Defined Radio: Software Communications Architecture, John Wiley & Sons, 2007.

[4] Stytz, M.R., "Considering Defense In Depth for Software Applications," Security and Privacy, IEEE Publication, Jan.-Feb. 2004, v. 2, Issue 1. pp. 72-75.

[5] Murotake, D. and Martin, A., "Updated System Threat and Requirements Analysis for High Assurance Software Defined Radios," SDR Forum Conference, Orange Co, CA., Nov.14-18, 2005.

[6] TG-005, NCSC, Trusted Network Interpretation, DoD TCSEC, DoD 5200.28-STD, Section C.3.2.

[7] Whittaker, J., *Defense in Depth* section in "Why secure applications are difficult to write" Security and Privacy, IEEE Publication, Mar-Apr 2003, Volume: 1, pp. 81-83.

[8] D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR-2997, The MITRE Corp, Bedford, MB, July 1975, (ESD-TR-75-306).

[9] "Managed risk," a laudable-sounding concept that "has been used to justify use of low- or medium-assurance components to secure classified data (especially at the SECRET level) without much analysis of the threat or evaluation of the adequacy of the offered countermeasures" F. Schneider, ed., Trust in Cyberspace. National Academy Press: Washington, DC, 1998, p. 109. Trust in Cyberspace was referring to MISSI, but the characterization of risk management is general.

[10] Lampson, B.W., *A note on the confinement problem.* Communications of the ACM, Oct.1973.16(10):p.613-615.

[11] IA context of the term "Black" may have originated with early encryption devices dubbed "blackers" because they obscured information into virtual blackness.

[12] General Accounting Office, Report to the Subcommittee on Air & Land Forces, Committee on Armed Services, House of Representative, GAO 08-877, August 2008.

[13] Trusted Computer Security Evaluation Criteria, DoD 5200.28-STD, December 1985.

[14] Structured Analysis and Design Methodology, Office of Government Commerce, an Office of the UK Treasury.

[15] Denning, D. E., "The Limits of Formal Security Models," National Computer Systems Security Award Acceptance Speech, Oct. 1999.

[16] President Clinton, "Executive Order 12958," as amended. The White House, Office of the Press Secretary, April, 17, 1995.

[17] K. Biba, "Integrity Considerations for Secure Computer Systems," MITRE Corp, Bedford, MA April 1977.

[18] NCSC, TG-101, A Guide to Understanding Security Modeling in Trusted Systems, in DoD TCSEC, DoD 5200.28-STD.

[19] Weissman, C., "BLACKER": Security for the DDN, Examples of A1 Security Engineering Trades," Proceedings of the 1992 IEEE Symposium on Security and Privacy pp. 286-292.

[20] Fellows, J., Hemenway, J., Kalem, N., and Romero, S., "The Architecture of a Distributed Trusted Computing Base," Proceedings of the 10th National Computer Security Conference, Sep. 1987, pp. 68-77.

[21] One COTS potentially MLS-capable OE based on a Formal Top Level Specification, August, 2008. http://www.aesec.com/

[22] Davidson, John A., "Asymmetric Isolation," Proceedings of the 12th Annual Computer Security Applications Conference, IEEE Computer Society Press, Dec. 1996.

[23] Denning, D. E., "A Lattice Model of Information Flow" Communications of the ACM, 1976, pp. 236-243.

[24] Bell, David Elliot, "Looking Back at the Bell-LaPadula Model," December 7, 2005, p. 10, as published online at http://www.acsac.org/2005/papers/Bell.pdf