# Trust Relationship Modeling for Software Assurance

David Burke, Joe Hurd, John Launchbury, Aaron Tomb
Galois, Inc.

June 2010

**Abstract**

Software assurance, as practiced through the Common Criteria, is a mixture of processes, heuristics, and lessons learned from earlier failures. At the other end of the spectrum, formal methods establish rigorous mathematical properties of portions of code. By themselves, neither of these practices are scalable to software systems with millions or billions of lines of code. We propose a framework that enables the collection and analysis of many disparate types of information to be applied to the issue of software assurance. *Trust relationship modeling* enables stakeholders to decompose the overall security policies into security obligations throughout a system, and then to reason about the consequences.

## 1 Introduction

A few decades ago, systems were typically standalone, so their environments were relatively constrained and predictable. Compare this to the current situation—most critical software doesn't operate in a static environment with well-defined inputs. Today's systems are essentially *systems of systems*, comprised of a tremendous number of components, with complex interconnections between them. These components are typically built by different teams at different times, making assurance a serious challenge. In addition, they are *socio-technical* systems: people aren't simply users set apart from it—they are an integral part. In this context, the current paradigm for system assurance is running up against limits. It's too expensive in terms of time and resources, and there is very little reason to believe that the results are based on rigorous foundations.

One bright beacon of light is the recent and dramatic leap forward in the capabilities of formal methods. Formal methods can bring a level of confidence in certain claims that testing cannot match. However, formal methods all rely on mathematical models for (portions of) a system. What happens if the model of the system is violated? Perhaps the model ignored an aspect such as integer overflow, for example, or perhaps external factors could violate the model, factors like voltage spikes, or cosmic rays flipping bits. This *category mistake* was the source of the controversy surrounding the VIPER processor, parts of which had been formally verified but even so there could be no guarantees that the physical chip would operate without error [2].

In this paper we outline a framework for gaining assurance about a whole system, even though parts of the system are outside the realm of the formal verification. The heart of our approach is the pairing of concrete, relevant claims with compelling evidence, where those claims have been derived from known threats confronting the system. We use the claims, and the evidence for those claims, to assess the residual risk from the threats confronting the system.

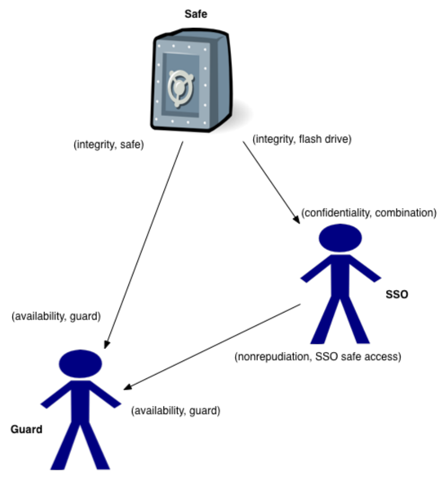
Security engineers traditionally derive system requirements from the anticipated threats to the system [1, 3]. But, we prefer instead to counter threats with 'claims' for the system. Claims are more naturally written in the form of a contract: a more concrete, specific, and direct language that captures the idea that the system is taking action to mitigate a threat. So for example, instead of saying that we have an encryption requirement for message traffic, we make a claim like "message traffic is protected against eavesdropping", or even better "message traffic is protected against eavesdropping through AES encryption".

There are two important implications of this approach. First, we need a method that links threats and the associated claims: a way to translate the statements of threats into security enforcement obligations (the claims) throughout the system. We call this process a *Trust Relationship Analysis.* Secondly, we need a method that unifies the diverse kinds of evidence generated in support of claims, so that a quantitative measure of the strength of claims and strength of evidence can be calculated. This process of structuring evidence against claims is called building *Security Assurance Cases.*

## 2 Trust Relationship Analysis

Suppose that we have a flash drive that contains important data, that we wish to protect against the threat of unauthorized modification (i.e., protect the integrity of the data). That is, we want to build a system whose overall security policy is "protect the flash drive against unauthorized modification". After making an assessment about the seriousness of the threat, we decide to mitigate it by physically protecting the flash drive. So we purchase a safe from a reputable manufacturer, and give the combination to the site security officer (SSO), who is responsible for not sharing that combination with anybody.

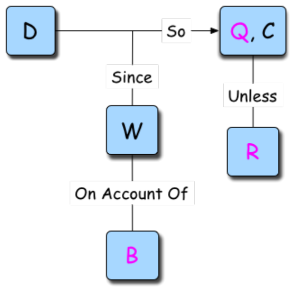
At this point in the discussion, we have identified two properties: (1) the safe has the obligation to enforce the integrity of the flash drive; and (2) the SSO has the obligation to enforce the confidentiality of the combination. We can say that, with respect to the enforcement of Property 1 (an integrity property), we are trusting that Property 2 (a confidentiality property) will be upheld. This is what we mean when we say that there is a *trust relationship* between the two properties. If Property 2 isn't trustworthy (say, the SSO shares the combination with his drinking buddies), then we have much less confidence that Property 1 will be enforced.



Suppose that further evaluation of the threats to our system reveals that we need to take seriously two additional threats: that somebody might attempt to gain unauthorized access to the safe, by either figuring out the combination, or by blowing the safe door open with explosives, or that the SSO engages in an insider attack, using the combination to sneak into the safe, and make changes to the data. To mitigate these threats, the system is extended with a guard in front of the safe. The guard doesn't have access to the safe combination, but the presence of the guard will deter the safe-cracking threat. In addition, the guard records whenever the SSO accesses the safe. This gives us the following additional properties to reason about: (3) the availability of the guard; (4) the integrity of the safe; and (5) non-repudiation of the SSO's safe accesses.

The diagram above illustrates the trust relationships in the system: one from the safe to the guard (the integrity of the safe is enforced by the availability of the security guard); and one from the SSO to the guard (nonrepudiation of safe access is enforced by the availability of the guard). Further refinements of the system can be made (do we need to worry about the SSO suborning the guard?) until we are satisfied that we have captured all of the relevant trust relationships in the system.

Notice that all the properties can be written as *property*-of-*asset*, where *property* is one of the canonical security properties: Confidentiality, Integrity, Availability, Authentication, or Non-repudiation. Given a system and a high-level statement of the overall security goals of the system (its *security policy*), we analyze the trust relationships by decomposing the overall system security policy into a set of trust relationships which enforce those obligations. As trust relationships are defined, they tend to expose more threats (for instance, now that we've identified a guard as being a system agent, we might want to consider threats against the guard being incapacitated, or distracted, etc.). So a trust relationship analysis ends up being a method for doing a threat assessment.



## 3 Security Assurance Cases

Once we have the trust relationships of the system, we build the evidence that the trust relationships are going to be maintained. We prepare an *assurance case*, much like a lawyer prepares a legal case, where we will draw in many different categories of evidence, making clear how each piece of evidence is relevant, credible, and supportive of the case.

For example, process evidence would provide evidence against other some kinds of claims (for instance, "due to our configuration management processes, it is very unlikely that somebody outside the project team could have modified the source code"). Correspondingly, formal methods are used as a tool to generate compelling evidence supporting the claim that key algorithms have been implemented correctly. And evidence of *pedigree* ("we have engineers on this project who have worked on similar projects that have been successfully deployed without serious incident") and *provenance* ("our code is built on top of X, which has been in use for 10 years without demonstrating any access control vulnerabilities") don't sound like special pleading in this framework: they simply become specific types of evidence for the case.

Organizing and structuring this evidence is challenging, and here our framework uses Toulmin structures, a form of argument that has been influential in the legal fields for a long time, and has been gaining traction in the safety community in recent decades [4]. The thrust of his work is to argue that real-world arguments rarely conform neatly to syllogisms. Instead, a *Toulmin structure* has three elements at its core: (a) a claim C; (b) evidence for that claim D; and (c) justification as to why the evidence supports the claim (a warrant W). The diagram of a Toulmin structure above shows the basic elements, plus elements to address challenges to the justification (a "Because", a "Qualifier", and a "Rebuttal").

We construct our assurance argument by producing a set of Toulmin structures. The idea is to start with a claim, and to decompose the claim into a set of sub-claims, and so on. We stop our recursive process of refining claims when the evidence to support the claim seems like the right level of granularity. So in the safe example case, knowing that we have access to somebody's security training record would encourage us to say "we can stop here: we think we have evidence that lines up well with the claim." Contrast this situation with one in which a claim is very broad, for example: "this information will not leak outside the organization." In this case, we'd likely need to refine the claim before feeling confidence that our evidence is, or could be adequate to justify the claim. And as mentioned previously, we would expect to produce a set of claims and associated evidence to cover both the design and implementation of the system.

## 4 Summary

At Galois, we have found that one of the immediate benefits of doing a trust relationship analysis is that it makes explicit the assumptions and relationships between the threats and the various components of the system that are meant to mitigate those threats. Trust relationship analysis lends itself to an iterative approach, where the decomposition suggests a preliminary security architecture, and as the trust relationships are defined, we can assess whether those obligations seem realistic. This technique also allows us to take into consideration the severity of the threats against the system at various stages of the design.

We have found that using Toulmin structures to organize our arguments allows us to combine evidence of many different kinds. We are producing models in which the assumptions are transparent, the methodology is repeatable, and the outcome is an assessment of residual risk. This evidence-based approach lets us tailor the tools that we bring to bear on each claim: formal methods; testing; configuration management; and so forth all have their place in an assurance argument. It demonstrates, for example, the role of *run time measurement* paired with the use of formal methods to establish the correctness of the system in the first place.

Given the success of Toulmin structures in constructing safety cases, it is not all that surprising that they apply well to security cases too. Some differences emerge, however. In the safety world, *mean time before failure (MTBF)* is the primary measure, which serves as a base case for probabilistic calculations of the safety of the whole system. With security there are multiple properties under consideration (confidentiality, integrity, and so on), and these all interact to support or negate each other, depending on the configuration of the system. Currently the trust relationship analysis is *pre-formal*, as it is unclear at this point what the underlying mathematical model may be. However, we believe it lends itself to a high degree of formalization, which is likely to be an important consideration when dealing with complex systems. In particular, we are finding that applying various modal logics to trust relationships enables us to make inferences about system security that wouldn't be possible from a visual inspection of the system. This is an area of active research at Galois.

Additionally, whereas the analysis of safety cases commonly bottom-out with physical systems (like the integrity of the metal forming a wing, for example), it's our experience that security cases very commonly bottom out with people, reflecting the adversarial nature of security compared with safety. Of course, in the end we really care about *dependability* rather than just security, and dependability encompasses both security and safety, so all the safety considerations will have to come into play too.

## References

[1] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* Wiley, 1 edition, January 2001.

[2] Donald MacKenzie. *Mechanizing proof: computing, risk, and trust.* MIT Press, 2001.

[3] Gary McGraw. *Software Security: Building Security In.* Addison-Wesley Software Security Series. Addison-Wesley, February 2006.

[4] Stephen Toulmin. *The Uses of Argument.* Cambridge University Press, 1958.