

# Infringo Ergo Sum

## When will software engineering support infringements?

Fabio Massacci  
University of Trento, Trento, Italy  
fabio.massacci@unitn.it

### ABSTRACT

Once upon a time a professor of computing and a father was complaining at a radiology ward. A CD with the X-rays of his son's chest had garbled images. Unfortunately, the CD burning process has been outsourced and, in compliance with e-health security policies, technicians could not see the images on the system. Only doctors could. The nurse had a decision to make: sidestep the father (send him away with empty hands to the pneumology ward) or sidestep the system (give the technician the doctor's password and thus the ability to access all images and not just this one). As a father he was happy of her decision. As a professor, this knowledge was of meager and unsatisfactory kind.

Any human decision maker who experienced the need of a *local IT infringement* in order to achieve her business goals knows that she is offered only the choice between strict compliance (and failure of business goals) or *global violation* (and failure of security goals).

Software engineers do not simply know how to deal with infringements. I believe that a different alternative should be possible. The goal of this paper is to sketch the challenges of such unexplored scientific alternative.

### Categories and Subject Descriptors

K.6.5 [MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS]: Security and Protection; D.2.11 [SOFTWARE ENGINEERING]: Requirements, Specifications—*Elicitation methods, methodologies*; H.1.2 [MODELS AND PRINCIPLES]: User/Machine Systems—*Human Factors, Software psychology*

### General Terms

Design, Human Factors, Security

### Keywords

Access Control, Infringement management, Human Factors, Requirements, Security, Socio-Technical Systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*FoSER 2010*, November 7–8, 2010, Santa Fe, New Mexico, USA.  
Copyright 2010 ACM 978-1-4503-0427-6/10/11 ...\$10.00.

### 1. INTRODUCTION

This simple, and unfortunately real-life, example stems from an irreversible evolution that IT systems have undergone: we no longer have a software system whose design and behavior can be predicted with purely technical methods. We face a socio-technical system (STS) [21] that

involve complex interactions between software components, devices and social components (people or groups of people), not as users of the software but as players engaged in common tasks [10].

Human decisions will not necessarily be software supported, nor planned in advance, sometimes nor even informed, but they will nonetheless be taken. The software has not advised the nurse of alternative solutions. Still, she had to choose.

The parallel technical developments of service-oriented architectures, remote privilege management infrastructures, remote maintenance and outsourcing have distributed and multiplied the points in which humans interact with a STS and control or anyhow influence its behavior. The rationale and even the potential existence itself of this particular decision point by the radiology staff are likely unknown to IT department.

How to manage requests to violate the specified company's policy? The engineering solution of choice is to escalate access requests to more powerful entities in the system. This approach is hardly usable even for personal users [29, 14]: how many users would click "No" on the "Do you allow services.exe<sup>1</sup> to modify your PC's configuration?" pop-up). This (attempt to a) solution is even less appropriate to STS, as noted by [22] in a field study on IT risks in finance:

These massive collections of people lead to greater anonymity of the employees as they fade into the masses and can conceal actions that challenges modern security. . . This is worsened by the organizational counterpart of service oriented architectures. . . it becomes less evident who reports to whom and who is responsible for permitting and terminating data access. Employees may no longer have direct managers as roles such as functional manager, group manager, engagement manager, review manager, and co-manager have become prevalent in financial institutions, professional service firms and corporations.

<sup>1</sup>The tragedy is that this name is not invented, it runs on your Windows box and of course you have no clue about its doings.

Usability studies (e.g. [2, 27]) also confirm the widening gap between policies decision makers and policy implementors, frequently due to the existence of many decision makers. In my own example, who should have authorized the nurse? The head of radiology? the head of the pediatrics? The head of IT department? The head of personnel?

The key word here is *change*. The context for which the e-health software was designed has changed and the nurse needed to cope with it. The S-side of STS can manage changes directly: we know that we might need to do different things than originally stipulated (i.e. infringements) because things change but goals stay the same.

The scientific challenge is that the T-side is not so responsive to changes while keeping a reasonable security. While significant research exists on enforcement formal models and techniques (e.g., Polymer [3], PSLang [9], and Security-by-Contract [7], Load-time security certification on Android [8], Usage Control [20], etc.), and significant methods for security engineering (e.g. anti-goals [28], misuse cases [25], security problem frames, SI\*/Secure-Tropos [11], etc.) or secure software development (e.g. Microsoft SDLC, or BSIMM) they are all permeated by the idea that infringements are violations and as such should not be permitted. Whenever infringements are permitted, this permission is totally assigned to the human component, being it a designer or a user without supporting her judgments.

Thus software infringement management is not predictable, circumstances are not replicable and the system cannot later audit and distinguish between correct infringements (where it was the best option, e.g. to save a patient's life) from incorrect ones (e.g. frauds).

We should design a technical solution so that organizations can balance the need of getting the work done (send a patient to the pneumology ward with the correct diagnosis) in presence of changes to the original course of action (the garbled x-rays) without incurring each and every time the risk of unforeseen toxic over-entitlements (the give away of the doctor's password).

This is the next frontier for software engineering: *software management of human infringements due to changing circumstances*. In order to address it we need at least to

1. capture business goals and trust relations among humans that might mirror illegal but de facto relations (the nurse had the doctor's password all the way through);
2. manage infringements in presence of changed circumstances (the failure of x-ray standard procedure with the request for a timely answer by a colleague in another ward);
3. identify higher-order policies that humans use to solve inconsistencies and that takes into account goals and priorities (if the image was not affected but only the report, nor a child be involved, most likely the nurse would have just handed me a fresh printout of the report and told me to come later for a new CD).

The practical problems we face are evident and in the rest of the paper I will discuss the societal impact of the challenge (by looking more in details at the figures that makes the headlines in §2) and whether it is actually a research problem and not just a problem of lousy implementation (by looking at the research literature to see what foundational concepts

it misses in §3). Finally I will conclude with a more detailed description of the research challenges (§4).

## 2. SOCIETAL IMPACT

The key claim here is that major software security problems stem from decisions of "users" that by making wrong decisions allow other (malicious) users to subvert the functionalities of the STS. This has been an old claim by E. Shawn and others' essay on insider threats and the psychology of potentially dangerous insiders:

Paradoxically, [...] there has been little systematic study of vulnerable insiders, while major investments are being devoted to devising technologies to detect and prevent external penetrations. Technological protection from external threats is indeed important, but human problems cannot be solved with technological solutions. Without a detailed examination of the insider problem and the development of new methods of insider risk management, such an unbalanced approach to information systems security leaves critical information systems vulnerable to fraud, espionage or sabotage by those who know the system best: the insiders [24].

Yet, the headlines are populated by smart, technical penetrations and exploitations of bugs by viruses, worms, sniffers and keyloggers. The 2008 survey of IC3 showed a total loss of frauds to individual of \$264.6 million. A study on the underground economy [15] showed that tens of thousands of credentials are stolen from e-Commerce websites, webmail providers (such as Windows Live, Yahoo, Google), or social networks (Facebook, YouTube, etc.) with almost 6.000 credit card numbers in the just a fraction of the dropzones of a key logger.

While such data offers a picture of the internet as the wild west of organized criminal hackers, a more careful analysis of the data reveals a completely different picture. Shawn et al paper was published in 1998 (almost 10 years ago). In the same year, the FBI Computer Crime Survey reported the average cost of a hacker penetration at \$56,000, while the average insider attack was \$2.7 million.

Ten years later, attack types have changed (keyloggers and botnets have replaced hackers and viruses) but the price differential between external and internal attacks has even increased! The IC3 reports on individual fraud in 2008 showed indeed that

"Nearly fifteen percent (14.8%) of these complaints involved losses of less than \$100, and (36.5%) reported a loss between \$100 and \$1,000. In other words, over half of these cases involved a monetary loss of less than \$1,000.

In sharp contrast, a US Secret Service study from the same year showed that losses reported to insider frauds took no less than \$500 and with more than 50% of respondents claiming losses over \$20.000 with 12 respondents claiming a loss over \$1.0 million. Further 28% of respondents added a significant loss in term of reputation

If one dig further on the figures of the IC3 reports, more than 90% of all frauds were indeed due to the (more or less gullible) human decision makers: non-delivery of merchandise and/or payment (32.9%), auction fraud (25.5%), confidence fraud such as Ponzi schemes, computer fraud, and

check fraud (19.5%) made almost all referred complaints. Traditional scams such as Nigerian letter fraud, phishing etc. together represented less than 9.7%.

In a nutshell, attacks to the T-component of the Internet as STS might make the headlines but play a significantly minor role in the impact on digital confidence which was mostly driven by the decision of humans, my key point.

### 3. MISSING FOUNDATIONAL CONCEPTS

We know how to engineer a system if we have some security policies. They can just be considered requirements (albeit of a particular kind). But what we can do if having a security policy is not enough?

The traditional RBAC model [23] was essentially proposed to better organize the assignment of permissions to user in order to reflect the organizational structure. Of course reality requires many access decisions to be dependent on the context and this lead to an escalation of RBAC models with constraints such a time, location, or run-time context [4, 6]. In order to provide more dynamicity a number of models have been proposed to regulate what a user can do in a workflow by using obligations languages for privacy policies [1] or usage control rules (=access control + monitoring users) [18, 20].

All these frameworks are interesting and appropriate in their application domains but they all are in the same frame of mind: at the beginning of the day the administrator sets the rules, as complicated and with as many exceptions as he can possibly write, and then the system will provide a strict enforcement of those conditions. The system can also monitor the user to make sure that he complies with additional obligations. Wrong access or missing obligations are sins to be prosecuted.

In order to avoid unfair prosecution we must make sure that all possible legal sequences are captured by the policy and not just the main workflow. In principle, we could think of eliciting all possible exceptions. Programming languages such as Java have try-catch constructs and a number of modeling and requirements languages allows for the description of exceptions in business processes (e.g. [26, 19] for the little JIL language). Unfortunately the very idea of investigating, reporting, regulating, and implementing all possible exceptions would be out of synch with the IT budget in the real world: it is simply too expensive<sup>2</sup>. If any of the access control system described above had been the back-end in my personal episode, the nurse at the ward would have likely faced the same decision. Even if we had an unlimited budget, there could still be cases in which we cannot actually list those exception for social reasons (as opposed to just technical incompleteness). I will discuss this phenomenon more in detail in the next section.

If access control papers don't allow for managing infringements, we could turn ourselves to run-time security enforcement mechanisms. For example Hamlen's work on rewriting [13], and Ligatti et al. works on edit automata [3]. Most works had a running system counterpart capable of enforcing those security policies [9, 12, 17].

<sup>2</sup>When I was deputy rector for ICT procurement, the RBAC system of accounting for the University of Trento included around 100 roles, 1500 activities, and exceptions to the 60K processes were "uncountable"... as it would have required a Deloitte consultant at 1KEuro/day to count them precisely.

Yet, as we have shown in [5], there is a gap between the theoretical constructions presented in the papers and their practical implementation. This gap can be explained by the too coarse grained classification provided by the formal properties of a "good" software enforcement mechanism: transparency and soundness.

Soundness says that every output of enforcement mechanism should be valid; transparency says that in case of valid input, the output should be equal to the input. Practical software systems (such as Polymer [3], PSLang [3] or our own Security-By-Contract inline monitor [7]), being implemented, will always correct the bad traces in *some* way (possibly useful to the user). But formal papers on enforcement don't provide formal means to distinguish those ways to deal with bad traces,. Most papers focused on the shape of good traces that can be potentially enforced with this or that enforcement mechanism.

In practice, this is not enough. What distinguishes enforcement mechanisms is not what happens when traces are good, because nothing should happen (transparency, isn't it?). The interesting part is how precisely bad traces are converted into good ones. To this extent soundness only says they should be made good. We see a glimpse here of our problem (users may misbehave) but the solution is under-specified (back to strict compliance, but how?).

Let's start from a concrete example of a workflow for drug dispensation, in which doctors must read through therapeutical notes. Suppose now a doctor forgot to click through one note. A principled implementation of the formal construction used by Ligatti, Bauer and Walker [3] to show that edit automata can enforce any renewal policy, would suppress actions of doctors, until the guilty one would insert that missing click, potentially for ever if such action was no longer possible. In [5] we provided an alternative mechanism that suppresses just the part of execution where the doctor made this infringement, but keeps working for the rest.

Where is the problem? both our approach and Ligatti's approach satisfy soundness and transparency on good traces. So we cannot formally distinguish between an enforcement system discarding the work of a day and a system dropping the last 20 minutes. Further, both software systems can claim to provide a faithful implementation of the policy because the specification of infringement management is traditionally not part of the policy. Clearly, these notions of soundness and transparency are not enough to provide a security foundation for software when humans are involved.

### 4. RESEARCH CHALLENGES

As shown above, we need a new foundational concept:

**CHALLENGE 1.** *The software enforcement mechanism should formally capture the idea of "reasonable" behavior when the requests (of the users) do not correspond to the policy.*

The first pillar of software engineering textbooks is that systems should be designed to meet the *stated* requirements (including user's expected behavior). Here, we are hacking away the basement of software engineering courses: seen all those use cases? Well, your system ought to be reasonable when the users do *not* behave like that.

This might look an absurd requirement for a designer of a T-component but it is an obvious observation for S-components. S-components have a clear notion of "venial errors" and "serious errors" and do not to bring a critical

process to a halt for a venial error. The challenge we face is that the T-side ignore this notion: all deviations are serious policy violations and are prosecuted by the T-system as such. As a consequence, the human decision maker finds the behavior of the T-component irrational and decides to bypass it. The result is a security failure.

The challenging part for the designer is that the word “reasonable”: the users involved must be convinced that the software’s behavior is not arbitrary. A software with infringement management capabilities should be able to reproduce the typical human decision strategies such as veto, favor, complementary, substitutability and other interactions among the viewpoints of the different decision makers [16]. The agents who legitimately infringed the policies - because at the moment it was the most reasonable action to take - should be able to demonstrate and justify the legitimacy of their actions.

In order to be (or look) reasonable, we must of course elicit requirements on what is a reasonable behavior but...

**CHALLENGE 2.** *No human can write in a formal document how to violate the letter of the company policy while enforcing its spirit. We must capture “unspeakable requirements and trust relations” in the work practice.*

Human components have certain trust relations that are implicit in their workpractice. In the Trento-Toronto line of work that I started with John Mylopoulos, Nicola Zannone and others we have tried to draw requirement models with trust constructs (eg [11]). Yet, we also sinned, as many requirement engineers from the RE assumptions that such relations can be made explicit. Many years of case studies showed me that what the users can explicitly tell is only a fraction of reality.

The problem we face for “security exceptions” is that those are *requirements that cannot be formalized* and still they are the de-facto standards. Eliciting these requirements is the real challenge. I’m sure that nowhere in the requirements documents of the e-health system and in the privacy policy document of the hospital there is a statement specifying that nurses can have doctors’ password but this is a well known phenomenon<sup>3</sup>.

If we want to meet this challenge we must be able to design models of trust appropriate for dynamic and evolving STS. the models should capture real but unspeakable word practices of users and organizations: the opinions of the users and their attitudes towards the activities supported by the software. Understanding their mental models on “how they think the stuff works” as opposed to “how the stuff actually works” as done in [29] is important to identify appropriate countermeasures to security threats. having this difference in mind, the T-side could be able to ascertain, from the goals and anti-goals of the actors and their behaviors, that a trust-relation applies and consider some actions venial errors and other actions serious errors. In order to do this, we need

<sup>3</sup>Another example: you were invited to a meeting located at an arbitrary EU Commission premise but your name is not on the list of authorized people at the entrance. What happens? If you know the officer’s name and the meeting’s title, the guard adds you and your passport details with a ball pen at the end of the printed list. Exercise for the reader: find trace of this process in any formal document. Once in January 2010 the guard was new and wanted to stick to the official process. It was a chaotic but interesting experience, in primis for the EU officer who invited us...

to understand work practices with ethnographic studies and try to refine decision theories when many stake holders are present. It is a new type of software engineering (trust engineering?) where the focus is no longer system or interface design but rather the human-human and human-computer decision making process.

Once we have understood how humans take (security) decisions and designed a “reasonable” system that can supports them what else is missing?

**CHALLENGE 3.** *It should be possible to derive in a principled way the enforcement mechanism from the policy and the high-level criteria for dealing with infringements.*

We don’t want a specification of the company policy with all possible exceptions and infringements, a spaghetti-workflow, likely full of errors and incomprehensible.

What we need is to extend the “routine” workflow with the notion of higher-level policies to provide technical support to “reasonable” infringements. For example, by introducing formal notions corresponding to venial errors or amendable errors in a software process, and by capturing notions of accountability and awareness of the context. Using these notions the software engineer should design a T-component that only takes the default policy and a specification on how to deal with venial errors and cope with them wherever they appear. If we change the notion of venial errors, the software for infringement management should follow suit. This is the bridge between the first and the second challenge.

In order to validate the challenges, it is important to have, beside theorems and prototypes, a validation with real users.

The challenges are ambitious and risky but worth taking:

- If we do not capture the higher-level mechanisms for policy enforcement, we will just generate yet another \*-RBAC , with more sophisticated constructs to deal with all uncountable exceptions.
- If we don’t capture unspeakable trust requirements we will generate another secure software engineering methodology. There will be less buffer overflows but not more secure software.
- If we do not involve real users in the validation, we could never be sure that the way decision making actually take place is the way designers have speculated.

Until software engineering research solves these challenges, the (mis)use case diagram for infringement management in STS could only include soft-eyes-and-kind-words as the only relationship between the human user (the father) and the S-component (the nurse) while they side step the system.

## 5. ACKNOWLEDGMENTS

I would like to thank N. Bielova, C. Labreuche, M. Nalin, A. Pretschner, A. De Angeli, M. Clavel, S. Etalle, Y. Paindaveine, D. Presentza, M. Petkovic, and N. Zannone for many useful comments on the topic. Most of the good ideas described here stemmed from discussions I had with them. The remaining weird ones are of course mine.

This research is partly supported by the EU-FP7-IST-IP-MASTER project<sup>4</sup>, the EU-FP7-FET-IP-SECURECHANGE project<sup>5</sup>, and the EU-FP7-IST-NoE-NESSOS project.

<sup>4</sup>[www.master-fp7.eu](http://www.master-fp7.eu)

<sup>5</sup>[www.securechange.eu](http://www.securechange.eu)

## 6. REFERENCES

- [1] M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. In *Proc. of the 8th ESORICS*, volume 2808 of *LNCS*, pages 162–180. Springer-Verlag, 2003.
- [2] L. Bauer, L. F. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea. Real life challenges in access-control management. In *Proc. of ACM CHI'09*, pages 899–908. ACM Press, 2009.
- [3] L. Bauer, J. Ligatti, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. *Int. J. of Inform. Sec.*, 4(1-2):2–16, 2005.
- [4] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *TISSEC*, 4(3):191–233, 2001.
- [5] N. Bielova, F. Massacci, and A. Micheletti. Towards practical enforcement theories. In *Proc. of The 14th Nordic Workshop on Secure IT Systems (NordSec'09)*, volume 5838 of *LNCS*, pages 239–254. Springer-Verlag, 2009.
- [6] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. Geo-rbac: A spatially aware rbac. *TISSEC*, 10(1):2, 2007.
- [7] L. Desmet, W. Joosen, F. Massacci, P. Philippaerts, F. Piessens, I. Siahaan, and D. Vanoverberghe. Security-by-contract on the .net platform. *Inform. Security Tech. Rep.*, 13(1):25–32, 2008.
- [8] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proc. of CCS'09*, pages 235–245. ACM Press, 2009.
- [9] U. Erlingsson. *The Inlined Reference Monitor Approach to Security Policy Enforcement*. Technical report 2003-1916, Department of Computer Science, Cornell University, 2003.
- [10] J. L. Fiadeiro. On the challenge of engineering socio-technical systems. In *Software-Intensive Systems and New Computing Paradigms*, volume 5380 of *LNCS*, pages 80–91. Springer-Verlag, 2008.
- [11] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements engineering for trust management: model, methodology, and reasoning. *Int. J. of Inform. Sec.*, 5(4):257–274, 2006.
- [12] K. Hamlen, G. Morrisett, and F. Schneider. Certified in-lined reference monitoring on .net. In *Proc. of the 2006 workshop on Prog. Lang. and analysis for security*, pages 7–16. ACM Press, 2006.
- [13] K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *TOPLAS*, 28(1):175–205, 2006.
- [14] C. Herley. So long, and no thanks for the externalities: The rational rejection of security advice by users. In *Proc. of the 2009 New Sec. Paradigms Workshop*, pages 133–144. ACM Press, 2009.
- [15] T. Holz, M. Engelberth, and F. C. Freiling. Learning more about the underground economy: A case-study of keyloggers and dropzones. In *Proc. of the 14th ESORICS*, volume 5789 of *LNCS*, pages 1–18. Springer-Verlag, 2009.
- [16] C. Labreuche. Argumentation of the decision made by several aggregation operators based on weights. In *Proc. of the Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 683–691, 2006.
- [17] J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *TISSEC*, 12(3):1–41, 2009.
- [18] J. Park and R. Sandhu. The UCON ABC Usage Control Model. *TISSEC*, 7:128–174, 2004.
- [19] H. Phan, G. S. Avrunin, and L. A. Clarke. Considering the exceptional: Incorporating exceptions into property specifications. Technical Report UM-CS-2008-32, Dep. of Computer Science, Univ. of Massachusetts, 2008.
- [20] A. Pretschner, M. Hilty, and D. Basin. Distributed Usage Control. *Commun. ACM*, 49(9):39–44, September 2006.
- [21] J. Rooksby, M. Rouncefield, and I. Sommerville. Testing in the wild: The social and organisational dimensions of real world practice. *Comp. Supported Cooperative Work (CSCW)*, 18(5-6):559–580, 2009.
- [22] S. S., S. Smith, S. Trudeau, M. Johnson, and P. A. Information risk in financial institutions: Field study and research roadmap. In D. Veit and et al., editors, *FinanceCom 2007*, volume 4 of *LNBIP*. Springer-Verlag, 2007.
- [23] R. S. Sandhu and *etal*. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
- [24] E. Shaw, K. G. Ruby, and J. M. Post. The insider threat to information systems: The psychology of the dangerous insider. *Security Awareness Bulletin*, 2, 1998.
- [25] G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *REJ*, 10(1):34–44, 2005.
- [26] B. Staudt Lerner, S. Christov, A. Wise, and L. J. Osterweil. Exception handling patterns for processes. Technical Report UM-CS-2008-06, Dep. of Computer Science, Univ. of Massachusetts, 2008.
- [27] J. Tam, R. W. Reeder, and S. Schechter. IŠm allowing what? In *Proc. of Security and Human Behavior (SHB'10)*, 2010. Available on the SHB web site.
- [28] A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proc. of ICSE'04*, pages 148–157. IEEE Computer Society Press, 2004.
- [29] R. Wash. Folk models of home computer security. In *Proc. of the ACM Symp. Usable Privacy and Security (SOUPS'10)*, pages 1–16. ACM Press, 2010.