Fifth International Workshop on Requirements Engineering and Law (2012)
doi: 10.1109/RELAW.2012.6347799

Licensing Security

Thomas A. Alspaugh and Walt Scacchi
Institute for Software Research
University of California, Irvine
{*alspaugh,wscacchi*}*@ics.uci.edu*

***Abstract*—There exist legal structures defining the exclusive rights of authors, and means for licensing portions of them to others in exchange for appropriate obligations. We propose an analogous approach for security, in which portions of exclusive security rights owned by system stakeholders may be licensed as needed to others, in exchange for appropriate security obligations. Copyright defines exclusive rights to reproduce, distribute, and produce derivative works, among others. We envision exclusive security rights that might include the right to access a system, the right to run specific programs, and the right to update specific programs or data, among others. Such an approach uses the existing legal structures of licenses and contracts to manage security, as copyright licenses are used to manage copyrights. At present there is no law of "security right" as there is a law of copyright, but with the increasing prevalence and prominence of security attacks and abuses, of which Stuxnet and Flame are merely the best known recent examples, such legislation is not implausible. We discuss kinds of security rights and obligations that might produce fruitful results, and how a license structure and approach might prove more effective than security policies.**

I. Introduction

Software systems security mechanisms for implementing security requirements and policies are often employed on an ad hoc basis rather than in a scalable, organized, and effective manner. Convenient, interactive approaches supported by automated evaluation and guidance are not available because there is no formal basis connecting security requirements and policies with the security mechanisms that are to fulfill them. What is available is a palette of disjoint mechanisms for implementing individual system security features [13], [20] augmented by generalized practices and process standards, such as:

1) mandatory access control lists;
2) firewalls;
3) multi-level security;
4) authentication (certificate authorities, passwords, ...);
5) cryptographic support (e.g. public key certificates);
6) encapsulation (including virtualization and hidden rather than public APIs), hardware confinement (memory, storage, port, and external device isolation) [21], and type enforcement capabilities;
7) data content or control signal flow logging/auditing;
8) honey-pots and traps;
9) functionally equivalent but diverse multi-variant software executables [9], [15];
10) security technical information guides (STIGs) for configuring the security parameters for applications [7] and operating systems [19];
11) secure programming practices (secure coding standards, data type and value range checking, ...) [18];
12) standards for development organization processes and practices rather than system security policies [12].

The reader will note that these mechanisms are *software implementation choices* or *software process choices* rather than *system architectural choices* or *security requirements/policy choices*. Between these mechanisms and a workable concept of a comprehensive security policy for a system or its substantial components is a gap, with no obvious way to bridge it.

- There is no common framework or conceptual basis with which to integrate and evaluate mechanisms in combination. It is unclear how the various security mechanisms are related and how one may contribute to or interfere with another.
- Guidance is scant for analysts, architects, and developers who need to decide which security mechanism to use where, when, how, and why; and also for integrators and administrators who need to decide how to update the selection of mechanisms and their configuration within a system as security needs and policies evolve.

No satisfactory framework exists in which they can be assembled in hierarchical patterns that can be designed and combined in a system architecture to meet specific high-level security policies and requirements.

We believe there is an opportunity to address security requirements challenges throughout a system architecture using *security licenses*.

In our previous work [1], [2], [3], [4] we showed how software licenses for the components of a system can be used to guide architectural choices and evaluate rights and obligations for the system as a whole, even when components are governed by different licenses. Using our approach, a system architect can work both down from the top, propagating desired license rights for the system down to individual components to see what license obligations are required to obtain those rights, and up from the bottom, combining license rights and obligations for components and then subsystems into the total rights and obligations for

the system. In either direction, our approach identifies any conflicts and mismatches among licenses in the architecture.

We propose the same approach for security licenses. System architects and analysts can select desired security rights, assign an expected security license to each subsystem or component, and evaluate interactions between these choices at every level from an individual component up to the entire system. Of course assigning a security license to a component does not guarantee that the component's developer will make it satisfy its security obligations, any more than accepting a component under GPL guarantees that the system's stakeholders will satisfy the GPL IP obligations. But assigning a license (whether security or IP) to each component records the assumptions being made about that component and its use, and evaluating those licenses in the context of the system's architecture identifies mismatches and conflicts among those assumptions for that architecture's design choices. When the evaluation is automated, as it is in our work [2], it forms the foundation for design guidance with respect to the issues raised by the licenses, and a means for combining the potentially dissimilar licenses to evaluate their overall interaction and effect, and thus the overall interaction and effect of the security mechanisms that are expected to satisfy the obligations and of the security requirements and policies that the rights express.

II. Security Licenses

In general terms, a security license is analogous to an ordinary software license such as GPL (GNU General Public License) [10]. Software licenses consist of intellectual property (IP) rights granted by the licensor, in exchange for corresponding license obligations imposed on the licensee. A license presents the rights that are offered, and for each right enumerates the obligations that are required in order for that right to be granted. Many of the actions required for the obligations are related to the actions allowed by the rights. This is particularly so for open-source licenses, for which fulfilling some of the obligations requires parts of the rights that are granted. Also particularly for open-source licenses, the obligations and rights are framed to take effect in an architectural context, with most obligations taking effect with respect to either the component for which rights are granted or component(s) determined by the connectors and architectural topology around that component. Because software licenses are expressed in natural language, the rights and obligations are often presented in an intermingled organization, and much of a license may be devoted to defining terms, classes of entities referred to, and conditions under which the various provisions take effect. But the conceptual structure remains that of a list of rights offered, each in exchange for specific obligations.

Our innovation is to similarly specify components' security rights and obligations, which we can then model, analyze, and support throughout the system's development and evolution, and use to guide its design and instantiation.

There is no "Securityright Act" analogous to the U.S. Copyright Act [22], or Berne Convention [5], to define the exclusive security rights of system stakeholders. We present these possible security rights and obligations as an indication of what sorts of actions might be regulated by security licenses for data organized into security compartments and code organized into components.

A. Some Possible Security Rights

1) The right to read data in compartment T.
2) The right to add data to compartment T.
3) The right to remove data from compartment T.
4) The right to replace component C with another component D.
5) The right to update component C to newer version C′.
6) The right to revert component C to older version C′.
7) The right to add component C in a specified architectural configuration.
8) The right to update component C in a specified architectural configuration.
9) The right to alter the architectural topology of subcomponent B.
10) The right to alter the architecture of system S.
11) The right to add security mechanism M in a specified configuration.
12) The right to update security mechanism M in a specified configuration.
13) The right to remove security mechanism M from a specified configuration.
14) The right to delegate security right R.
15) The right to read the security license of component C.
16) The right to replace the security license L of component C with another security license L′.
17) The right to update security license L.

B. Some Possible Security Obligations

1) The obligation for user U to verify his/her identity, by password or other specified authentication process.
2) The obligation for user U to have been vetted by authority A to exercise security right R.
3) The obligation for user U to be delegated a one-time right by authority A to exercise security right R.
4) The obligation for component C to have been vetted by authority A to exercise security right R.
5) The obligation for component C to have been vetted by authority A to be the object of security right R.
6) The obligation for each component connected to component C to allow it to exercise security right R.
7) The obligation for security license L to meet specified criteria.
8) The obligation for security license L to be approved by authority A.

III. Effectiveness, Manageability, Evolvability

Consider the case of the development of an open-architecture (OA) system integrating proprietary and open-source components from a variety of producers, most of whom do not coordinate their activities and none of whom are controlled by the organization producing the OA system. From the point of view of ensuring security, this is arguably the worst possible case, but it is an increasingly prevalent development model [4]. The OA approach gives access to a wide selection of complex components of high quality, and allows the system to evolve as quickly as its integrators can find appropriate new versions or new components and evolve their architecture and shim code to accommodate them.

Since the producers do not coordinate, they are unlikely to use the same security approaches, and indeed may not even publish what those approaches are. To control security in the resulting system, each component is enclosed in a *containment vessel* [17] that isolates the component with a hypervisor [24] and mediates all communication with the component (method/function calls, data streams, ...) through shim code that monitors and restricts it.

A typical current-day technique [14] for managing security measures is to use *capability lists* to control each component's access to resources such as function calls and data compartments. Each access is delayed briefly while the monitor checks the access against the accessing component's capability list, then blocked if the component was not granted the capability to access that resource. In our experience, each capability list is a text file listing allowed and/or forbidden capabilities, managed manually; new capabilities are typically added to the end of the file. As there appears to be no formal model supporting relationships among capabilities, interactions between capabilities are also identified and managed manually. The text files are detailed, which is a positive aspect, but therefore also long and mind-numbingly tedious, so errors inevitably creep in and are not noticed. Because a capability list has no hierarchy or recursive structure, managing them is not scalable.

A more sophisticated approach is possible using a declarative policy language such as Ponder [6] or an ontology-based language such as KAoS [23] that groups capabilities hierarchically, in ontologies (KAoS) or grouped by roles (Ponder). However, they have no provision for organizing capabilities by software components, combined hierarchically into system architectures, and no obvious connection to law.

We contrast the use of security licenses. In some ways, the approaches are similar, in that our candidate security rights are reminiscent of capabilities, and security licenses can also be used to identify and block disallowed operations automatically. However, because many of the actions required for the security obligations are related by subsumption to those granted by the security rights, and many of the obligations are in the context of the component for which corresponding rights are being granted, it is possible to automatically calculate the interaction of rights and obligations throughout the immediate neighborhood of each component, the subsystem containing the component, and so on recursively on up to the system as a whole [1]. Structuring the security policies as licenses gives a form that is more readily accessible to human readers, and helps convey intention and rationale by relating each obligation to the right it contributes toward. Where the security licenses assigned to the components in the architecture conflict or misalign, automated support can identify the provisions in conflict, locate the conflict to the modules involved, and provide explanations showing the architectural chain of effects that led up to the conflict [2]. Perhaps most importantly, it supports automation of the analysis of interactions between security measures and of the assessment of the system's overall degree and kind of security as a function of the measures taken for each component, group of components, subsystem, and so forth recursively up to the system as a whole..

IV. Recent Events

Coordinated international attacks on vulnerable software-intensive systems of high value and controlling complex systems are becoming ever more apparent. As the Stuxnet case demonstrates, security threats to software systems are multi-valent, multi-modal, and distributed across independently developed software system components [8]. Similarly, it is now clear that even physically isolated systems are vulnerable to external security attacks, via portable storage devices like USB drives, modified end-user devices like keyboards and mice [11], and social engineering techniques [16]. New security measures and policy types are required to defend such systems through new threat detection and parrying methods, as well as appropriate active defense mechanisms. What makes a system or system architecture secure changes over time, as new threats emerge and as systems evolve to meet new functional requirements. Consequently, there is need for an approach that can continuously assure the security of complex, evolving systems in ways that are practical and scalable, yet robust, tractable, and adaptable.

The Stuxnet attacks entered through software system interfaces at either the component, application subsystem, or base operating system level, and their goal was to go outside or beneath their entry context. However, all of the Stuxnet attacks on the targeted software system could be blocked or prevented through security capabilities associated with the open software interfaces that would (a) limit access or evolutionary update rights lacking proper authorization, as well as (b) "sandboxing" (i.e., isolating) and holding up any evolutionary updates (the attacks) prior to their installation and run-time deployment. Furthermore, as the Stuxnet attack involved the use of corrupted certificates of trust from approved authorities as false credentials that allowed

evolutionary system updates to go forward, it seems clear that additional preventions are needed that are external to, and prior to, their installation and run-time deployment. The development-, installation-, and configuration-time rights and obligations of Section II extend the ordinary run-time benefits of security licenses to defend against development-, distribution-, configuration-, and update-time attacks.

V. Exclusive Security Rights

If there could be legally defined and protected exclusive security rights, what would they be? We nominate the following candidates for discussion:

1) *The right of the owner of a copy of a system to replace, update, or revert any of its components.*
2) *The right of the owner of a copy of a system to add or remove components or otherwise alter its the architectural topology.*
3) *The right of the owner of a copy of a system to replace or update the security license of the system or any of its components.*
4) *The right of the owner of a copy of a system to alter its user IO streams or ephemeral data.* (We envision that persistent data may fall into a different category of protected entity.)

As with the exclusive copyright rights, the owner of a right may license all or part of it to someone else in exchange for obligations, for example to allow a trusted system provider to automatically install certain kinds of updates.

Acknowledgements

This research is supported by grant #N00244-12-1-0004 from the Acquisition Research Program at the Naval Postgraduate School, and by grant #0808783 from the U.S. National Science Foundation. No review, approval, or endorsement implied.

References

[1] T. A. Alspaugh, H. U. Asuncion, and W. Scacchi. Intellectual property rights requirements for heterogeneously-licensed systems. In *17th IEEE International Requirements Engineering Conference (RE'09)*, pages 24–33, 2009.

[2] T. A. Alspaugh, H. U. Asuncion, and W. Scacchi. Presenting software license conflicts through argumentation. In *23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, pages 509–514, 2011.

[3] T. A. Alspaugh and W. Scacchi. Heterogeneously-licensed system requirements, acquisition, and governance. In *Second International Workshop on Requirements Engineering and Law (RELAW'09)*, pages 13–14, 2009.

[4] T. A. Alspaugh, W. Scacchi, and H. U. Asuncion. Software licenses in context: The challenge of heterogeneously-licensed systems. *Journal of the Association for Information Systems*, 11(11):730–755, 2010.

[5] Berne Convention for the Protection of Literary and Artistic Works, 1979.

[6] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder policy specification language. In *Int. Workshop on Policies for Distributed Systems and Networks*, pages 18–38, 2001.

[7] Defense Information Systems Agency. *Android 2.2 (Dell) Security Technical Implementation Guide (STIG)*, 2011.

[8] N. Falliere, L. O Murchu, and E. Chien. W32.Stuxnet dossier. Technical report, Symantec, 2011.

[9] M. Franz. E unibus pluram: Massive-scale software diversity as a defense mechanism. In *2010 Workshop on New Security Paradigms (NSPW '10)*, pages 7–16, 2010.

[10] Free Software Foundation. GNU General Public License version 3, 2007. http://www.gnu.org/licenses/gpl-3.0.html.

[11] E. Henning. Attack of the computer mouse. *The H Security*, 2011. http://h-online.com/-1270018.

[12] ISO/IEC. International standard 27001, 2005.

[13] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, et al. The inevitability of failure: The flawed assumption of security in modern computing environments. In *21st National Information Systems Security Conference (NISSC'98)*, 1998.

[14] T. Luo and W. Du. Contego: Capability-based access control for web browsers. In *4th International Conference on Trust and Trustworthy Computing (TRUST'11)*, 2011.

[15] B. Salamat, T. Jackson, G. Wagner, C. Wimmer, and M. Franz. Runtime defense against code injection attacks using replicated execution. *IEEE Transactions on Dependable and Secure Computing*, 8(4):588–601, 2011.

[16] P. Sawers. US Govt. plant USB sticks in security study, 60% of subjects take the bait. *The Next Web (TNW)*, 2011.

[17] W. Scacchi and T. A. Alspaugh. Advances in the acquisition of secure systems based on open architectures. In *Journal of Software Technology*, July 2012.

[18] R. C. Seacord. *CERT C Secure Coding Standard*. Addison-Wesley, 2008.

[19] S. Smalley. The case for security enhanced (se) android. Android Builder's Summit, 2012.

[20] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The Flask security architecture: System support for diverse security policies. In *8th USENIX Security Symposium (SSYM'99)*, pages 11–11, 1999.

[21] K. Sun, J. Wang, F. Zhang, and A. Stavrou. SecureSwitch: BIOS-assisted isolation and switch between trusted and untrusted commodity OSes. In *19th Network and Distributed System Security Symposium (NDSS 2012)*, 2012.

[22] U.S. Copyright Act, 17 U.S.C.

[23] A. Uszok, J. M. Bradshaw, M. Johnson, et al. KAoS policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41, 2004.

[24] Xen hypervisor. http://xen.org/products/xenhyp.html.