Applying Problem-Structuring Methods to Problems in Computer Security

Peter Gutmann
University of Auckland
Department of Computer Science
Private Bag 92010, New Zealand
+64 9 373-7599
pgut001@cs.auckland.ac.nz

ABSTRACT
Solutions to security problems, particularly ones involving cryptography, have typically been approached through the Inside-Out Threat Model, "this is our solution and whatever it addresses is the threat". Email encryption/signing and SSL/TLS are two examples of the Inside-Out Threat Model, with the existence of a multi-billion dollar global cybercrime industry testifying to the fact that the threat-modelling performed during the design process was aimed more at satisfying the cryptographers' rather than the end users' needs. This paper looks at the application of problem-structuring methods or PSMs, a technique from the field of social planning, to address computer security problems, not so much to define technical solutions but to help analyse the problem so that the most appropriate, rather than simply the most technologically trendy, solution is applied to the problem.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – *elicitation methods*; D.2.10 [**Software Engineering**]: Design – *methodologies*; H.1.2 [**Information Systems**]: User/Machine Systems – *human factors*; K.6.1 [**Management of Computing and Information Systems**]: Project and People Management – *systems analysis and design*.

General Terms
Design, Security.

Keywords
Problem-structuring methods, wicked problems, soft operations research, soft systems methodology.

1. INTRODUCTION
Engineering solutions for security problems is always a tricky business. Try this simple exercise: Grab a passing geek and ask them how they'd solve the problem of securely authenticating users over the Internet. They'll probably tell you to use OpenID (or some pet equivalent), LDAP, SecurID (or a pet equivalent), smart phones as access tokens, or something similar. They're unlikely to ask who's being authenticated, to what, under which conditions, in which environment, what the budget is, how easy the authentication mechanism has to be to use it, and so on, or even whether authentication makes any sense when what's usually required is authorisation of an action rather than just plain authentication. This is a prime example of the Inside-Out Threat Model in action, with the solution decided at the wrong end of the design process. The intent of applying formal problem-structuring methods is to turn this process around, moving the technology decisions to the end of the design process (if they're even needed) and considerations that affect the choice of technology to the start.

Problem-structuring methods are a technique designed to solve "wicked problems" and come from the field of social planning. Wicked problems were first proposed in the early 1970s as a way of modelling the process for dealing with social, environmental, and political issues and have since been extended to various other fields, but not (as far as the author is aware) to computer security. Amongst a wicked problem's weaponry are such diverse elements as a lack of any definitive formulation of the problem, a lack of a stopping rule (so that one of the core requirements for dealing with a wicked problem is the art of not deciding too early which solution you're going to apply), solutions that are rateable only as "better" or "worse" and not true or false, no clear idea of a which steps or operations are necessary to get to the desired goal, and a variety of ideological and political differences among stakeholders. With these sorts of problems, simply defining and analysing the problem becomes a significant, if not major, component of the solution. The techniques used to address these sorts of problems are therefore rather appropriately labelled "problem-structuring methods" (PSMs) rather than "problem-solving methods".

2. PROBLEM-STRUCTURING METHODS
Employing problem-structuring methods or PSMs to address computer security problems is helpful in order to avoid the natural tendency of geeks to leap in with their favourite piece of technology without considering the environmental, social, political, and legal aspects of the overall problem. A typical example of this occurred just before this paper was being finalised when a CA trusted by web browsers was used to issue fraudulent certificates for high-value sites. The response to this failure of browser PKI was endless discussion in technical forums about how to solve the problem through the application of more PKI, despite there being "no evidence of a single user being saved from harm by a certificate error, anywhere, ever" [1]. Consideration of what the overall problem that was meant to be being solved was (in the case of web browsers, wide-scale phishing of users and to

NSPW'11, September 12–15, 2011, Marin County, CA, USA.

a lesser extent drive-by downloads and other unpleasantness), was almost entirely absent.

There are quite a range of PSMs (a more complete discussion of the background behind them and their various pros and cons is given elsewhere [2]), but the one that seems most appropriate for looking at technology problems involving computer security is the Soft Systems Methodology or SSM [3][4][5]. The framework provided by SSM presents a powerful analysis tool for examining security issues in a manner that's usually not applied to this type of problem.

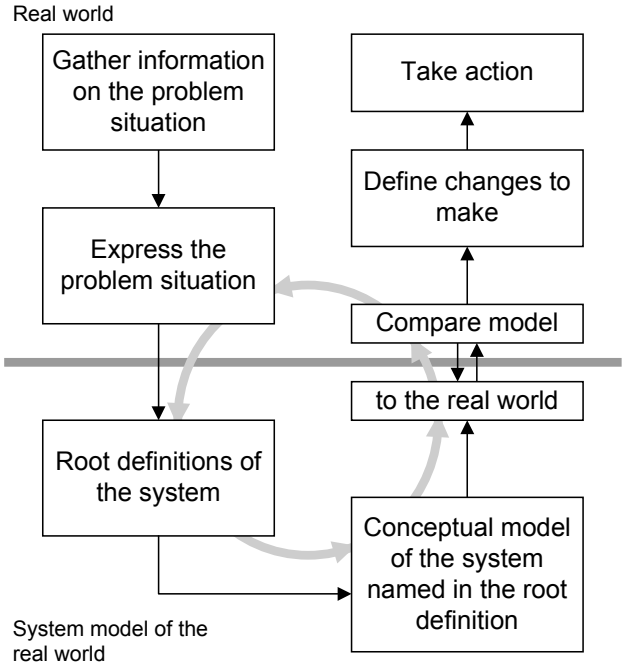

Figure 1: The Soft Systems Methodology as a problem-structuring method

As shown in Figure 1, PSMs work by analysing the real-world situation, building a conceptual model of it, comparing the real-world situation to the model and applying corrections if necessary, and then making any changes that are indicated by the model. In the words of one of the originators of PSMs, they represent "an organised version of doing purposeful 'thinking'" [6]. This sort of thing really appeals to the way that technologists think and work, because it follows a very careful, systematic approach to dealing with an issue.

What PSMs do is act as forcing functions for designs, making participants consider all manner of environmental factors before they begin and only allowing them to decide on concrete solutions towards the end of the design process. In other words providing that the participants follow the design process correctly they're forced to choose a solution that actually addresses the problem rather than just picking a silver bullet out of a hat. PSMs present the exact opposite of the process involved with the Inside-Out Threat Model.

3. APPLYING THE SOFT SYSTEMS METHODOLOGY

Explaining how the Soft Systems Methodology (SSM) works is best done by walking through an example of how it might be applied in practice, in this case applied to the problem of users interacting securely with an online service like a bank or an online store. This is something that we haven't really figured out how to do yet, or more specifically it's something for which we have endless numbers of technology-based proposals for solutions but nothing that really works very well in practice, making it a useful known-hard problem to apply the SSM to.

3.1 Finding Out

The first phase of the SSM is called Finding Out, and consists of discovering the scope of the problem and its environment. The Finding Out stage consists of two sub-stages, the external and the internal Finding Out stages. The external Finding Out stage involves going out and asking everyone that'll be involved in using, deploying, administering, and paying for the system, or in problem-solving jargon the stakeholders, what they consider the issues to be. This is an information-gathering stage that involves acquiring enough information from the stakeholders to build a general picture of what the problem that's meant to be solved actually is. It's important not to skip this stage (no matter how superfluous it may seem) because participants typically don't know it nearly as well as they think they do. For example in one study, in which the participants reluctantly went through the Finding Out phase to appease the external observers that were present, they ended up gathering twenty-two pages of material that painted a rather different picture of the problem than they had initially assumed [7].

Once the field work has been completed and used to obtain any required data from the external Finding Out stage, the next step is the internal Finding Out stage. This takes the information that's been gathered and uses it to build an (unstructured) picture of the problem to be solved. There are a variety of ways in which this Finding Out phases can be carried out, but one quite usable form breaks things down into three related analysis steps. The first step involves identifying the roles of the participants in the system, generally referred to as 'clients' in SSM terminology. The second step involves defining the social environment such as social rules, values, and norms of behaviour in which the problem to be solved is situated, referred to as the 'social system' in SSM terminology. Finally, the third step involves examining the political environment in which the system has to operate, identifying 'commodities' in SSM terminology, authority, political, and legal constraints and how they're applied and transmitted.

In some problem situations these issues can become very thorny, and a long, long way removed from anything technology-related. Consider the problem of Internet voting. Apart from the very obvious environmental requirement that users have to trust the system, the loser in the election also has to be convinced that they've lost. In the case of a hotly-contested election such as in the US in 2000 the entire election process (and by extension the government of a country) can be derailed if the system isn't able to provide convincing evidence not so much that the purported winner really won (they're unlikely to want to challenge this result) but that the purported loser really lost. This threat-model view is more or less the inverse of the electoral view, which only concerns itself with who won.

Note how just this initial process already differs radically from the traditional security approach of deciding on a particular solution like SSL/TLS or digital signatures and only then trying to figure out how to apply it, with issues such as asking whether it can actually work in this environment being left to post-mortem

analyses after deployment. SSM (and PSMs in general) make this background analysis a fundamental step in the problem-solving approach, forcing participants in the process to think about the environment in which their (potential) solution has to operate. This prevents the automatic application of the traditional Inside-Out Threat Model, both because it now becomes hard to justify blindly applying it and because participants won't get to that particular step until much, much later in the SSM process.

For the case of bootstrapping secure communications, typical roles involved in the process are the end users, the system administrators (meaning the people who administer and control the system and not just the IT system administrators or sysadmins in charge of running the actual equipment, a group of people that ranges from those with direct influence like company directors and managers through to ones with indirect influence like company lawyers and marketing people), the system developers, and (obviously) various types of attacker.

In terms of the environment in which the system has to operate, the end user just wants to get things done and doesn't want to have to take a series of night-school classes just to be able to use a site's logon mechanism (this particular issue is very hard for geeks to understand, since if they're able to eventually figure it out then absolutely anyone should be able to figure it out), they expect things to happen automatically without requiring tedious manual intervention, they generally have little awareness of security threats (if it says "bank" on the sign then it's a bank and there's no need to go out and perform a series of background checks to verify this), and they want to be able to authenticate from work, from home, and from an Internet café in Kazakhstan using whatever mechanism is most convenient.

System administrators (the technical ones in this case) want to go with whatever requires the least amount of work for them. The middlemen (network providers and ISPs) want no part in anything, they just provide the tubes and collect rent for them. Corporate-level administrators want to spend as little money as possible, and their primary security concern is "will this affect our stock price", or in some cases "will this appear on the front page of *$national_paper*". The marketing people are more interested in the perception of security than actual security (it's their job to convince customers/users to come in, and that requires creating the perception of a safe environment). Finally, the lawyers are worried about legal liability, regulatory constraints, and all the other things that lawyers are paid to worry about (this case tends to blend with the next step, examining the political environment in which the system has to operate).

Finally, at the political level, there are compliance and regulatory constraints like PCI-DSS, consumer protection laws, computer crime laws, the general reluctance of law enforcement agencies to pursue computer crime, and various messy cross-jurisdictional issues such as the fact that even if your current physical environment is one where X is the norm, your logical environment may be one where Y is the norm (a participant in one problem-structuring exercise described this situation as "like going to a corner dairy in Pakistan and being fed pork rinds"). Consider for example a German tourist currently on holiday in Spain who goes to a hotel's web site to book a room for a few days, with the site being run through a cloud provider in Ireland. The 2006 EU Data Retention Directive requires that all EU countries create a law requiring that data be retained for between six months and two years. In Germany it's six months, in Spain it's a year, and in Ireland it's two years. The German government is quite adamant that when German citizens are involved the data has to be deleted after six months. Spain claims that its law takes precedence. In Ireland you're breaking the law if you delete the data before two years are up [8]. This is the sort of situation in which, if participants don't get their lawyers involved fairly early in the problem-solving process, they can end up in deep trouble.

3.2 Formulating Root Definitions

Following the Finding Out stage, the next SSM step consists of Formulating Root Definitions, which define what's relevant in exploring the problem space. These are formalised using the mnemonic CATWOE, which stands for Customer, Actors, Transformation Process, Weltanschauung, Owner, and Environmental Constraints. The Customer is the beneficiary (or sometimes the victim) of the system, the Actors are the participants in the system, the Transformation Process is what the activity of the system operates on expressed in terms of the inputs and outputs of the transformation process, the Weltanschauung is the world view underlying the system ("Weltanschauung" is a German word that's usually translated as worldview, although that's something of a simplification of its full meaning), the Owner is the person or people with the ability to stop the system (sometimes Customers, Actors, and Owners can overlap), and the Environmental Constraints are the constraints that the environment places on the system.

CATWOE isn't just an arbitrary categorisation but was built from real-world experience with observing what people were and weren't taking into account in the problem-solving process. In particular SSM practitioners found that people tended to omit both Actors and Owners because they were "too obvious to be noticed" and so they were never considered as part of the SSM process [5]. By explicitly requiring them to be specified as part of CATWOE, SSM ensures that they're taken into account during the problem-solving process.

Going back to the Finding Out results, it's now possible to create the necessary Root Definition using CATWOE. In most designs involving security, the role of the Customers, Actors, and Owners are fixed: The Customer is the user, the Actor is the organisation that's running the system, and the Owner is the attacker. This leaves the Transformation, Weltanschauung, and Environment to be resolved. Since CATWOE is a mnemonic used to help remember what's involved and not a strict ordering of operations, it's not absolutely necessary to go through the process in the order implied by the mnemonic. In particular for security modelling it's often easier to leave the Transformation step until the end, since it's heavily influenced by the Weltanschauung and Environment. This means that the remaining TWE steps would be done as WET.

Continuing the process of addressing the sample problem given earlier, the Weltanschauung of the users (or at least as it's typically perceived by security geeks) is that the users trust too much while the security geeks would be seen by the users as trusting too little. In addition if something goes wrong then the users regard the system as being at fault and not themselves, and specifically they consider that it's the system's job to protect them and not their job to invest massive amounts of effort (far beyond anything required in the real world) to stay secure.

The Environment consists of unreliable (both in the sense of availability and of resistance to attack) networks, the general need

to run things over HTTP because of firewalls, a need to make a profit at some point (that is, it doesn't make much sense to spend $1M to protect $5 unless you're a government department) and by extension the fact that most organisations see security as a "hygiene issue", it's something that's good to have but that doesn't really add any value since you don't directly make money off it, a user endpoint in an unknown state, the fact that the user is geographically separated from the systems that they'll be interacting with, and the fact that we have no direct physical channel to the user (users generally trust things involving physical presence more than they do the more nebulous presence of a site on the Internet, and a direct physical channel could be leveraged to help secure the Internet channel, as some European banks do by bootstrapping Internet authentication from bank branch visits or information distributed via postal mail).

The Transformation is fairly straightforward, we want to go from an untrusted to a trusted state, or more abstractly we want to solve the problem of trusted knowledge distribution.

Finally, we have the Root Definition, which is that we want to validate customers using systems that we don't control over a network that we don't control against systems that we do control in a situation where it's advantageous for attackers to manipulate the process, and it all has to be done on a shoestring budget (there's a good reason why these sorts of things are called "wicked problems").

The above is only one particular way of approaching things, which is why Figure 1 shows this stage as being part of a very iterative process. At the moment we've framed the problem from the point of view of the defender. What happens when we look at it from the attacker's perspective? In other words rather than looking at what the defenders are trying to achieve, can we look at what the attackers are trying to achieve? About a decade ago the primary motivation for attackers would have been ego gratification, whereas today it's far more likely to be a commercial motive. On the other hand for targets with little directly realisable financial value to attackers it may be that the only motivation for attackers would be either ego gratification or espionage in the case of certain government and industry targets.

With this alternative view the Customers are now the hackers and/or the people paying them, the Owners become the defenders, the Actors become the people working with and using the system (which includes the bad guys), and the Transformation and Root Definition are restated in terms of the attackers' goals rather than the defenders goals. A typical root definition for a financially motivated attacker might be that they want to obtain (if it's a phishing attack) or extract (if it's a data theft attack) access-control and authorisation information without the defenders being aware of the loss so that the information can then be exploited at leisure. As with the defenders' root definitions, this can then be moved on to the next stage.

3.3 Building Conceptual Models

The next SSM stage involves Building Conceptual Models. This takes the Root Definition and uses verbs to describe the activities that are required by the Root Definition. It's important that the model contains a monitoring mechanism (in SSM terms this is the 'monitoring and control' system) that monitors its effectiveness (is it doing the right thing?), its efficacy (does it work properly?), and its efficiency (is this the best way of doing this?). Two other options that are sometimes added to the monitoring mechanism for the general-purpose SSM are ethics (is it morally sound?) and elegance (is it beautiful?). These are appropriate in some situations in which the SSM is applied, but are generally unneeded here: it's hard to think of how one would create an unethical encryption mechanism (although some uses of DRM have been suggested as possible candidates), and arguing with geeks about the aesthetics of a technical solution would be like wrestling with a pig in mud where after awhile you realise that the pig is enjoying it. For these reasons it's best to focus only on the first three 'e's, effectiveness, efficacy, and efficiency.

One important factor to take into account when building the model is that it must use only those terms that are present in the Root Definition. So for example one part of the model could specify that "communications with users (customers) cannot infringe on PCI-DSS or other regulatory controls", "attackers (owners) cannot be allowed to have knowledge of communications", and "attackers (owners) can pretend to be users (customers) or administrators of the system (actors)", but it couldn't specify that "users (customers) will use smart phones as authentication tokens" because this appears nowhere in the root definition. Without this constraint it becomes far too easy to start inserting specific instances of real-world systems into the model, micromanaging it to death (or at least to unworkability) before it can be passed on to the remaining steps in the SSM process.

For the secure communications bootstrap problem, one approach, which requires very little in the way of actual security technology, might be to simply convince the customers that they're secure without doing much else, thus meeting the needs of at least some of the stakeholders (the marketing people, the sysadmins, and probably management) even if it may not satisfy some of the others (notably the lawyers). One way of convincing the customers that they're secure might be to refund their money in the case of fraud, allowing you to claim that "no customer has ever lost money through fraud", not because there isn't any fraud but because when there was some, the customer didn't have to carry the cost. This is more or less the system used by banks that issue credit and debit cards when they dump liability on merchants, and in this case is a situation where applying the full five 'e's, specifically including ethics, would provide a better model than using just the basic three 'e's.

Looking at this from another point of view, can we analyse the problem from the perspective of the prevent/detect/correct approach that's sometimes applied to situations like this? The problem can't readily be prevented since there's no direct control available over the client environment or the network (the few attempts by banks to force customers to use a particular PC configuration in order to engage in online banking have resulted in little more than extensive negative media coverage for the banks), we can only take limited steps to detect problems through fraud-monitoring techniques, and therefore our only real option is to step in at the correction stage by refunding the customer's money in the case of any losses.

One mechanism that's been proposed for dealing with the risk-avoidance that organisations like to engage in at this point is by trying to put a financial cost on the value of security. This is a real problem with many organisations (or at least the people who run them) who don't think ahead too far, being concerned mostly with the cost right now rather than how much they should spend for future security. Unfortunately this approach has historically proven very difficult (if not impossible) to implement, being

subject to Geer's Law, after security philosopher Dan Geer, "Any security technology whose effectiveness can't be empirically determined is indistinguishable from blind luck" [9].

For the alternative model that looks at the situation from the financially-motivated attacker's point of view (in which, obviously, the 'e' of ethics doesn't have much place), the monitoring mechanism is fairly straightforward and is derived from the attacker's dual goals are of escaping detection (or at least prosecution) and obtaining valid, fresh financial information and using it before the defenders have time to react. The monitoring mechanism for the validity and usefulness of the financial information that's being obtained is more or less built in, since the attackers have direct feedback as to whether the account credentials that they've stolen are current and valid. The monitoring mechanism for evading prosecution is less obvious, but checking whether botnets and servers are being shut down by defenders provides some level of feedback. The efficiency in this case isn't usually a major consideration for attackers since the resources being consumed are someone else's, and efficacy concerns are addressed by applying the attack in quantity rather than quality.

Trying to analyse the attackers' situation through the application of standard economic models leads to very odd results. The attackers are using other people's resources, running over other people's bandwidth, financing their attacks with other people's money (stolen credit card and bank account credentials), and if something goes wrong then someone else gets blamed. Conventional economic theory doesn't really have any way of representing something like this because theft isn't normally a part of standard economic models. For example, externalities theory, which looks at costs or benefits that are spilled over onto third parties, assumes honest trade, not theft. One effect of this unusual situation is that even the most ineffective and inefficient attacks are still worthwhile for attackers, because someone else is carrying all of the costs. So if standard analysis tools like conventional economic theory can't deal with this, is there a way of using the SSM to analyse this problem?

Unlike the defenders, who as Geer's Law points out often have no way of determining how successful they've been, the attackers have a very easily quantifiable success metric, either the number of accounts looted or how much their employers are paying them for their work. Since the intended goal of using the attacker's model is to help analyse things from the defender's point of view, this immediately points to two defence strategies that target both of these success metrics. On the one hand we can try and make it much harder for attackers to know which accounts are valid and which aren't, perhaps by seeding financial data with large numbers of tarpit accounts that appear valuable but aren't, causing them to waste the one resource that they can't get for free, their own time, on no-value accounts, and on the other hand we can try and target the higher-level financial controllers rather than the low-level, disposable foot soldiers (more fully exploring these alternative paths goes somewhat beyond the scope of this paper).

3.4 Using Models

The next SSM stages involve Using Models, which take the model and look to see how it applies to the real world. This can create the iterative situation illustrated in Figure 1 in which an attempt to apply the model indicates that it has some shortcomings that need to be addressed, requiring going back an earlier stage in the SSM process to redo a definition or portion of the model.

One of the simplest ways to handle this stage is to "operate" the system on paper, checking how well the model copes within the framework of the Root Definitions. This process is particularly appropriate for software developers, who often use a similar process of mental symbolic execution of code during the coding process [10][11][12][13].

One useful feature of SSM's built-in monitoring mechanism is that it can help avoid a situation in which a solution converges on a local maximum that may not actually be a particularly good overall solution. By explicitly building an evaluation mechanism into the overall design process, SSM tries to avoid having a design converge on an inviting but suboptimal solution.

An example of an iteration might be the earlier requirement that "attackers (owners) cannot be allowed to have knowledge of communications". Quite frequently, what's actually required in this case isn't confidentiality but authorisation. For example since the numbers on credit cards are used as authorisation tokens it's necessary to keep them secret, but if they were used as part of a robust authorisation mechanism then there'd be no need to keep the credit card number secret because even if the information was sent in the clear an attacker would still have the authorisation mechanism to contend with. So by changing this part of the model to "attackers (owners) cannot undetectably manipulate the communications" and then operating this new variant on paper, it's possible to see whether the change improves (or worsens) the overall situation.

As Figure 1 shows, these last two stages of the SSM are a very iterative process. Much more so than for the defenders, the attackers would use this stage to insert a typical attack into the model, run through it to see how well it works, and then try again if it doesn't (although given that any attack will be successful if you throw enough of someone else's resources at it, and the attackers have little shortage of those, the number of iterations may be less than expected).

One way of using the concept of operating the model as a simulator is to apply the paper-execution process while changing some of the input parameters [14]. For example what happens if you change some aspect of the Weltanschauung or the Environment? Does this make things harder or easier? This type of exploratory evaluation can help identify situations where the solution to the problem isn't some magical application of technology but to modify your underlying assumptions about the nature of the problem, redefining it in such a way that it's more amenable to solution. An example of this occurs with the problem of securely sending email between different branches of a company, where redefining the underlying assumption from "everyone has to have email encryption on their desktop" to "we need to securely get email from branch A to branch B" moves the potential solution from the near-impossible task of deploying email encryption to every desktop to the much simpler one of using STARTTLS [15] or a corporate S/MIME or PGP gateway. Another example of changing the model was the one given earlier of switching from confidentiality as a communications goal to authentication/authorisation as a communications goal. Even if it seems like a lot of work, the process of using the model on different sets of input data can provide real insights into the true nature of the situation.

You can also use the simulator runs to try and explore what happens when some of the stakeholders have conflicting goals that are identified during the Finding Out phase. By running the two different viewpoints through the simulator (or changing the simulator's parameters, depending on the level at which the differences take effect) you can explore which of the different options produces the best (or perhaps the least bad) result.

This iterative operation may mean going back to an even earlier stage in the SSM processing, requiring that you gather additional information on the problem situation that may not have been considered relevant the first time round. For example your solution may be one that requires the middlemen (network providers and ISPs) take a more active role in the process. Seeing whether this is in fact feasible or not would require going back to the Finding Out phase to gather further information.

This can even involve discarding an initial requirement if it's now been found to create insurmountable obstacles. For example there might be a particular feature that was added to the design because it was felt that it'd be nice to have it present (this is another case of a hygiene-issue feature mentioned earlier that everyone agrees is a good idea even if there's no compelling argument supporting it) or something that was added for political or marketing reasons that now turns out to create an insurmountable obstacle to coming up with a solution. Something like this can be discarded and the SSM process re-run to determine whether it really is as necessary (or even just desirable) as it may have first appeared.

3.5 Defining Changes and Taking Action

The final steps, Defining Changes and Taking Action, are pretty self-explanatory and involve making changes in the real-world system based on what's been defined by the model. The Defining Changes step may have identified a number of changes that *could* be made, but that doesn't necessarily mean that they *should* be made, which is why it's a separate step from Taking Action. Defining Changes identifies the changes that are worth trying, meaning that they're both desirable and feasible, and then finally Taking Action puts them into effect. As with the earlier use of Actors and Owners, CATWOE's explicit inclusion of the Weltanschauung within which the system has to operate ensures that constraints and conditions imposed by the environment can't be easily ignored. This is particularly important for technological systems because it's something that geeks often ignore.

Note that throughout this entire discussion, the actual technology that might be applied hasn't really cropped up yet. In fact for the case of the example problem that's used here, to the distress of geeks everywhere, the best action for the defenders to take may require the skills of the marketing department more than those of the IT department. Applying a PSM isn't guaranteed to produce the results that geeks would prefer, but rather the results that arise from the information that's gathered and the model that's built with it. In practice this requires a fairly strong-willed coordinator to resist the intense desire of the geeks to "solve" the problem with their favourite technology, a real-world problem that'll be covered in more detail in a future paper.

Applying a PSM may also not produce the results that a manager, believing that his company has the best programmers in the world and that surely they can come up with a software solution to this problem, would prefer (this is the so-called moon-ghetto metaphor, "if we can put a man on the moon then we should be able so solve the problem of inner-city ghettos" [16]) would prefer. Working through a PSM lets them derive the fact that there really is no technical solution to a problem and it'll have to be addressed another way. Having them to discover this form themselves is an important step, since they may not believe it if anyone else tries to tell them.

4. CONCLUSION

This paper has presented a new mechanism for addressing computer security problems whose primary contribution is the fact that it's a formal problem-structuring method that forces participants to think about the problem that they're solving rather than applying the more usual approach of leaping in with their favourite technology and hoping that, like the mythical silver bullet, it'll end up doing what they want. Problem structuring methods like the SSM described here provide a problem *structuring* method and not a guaranteed problem *solving* method. As the introduction pointed out, the problem may be a genuinely unsolvable one, with the only possible tradeoffs being between an awful solution and a less awful one. In this case an approach such as having the marketing people convince users that they're safe and refunding their money in the case of fraud may indeed be the best (meaning the least awful) one.

5. ACKNOWLEDGMENTS

The author would like to thank Bob Williams, members of the Auckland Information Security Interest Group (ISIG), attendees of Kiwi Foo, and participants in the 2011 New Security Paradigms Workshop for their input into this work.

6. REFERENCES

[1] "So Long, And No Thanks for the Externalities: The Rational Rejection of Security Advice by Users", Cormac Herley, *Proceedings of the 2009 New Security Paradigms Workshop (NSPW'09)*, September 2009, p.133.

[2] "Engineering Security", Peter Gutmann, to appear.

[3] "The Use of Soft Systems Methodology in Practice", John Mingers and Sarah Taylor, *Journal of the Operational Research Society*, **Vol.43**, **No.4** (April 1992), p.321.

[4] "Soft Systems Methodology: A Thirty Year Retrospective", Peter Checkland, *Systems Research and Behavioral Science*, **Vol.17**, **No.S1** (November 2000), p.S11.

[5] "Soft Systems Methodology", Peter Checkland, in "Rational Analysis for a Problematic World Revisited (2nd ed)", John Wiley and Sons, 2001, p.61.

[6] "Using Soft Systems Methodology in the Design of Information Systems", John Mingers, in "Information Systems Provision: The Contribution of Soft Systems Methodology", McGraw-Hill, 1995.

[7] "Soft Systems Methodology in Action", Peter Checkland and Jim Scholes, John Wiley and Sons, 1999.

[8] "Cloud Computing Roundtable", Eric Grosse, John Howie, James Ransome, Jim Reavis and Steve Schmidt, *IEEE Security and Privacy*, **Vol.8**, **No.6** (November/December 2010), p.17.

[9] Paraphrased from the analysis first presented in "Information Security: Why the Future Belongs to the Quants", Daniel

Geer, Kevin Soo Hoo and Andrew Jaquith, *IEEE Security & Privacy*, **Vol.1**, **No.4** (July/August 2003), p.32.

[10] "Towards a theory of the cognitive processes in computer programming", Ruven Brooks, *International Journal of Man-Machine Studies*, **Vol.9**, **No.6** (November 1977), p.737.

[11] "Change-Episodes in Coding: When and How Do Programmers Change Their Code?", Wayne Gray and John Anderson, *Empirical Studies of Programmers: Second Workshop*, Ablex Publishing Corporation, 1987, p,185.

[12] "Cognitive Processes in Software Design: Activities in the Early, Upstream Design", Raymonde Guindon, Herb Krasner, and Bill Curtis, *Proceedings of Human-Computer Interaction (INTERACT'87)*, Elsevier Science Publishers, 1987, p.383.

[13] "A Model of Software Design", Beth Adelson and Elliot Soloway, in *The Nature of Expertise*, Lawrence Erlbaum and Associates, 1988, p.185.

[14] "Soft Systems Methodology", Bob Williams, December 2005, `http://users.actrix.co.nz/bobwill/-ssm.pdf`.

[15] "SMTP Service Extension for Secure SMTP over TLS", RFC 2487, Paul Hoffman, January 1999.

[16] "Intellectualizing about the Moon-Ghetto Metaphor: A Study of the Current Malaise of Rational Analysis of Social Problems", Richard Nelson, *Policy Sciences*, **Vol.5**, **No.4** (December 1974), p.375.