Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 137 (2005) 225–242
www.elsevier.com/locate/entcs

Breaking the Model: Finalisation and a Taxonomy of Security Attacks

John A. Clark, Susan Stepney, and Howard Chivers

Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK

Abstract

It is well known that security properties are not preserved by refinement, and that refinement can introduce new, *covert*, channels, such as timing channels. The finalisation step in refinement can be analysed to identify some of these channels, as *unwanted finalisations* that can break the assumptions of the formal model. We introduce a taxonomy of such unwanted finalisations, and give examples of attacks that exploit them.

Keywords: Finalisation, observed system, security model assumptions.

1 Introduction

Refinement is the standard process of transforming a specification into executable code. A refinement can be proved correct, meaning that all the functional properties of the abstract are present in the concrete.

It is well known that security properties are not necessarily preserved by classic refinement [14]: widening the precondition may allow new, Trojan, behaviour; peculiar resolutions of non-determinism may be used to leak secret information. Additionally, as discussed in this paper, *finalisation* refinements may break the assumptions of the formal model, and allow information to leak.

$$\begin{tikzcd}
& A \arrow[r] & A' \arrow[r] & A'' \arrow[r] & \cdots \arrow[dr] & \\
G \arrow[ur] \arrow[dr] & & & & & G' \\
& C \arrow[r] & C' \arrow[r] & C'' \arrow[r] & \cdots \arrow[ur] &
\end{tikzcd}$$

Fig. 1. The relational model of refinement

$$\begin{tikzcd}
& A \arrow[dd, leftrightarrow] & A \arrow[r] \arrow[dd, leftrightarrow] & A' \arrow[dd, leftrightarrow] & A \arrow[dd, leftrightarrow] \arrow[dr] & \\
G \arrow[ur] \arrow[dr] & & & & & G' \\
& C & C \arrow[r] & C' & C \arrow[ur] &
\end{tikzcd}$$

Fig. 2. A simulation, used to prove refinement

2 Finalisation and refinement

2.1 The relational model of refinement

The relational model of refinement is cast in terms of a many-many relation between an initial and final global state. Refinement is a relationship between an abstract program expressed in some abstract world that captures this initial-to-final relation, and an equivalent concrete program expressed in the concrete world [11]. See figure 1.

In this relational model of refinement, the refinement relationship that must hold between the two programs reduces to the subset relation between the two global to global relations (whilst maintaining totality, so the empty program is not a refinement). That is, it is a resolution of non-determinism, coupled with a change of data representation.

The existence of the subset relation is difficult to prove, being expressed over general sequences of operations, so a (forward or backward) *simulation* is introduced, reducing the global proof obligation to three simpler ones: an initialisation, a finalisation, and one for a single operation only (figure 2).

2.2 Finalisation and observability

The last part of the process, moving from the program world (be it abstract or concrete) to the global world, is called *finalisation*. The finalisation step defines what properties of the implementation world are *observable*. Only the global world is observable in the original specification relation. (Anything not observable in the program world is there merely for implementation convenience.)

In a specification language like Z [24] [3], this simple relational model is given additional structure so that parts of the relational state can be used to model internal Z state, inputs, and outputs. Also, Z operations can be partial relations. The refinement proof obligations become correspondingly more complicated [24], primarily by the introduction of an applicability (precondition) law to handle partial relations.

The traditional Z refinement rules in [24] make certain simplifying assumptions [27] [25], leading to certain restrictions. In particular, the rules do not permit any refinement of the inputs or outputs, or any observation of state except by inputs and outputs. (Technically, nothing at all is observed until the finalisation step, at the end of the computation. But yet another simplifying assumption, about the way the outputs are embedded in the relational world, allows the sequence of *independent* outputs to be observed incrementally, as each one happens, since nothing that occurs subsequently can change them.)

Under these assumptions, and for forward simulations [24], the finalisation step reduces to the identity transformation on the outputs (the program outputs and the global world outputs are always identical, at both the abstract and concrete levels), and the null function on the internal state (no internal state is observable). Hence there is no explicit finalisation proof obligation in the classic Z refinement rule [24].

When these simplifying assumptions are relaxed, however, more general refinements are possible. For example, consider an abstract program that has an operation that outputs a set, where the corresponding concrete operation outputs a sequence. If the finalisation step for the abstract model is the identity function (the set is observed), then the corresponding finalisation step for the concrete program extracts the range of the sequence (collapsing the sequence to the observed set).

In the refinement process, the most abstract model usually has an identity output finalisation (if it models the outputs in the most abstract way, that of the global model). But in a series of stepwise refinements, the concrete model in one step becomes the abstract model in the next, and so both models can soon have non-trivial finalisations.

The more general refinement rules in [25] provide the proof obligations for output refinement. There is an explicit finalisation proof obligation, and a need for a retrieve relation between abstract and concrete outputs.

2.3 Finalisation glasses

An intuitive explanation of the meaning of finalisation is as follows. Finalising the state can be thought of as "breaking open the device" and observing its state, through "finalisation glasses" (or finalisation spectacles). Finalising

$$\begin{tikzcd}
A \arrow[r] \arrow[d, leftrightarrow] & A1 \arrow[r] & \cdots \arrow[rr] & & Am \arrow[d, leftrightarrow] \\
C \arrow[r] & C1 \arrow[r] & \cdots \arrow[r] & Cn\text{-}1 \arrow[r] & Cn
\end{tikzcd}$$

Fig. 3. Generalised m:n simulation

the outputs can be thought of observing the outputs, through “finalisation glasses”. These finalisation glasses transform the raw program values into the corresponding global specification values. In the earlier example, they transform the concrete sequence into the global set. (As they transform mundane brickwork into the Emerald City [4].)

Finalisation solves the old paradox of why the clock that is 5 minutes slow (hence never right) is better than the stopped clock (right twice a day). The first clock has a simple finalisation that can be applied to it: add five minutes to the displayed time. The second has no finalisation that produces a useful result (without recourse to a second clock).

2.4 Other models

2.4.1 Schellhorn’s Generalised forward simulation, and breaking atomicity

Generalised forward simulation [22] allows arbitrary m:n diagrams between abstract and concrete operations (figure 3), rather than the 1:1 diagrams assumed above, with no requirement for the simulation to hold part way through. So it is possible to have a single, atomic abstract operation refined by a compound sequence of concrete operations.

In the 1:1 relational model it is possible to finalise at any point. For a correct refinement in the m:n model, finalising part way through the sequence of concrete operations (for example, by removing power from the device so that it cannot continue functioning) should result in a state that corresponds to no abstract change. However, we can also use this model to investigate the consequences of breaking of the atomicity assumption.

2.4.2 CSP refinement

CSP [12] [20] has a rather different model of computation, in that there is no explicit finalisation step, no finite end point to the computation: processes can continue engaging in events without limit. But refinement in CSP is still dictated by what one chooses to observe.

In the simplest case, of *traces refinement*, the chosen observations are just the traces of events, and the refinement condition reduces to subsets of traces, giving behaviour-equivalent processes. More sophisticated observations can be made, each restricting the set of concrete processes that are allowed as valid refinements of the abstract. So *failures refinement* additionally gives deadlock-equivalent processes. *Failures-divergences refinement* additionally gives livelock-equivalent processes. In all these cases, the processes must have the same alphabets, so there is no input/output refinement. ([8] augments a relational state model with an observed component that records these more sophisticated observations, allowing these form of process refinement to be incorporated into the relational model.) There are more yet more sophisticated observations possible, for example, including timing [19] or fairness [17].

In all these cases, however, the key thing is that the process is defined by what we chose to *observe*. The fact of whether a system is a refinement of another depends solely on those observations. So the arguments we make in the rest of the paper also apply to CSP-style refinements.

3 Unwanted finalisations in practice

3.1 Unwanted finalisations as covert channels

The specification defines what is intended to be observable about the system, and what is not. Parts of the system may be intended to be unobservable, often for security reasons (for example, a secret cryptographic key). The specifier captures this property in terms of a finalisation.

Once the refinement model constraints have been relaxed to include non-trivial finalisations, it becomes easier to see how certain covert channels arise. It may be possible to apply finalisations other than the intended one, in order to observe different information. Such finalisations may not provide formal refinements of the original models, but are important because they may be performable in practice.

In the sequence/set example, what if you take off your finalisation glasses, and observe the raw values actually output? You see the concrete sequence, which has more information, because it orders its elements. If you can use that order to deduce something about the internals of the system that are supposed to be secret, you have observed a *covert channel*. You have performed an *unwanted finalisation* (unwanted by the system owners, that is).

3.2 The maximal identity finalisation, and the Real World

The identity finalisation (no finalisation glasses) enables the most data to be seen, within the modelling assumptions. By the time a very concrete level of model (or physical implementation) has been reached, there may be a remarkable amount of data visible to other finalisations: not simply a bit-stream, but also real world information outside the formal model, such as timing information, or power consumption. Also, the actual finalisations possible depend on the details of the implementation technology chosen.

Note that the maximal identity finalisation result is the case only for classical computation. Quantum finalisations are not composable: there is no such thing as the maximal identity finalisation. The observation "collapses the wavefunction", rendering complementary variables no longer observable; a different observation might have given different information, and so have allowed different inferences.

3.3 Variation in what you see

One might say, so do not allow output refinement, do not allow non-trivial finalisations. Yet an actual computing device does not output a set of abstract values in some timeless domain; it outputs electrical signals over time. Refusing to allow output refinement requires either polluting the abstract model using concepts at the level of electrical signals, or fudging the issue by implementing the refinement outside the mathematical analysis.

So, many covert channels can be considered as unwanted finalisations, as being able to observe more in the implementation than is specified at the most abstract level. This realisation can be used to help analyse covert channels, and to prevent them. In the end, it is *observations* that cause breaches of security. Information that affects no agent cannot cause any harm. *Finalisation* formalises the notion of observation by external agents interacting with the system. The adoption of various observation strategies can, therefore, be viewed (modelled) as *applying finalisations.*

Arguing about the security of modern systems grows increasingly hard, and there have been numerous surprises over the past decade in particular (timing attacks [15], power attacks [16], etc). Systematic analysis is needed. We provide a simple taxonomy, motivated by the formal refinement models presented earlier.

4 A taxonomy of unwanted finalisations

With any system there is an obvious finalisation: that intended by the specifier. This may give sufficient security (or it may not). It is, however, a choice, and comes with its own assumptions about the system and its context. Considering other choices of system elements and context provides an interesting and informative means of highlighting possible unwanted analyses.

4.1 Varying the "finalisation glasses"

The **intended** observation is with the intended finalisation glasses (for example, the discrete values produced by a crypto-algorithm). We can consider the intended finalisation to correspond to the identity applied to the outputs of the most abstract specification. **Unintended** observations vary the finalisation glasses, and can observe discrete properties (for example, page faults, interrupts, i/o buffers), or analogue properties (for example, power, timing, RF).

The intended observations may be **direct**, of outputs from single abstract operations. Or they may be **enhanced**, involving some degree of post-processing, making use of information from *sequences* of abstract outputs, for example, cryptanalysis on millions of ciphertext blocks. Unintended observations often also exploit an enhanced viewpoint.

We may take a **single** viewpoint (for example, power information) or **multiple** viewpoints (intended, power and timing together).

An **invasive** observation requires the analyst to force an additional finalisation channel, for example, dropping a hardware microprobe onto a circuit track. Otherwise the observation is **passive**.

4.2 Varying the system being finalised

We may observe a **standard** system instance (operating in the intended manner). Many innovative analyses have emerged, particularly over the past decade, that involve a **perturbed** system (operating in some unintended manner).

A perturbed instance may arise due to **invasive** or **passive** means. Fault injection using an ion gun is clearly an invasive attack. A physical flaw could arise in the system due to manufacturing inadequacy or simply as a result of wearout (things do sometimes just break).

Perturbed instances can result in **unreachable states** (as far as the model is concerned). Atomic abstract operations may be implemented by a sequence of concrete operations. Disruption part way through the concrete sequence of operations (breaking abstract atomicity) may cause the system to end up in a state that is unreachable in terms of the abstract model. This may permit all manner of unwanted finalisations (immediate or subsequent). This applies not only to general operations but also to initialisation (for example, interrupting trusted startup). Lower level perturbations (for example, bit flipping) can result in unreachable concrete states.

A particular source of system variation is the chosen **initialised system state**. This initial state may simply be insecure (for example, unfortunate default passwords or permissions).

We may choose to observe the operation of a **single** instance of a system (for example, a single smart card) or **multiple** instances. The single instance is the usual user view; an analyst may well prefer the multiple instance system, allowing differentiated analyses. Instances of standard and perturbed systems may be analysed together.

Multiple instances may be **homogeneous** or **heterogeneous**. Collections of systems may play both roles at the same time (instances may be homogeneous from the intended viewpoint, but heterogeneous from an unintended viewpoint). Heterogeneity may be engineered deliberately (for example, maliciously by fault injection, or for commercial reasons such as running on different hardware platforms), or occur naturally (for example, because no two processors of the same type are precisely the "same").

4.3 Varying the environment

Many systems have specified environmental ranges for operation. For example, smart cards have power supply specifications and operating temperature ranges. Each attribute may be **standard** (within specification) or **perturbed** (out of specification). Environmental variation provides opportunities for altering the finalisation; for example, digital circuitry operates differently at different temperatures. Since we are generally dealing with ranges, there are also possibilities for variation even *within* specification.

We may choose to vary a **single** attribute, or **multiple** attributes, of the environment.

Variation may be **passive** (for example, the climate is extremely cold), or **invasive** (for example, a power supply deliberately modulated in an unhelpful way, or the system deliberately heated).

We can be flexible in our interpretation of the environment. The above examples are expressed in terms of concrete environments. For an *abstract* model, the environment could encompass elements such as assumptions about operations of use, for example, that a system is subjected only to limited demands, or that the users understand the system sufficiently well not to breach security inadvertently.

4.4 Varying the refinement

Ideally, refinements are carried out formally and correctly. But there is always the possibility of **erroneous refinement** (for example, stack overflows).

It is not always feasible to perform a pure refinement: modelling assumptions have to be made at the abstract level that do not hold at a more concrete level, or in the implementation (for example, that voltage levels are precisely binary). One formal technique attempting to handle deviations from pure refinement is **retrenchment** [2]. An interesting open problem is how much can properties of interest be preserved under such circumstances.

4.5 Varying which system is finalised: higher order finalisations

Typically, the system being observed is the system being analysed. It is also possible to make higher order observations. For example, the analysis techniques themselves have standard and non-standard properties. One common analysis technique is meta-heuristic guided search (genetic algorithms, simulated annealing, etc) to find a potential solution. A guided search has a final result, but also has a *trajectory* (the path followed to reach that result).

Viewing the search process merely as *atomic* throws away huge amounts of information. Observing the search in action [7] can reveal far more. For example, a search by simulated annealing may move around a key space by considering moves one bit flip away from the current one. As the process "cools", some key bits become stuck at particular values. The relative times when particular bits become stuck can give a great deal of information about the underlying secret solution sought. Even if search-based analyses "fail" (do not find the solution), repeated runs may provide results whose *distribution* may allow the actual solution to be derived.

Analysis analogies can be found for other approaches too. For example, perturbing the mathematics in some way ([7] uses the term *problem warping*) and observing the results of searches can also give rise to new analyses. Thus, a higher order analogue of fault injection may apply. Due to the way meta-heuristic search proceeds, non-standard or highly perturbed cost functions typically produced better overall results [7].

Some, possibly many, of the unwanted finalisations identified earlier may have analogues when applied to the analysis processes. For example, what is the equivalent of "power analysis" finalisations for analysis processes? The systematic application of our finalisation taxonomy to analysis/search approaches may generate new analyses. This is currently under investigation.

5 Illustrating the taxonomy

In summary, the main points of our taxonomy cover the finalisation, the system, or its environment being viewed as an individual or a collection, using either a standard (intended), or a non-standard (unintended) view, the latter of which may be brought about passively or actively. We now give examples to illustrate our taxonomy. For presentation purposes we choose variation in the finalisation glasses, and number of systems under observation, as the main classifications, and annotate the specific examples with other aspects, where they are of a "non-standard" variety.

It is interesting to note that there are few such annotations, and even fewer multiple annotations. Published analyses tend to exploit only a single viewpoint, for example, unintended finalisations involving power *or* timing. There appears to be little in the way of multiple viewpoint analyses, where timing and power information, say, are used together. Given the considerable success of the single aspect analyses, availing oneself of multiple sources and exploiting correlations between them would seem a promising avenue to explore (if one is an attacker).

5.1 Intended Finalisation, single system

This is the most common and obvious viewpoint. One simply observes the normal abstractly defined operation of a single system instance. No clever tricks or sophisticated equipment are needed here. It is therefore crucial that the intended finalisation is secure.

Enhanced: Security-breaching finalisations may occur due to resource exhaustion. Suppose a directory can contain up to 1024 files. Some files may be highly classified, and so invisible to a lowly classified user. But the existence of n highly classified files limits the number of unclassified files that can be created to 1024-n. Any attempt to create more fails with "directory full". Deleting a highly classified file then allows a lowly classified file to be created. We have therefore the basis of a covert channel when observed over sequences of operations. Similar considerations apply to memory exhaustion, and file-locks.

Erroneous refinement. For systems such as cryptosystems a great deal of effort goes into ensuring that particular known (to the designers at least) finalisations are intractable. There remains the possibility of a "trapdoor" or flaw in the algorithm: the given specification does not capture the intent. A hashing algorithm may simply be flawed, allowing an easy break.

Enhanced, erroneous refinement: Although particular analyses may require trillions or more data points, it may be that knowledge of the ciphertexts corresponding to, say, one hundred very specific chosen plaintexts suffices to leak a particular key bit.

Perturbed system: [6] describes how fault injection on a crypto system (causing some internal state bit to flip) could be used to break RSA and other algorithms.

Perturbed user: Even when a system works as specified and is considered secure, lack of security understanding by the user may cause problems. There have been many examples of *social engineering* ("cognitive hacking") attacks, where the user is persuaded to carry out actions favourable to the attacker.

5.2 Intended finalisation, multiple systems

A cryptographic key's use may be limited on any specific smart card, but access to 10000 smart cards all with the same key may significantly affect cryptanalysis. Also, access to 10000 processors may radically affect the probability of a successful analysis, for example, the various distributed searches for prime factors for the RSA challenges.

Traffic analysis most naturally consists of observation of multiple systems. Even if the content of messages over a network is encrypted, analysis of network source and destination fields leaks information. (One could instead view this as a single networked system with multiple probes, distributed around the system.)

5.3 Unintended finalisation, single system

Many specifications assume that operations are *atomic*. In practice, different atomic operations may take varying amounts of time to compute. The most high profile exploitation of this has been Kocher's timing attack [15] on exponentiation; the analysis exploits the fact that the time taken to carry out exponentiation is data dependent (and the detailed form of that dependence does not need to be known).

There may be timing attacks on the finalisation operation itself. How long does it take to compute a finalisation that is more complicated than the identity? A directory listing operation invoked by an UNCLASSIFIED process might need to filter out the names of more highly classified files. Although the listing may output only UNCLASSIFIED files, the time taken to complete may depend on the presence of more highly classified files.

Analysis can measure power fluctuations during computation (between output events) [16], which might be correlated with something internally secret.

Consider the TENEX password attack (see, for example, [9]). The password is checked byte by byte, and the check aborts if an incorrect byte is found. If the password is located in memory that crosses a page boundary, a page fault is generated only if all initial bytes on the first page are correct. If such configurations can be engineered then an unintended finalisation that observes the paged faults is possible. Similar analyses may also apply to timing considerations if protocol message fields are validated incrementally.

Some safe locks require discs to be rotated to specific positions in a particular order to release the door. The modelling assumption is that the only observation is success or failure in opening the safe door. Film-goers, or bank robbers, reading this will realise that a safecracker using a stethoscope to listen to the tumblers when the discs are rotated is performing an unintended finalisation.

It is common to assume that analysts have ideal conditions for performing their finalisations, but we must be very careful in defining "ideal". A user may normally be unable to monitor the timing performance of a process in action at sufficient granularity to breach security. If other users access the system, however, this slows down the actual rate of execution, and could make timing analysis tractable. Such cooperative analyses may be unwitting. What might be viewed as "denial of service" here becomes "provision of attack capability"! [1] provides an account of a crypto attack in which execution is monitored one step at a time (with resets in between), to enable electron microscopes of limited sophistication to be used.

Perturbed environment: In the early 1980s it was found that the contents of static RAM could persist for up to minutes after power is removed if the temperature were reduced to below $-20\,^\circ$C [1]. **Perturbed system**: There are accounts of RAM contents maintained at a specific value being burned in: when the RAM is powered up about 90% of relevant bits assume their previous values. A variation on this remanence theme should be familiar to many whose experience includes VT100 terminals: the login prompt burnt onto the screen.

Perturbed environment: the power consumption profile of a cryptosystem may leak more information if the system is cooled, since thermal noise is reduced. **Perturbed system**: temperature variation may be used to cause the basic circuitry to malfunction (for example, by overheating), or to alter its timing properties (for example, temperature induces time-dependencies in FPGA circuitry [26]). Interference with the power supply, or with a supplied clock frequency, may cause system malfunction (for example, the so-called *glitch attacks*).

Retrenchment: Hardware implementations typically use analogue approximations to logical concepts. Consider the operation of hard disk storage. We view data logically as binary 0/1 values; values are physically stored by affecting the magnetic properties of locations on a disc. Under detailed scrutiny minor variations in disc head positioning may leave visible traces of previous recorded data despite that data being "overwritten". Binary values are implemented by analogue *ranges*, for example, binary 1 may be represented by a voltage range of 4.5–5.5V. In communications, a message may logically comprise a stream of 0s and 1s, where the logical abstraction is a step-function, but the analogue implementation is not. In principle, the analogue waveform may encode infinite information ignored by the system's internal interpretational mechanisms. Variation in the rise and holding times can be analysed.

Invasive observations: the ability to drop a single microprobe often suffices to break a cryptosystem (for example, knowledge of a single bitplane suffices to break algorithms such as RSA [10]). Electron microscopes have been used to read voltages on smart card chip surfaces. Thermal imaging is used to evaluate the reliability of ASICs, as hot spots are likely to fail first; can such differences be correlated with logical dynamics (for example, frequency of flipping)? What are the possibilities for detecting minor magnetic field differences?

5.4 Unintended finalisation, multiple systems

No two processors are identical in all their performance characteristics (even if they are of the same type). Obvious sources of variation are: clock speed; on-chip cache capacity; timing; pipelining; power consumption. Measurements of an algorithm running on different architectures, or on different physical instantiations of the same architecture, could be correlated to provide extra information (a "differential processor attack").

For messages over a network the intended finalisation is the content, but messages also have length, and appear on the LAN at a particular times or intervals. Variation in these can leak information too.

6 Preventing Unwanted Finalisations

6.1 Make it impossible, or infeasible, or meaningless, to observe

The most obvious and direct approach is to enforce the use of the intended finalisation glasses. These may, for example, form a layer between electrical signals and what the analyst can observe. Invasive attacks may be prevented, or made more difficult, by a variety of means. For example, smart cards may have resin coatings applied, and be encircled with tamper-detection coils.

Overwhelming the resources of an analyst provides another means of effecting security. For example, user A may communicate with B by very rapidly sending trillions of bits of potential key material having secretly agreed previously which time slot contains the actual key to be used. An attacker without this knowledge could observe all the data but would not be able to store it all for future reference.

The system may ensure that any finalisation an analyst can do does not leak any unwanted finalisation. So any (practical) alternative finalisations are not "unwanted". In the sequence/set example, the system might deliberately randomise the order of the sequence, or impose a particular order, so that an analyst cannot "see" any other underlying order. (One would then have to consider the effect this might have on timing finalisations.)

6.2 Intractable finalisations

In some cases, the information needed to do that intended finalisation is secret. The output is encrypted, and the "finalisation glasses" require the use of a secret key to work. The identity finalisation appears to be "noise", and only some privileged people have the ability to do the intended finalisation.

The prevention is trying to stop the analyst "putting on" the glasses. The analyses try to get enough information to be able to put on the glasses.

6.3 Managing attacks

If an attack is detected, evasive action might be needed. A soft way to do this is to "slug" the system: simply slow down the covert channel (for example, by reducing processor time allocated to the suspicious process). This may allow you to "play for time" whilst managerial action is taken. Another example is deliberate slugging of network requests to the system to prevent resource exhaustion.

For systems where management is not available (for example, smart cards held by the population at large), destructive action may be needed. For example, on detection of tampering, shared key material is destroyed, say by overwriting memory. In some military systems explosives may be used to render useful finalisations impossible. Explosive *partial* destruction may on the other hand form an attack technique, for example, by rapid destruction of the supporting mechanisms used to overwrite memory. (When will we see the first high velocity 'micro-bullet' attack on a smart card?)

7 Unusual finalisations

We have, perhaps, given the impression that covert channels are "a bad thing". This really depends on who you are. In some cases more powerful finalisation can be a *source of security* as well as a source of insecurity.

7.1 Different finalisations, different refinements

A specification may include several different intended finalisations, to define what is to be observed by different classes of user, or in different circumstances. Multiple refinements, capturing the multiple finalisations, are then possible. For example, a system might be provided with two intended finalisations: an ordinary user finalisation, and a special administrator finalisation that can observe more of the system, such as other users' data, and audit trails. In general, there may be privilege-dependent finalisations: for example, different security clearance providing a filter (the more clearance you have, the less dark are your finalisation glasses).

An atomic abstract operation might be implemented in a multiple step concrete operation. Breaking atomicity by finalising part way through the sequence of concrete operations could yield a useful partial result (accurate, but not yet as precise as that specified) that could be valuably observed (for example, the partial loading of certain image files by Web browsers).

7.2 Finalising the user

Consider hand signatures. It is relatively easy to forge a facsimile (the intended visual finalisation) but very difficult to forge the signature dynamics (speed, acceleration, pressure). So observing the dynamics can give a more secure signature authentication system.

Similarly, keyboard dynamics can be used as source of continuous authentication (a modern day equivalent of recognising Morse code operators from their signalling styles). A user's typing patterns could be observed and used as the basis of *continuous authentication.* The user does not consciously "supply" this information when at the terminal, and should not be able to successfully spoof another user. Or alternatively, this observation might be used as a kind of "audio Tempest" attack. It can be an interesting exercise to (occasionally) ask people who enter one's office to close their eyes and try to guess what keys are being pressed on one's computer keyboard. Everyone is able to recognise a spacebar: how would more sophisticated acoustic analyses perform?

Written text has been analysed to verify claims of authorship. The usual finalisation is semantic content, but more detailed linguistic processing (an enhanced finalisation) can be applied.

In a sense, we may view the above as unintended finalisations of *the user*. We must consider what the security policy of the system is to decide whether this is a good thing or not. If anonymity is a security requirement, then such authentication channels would be viewed in a poor light. There is also an issue as to differences between *stated* policy and *actual* policy. The general user may simply be unaware that they are being finalised in an unintended (by them) way.

7.3 Destructive quantum finalisation

Quantum computing achieves great potential power by simultaneously acting on superpositions of states. Here finalisation is an act of measurement, causing projection onto one of a number of subspaces (the state space is often said to "collapse"). The particular projection witnessed is probabilistic. Quantum computer scientists do not normally talk in terms of finalisation; [21]'s provision of finalisation as part of their quantum guarded command language forms a welcome bridge to classical formal methods.

The *destructiveness of quantum finalisation* is at the heart of security of quantum-related systems. It forms, for example, the basis for detection of eavesdropping in quantum key distribution protocols [18].

8 Conclusions

We have shown how finalisation can be viewed as a crucial formal framework for explaining many security-related aspects of systems. We have examined the power of various finalisations, enabling factors and countermeasures. Above all we have shown that finalisation is a *practical* as well as a *formal* issue. When we do formal specification and refinement we are working on formal models of an envisaged system. Each model comes complete with a set of assumptions. These may be particular to the application concerned or else derive from the semantics of the representations used. If the assumptions do not hold (or can be made not to hold) in an implementation then we have the basis of an attack. Our taxonomy and many of the attacks outlined (for example, multiple systems) indicate that a useful criterion for formal analysis may be to model the system not as the user is expected to access it, but as an attacker may view it.

We have summarised a variety of ways in which abstract assumptions can be broken, and provided a taxonomy that may be used to categorise attacks. It is well known that vulnerabilities come from "incomplete" modelling. Our taxonomy should provide a mechanism for reasoning about which aspects of a model may be incomplete.

As Jackson [13] points out, the world is unbounded. There is a richer set of experiences to be had in the implementation (physical) world. The quest for attackers is to sample that richness, in order to avail themselves of correlations and relationships with data of interest.

As technology gets ever more sophisticated, opportunities for analysis increase. Other computing paradigms affect finalisations. DNA computing algorithms to break DES have been suggested [5]. Quantum computing increases what is tractable, for example, Shor's polynomial time Quantum Discrete Fourier Transform [23] renders factorisation tractable, on a quantum computer. We cannot know where future technology-dependent finalisations will take us.

References

[1] Anderson, R., "Security Engineering: A Guide to Building Dependable Distributed Systems," Wiley, 2001.

[2] Banach, R. and M. Poppleton, *Retrenchment: An engineering variation on refinement*, in: *B'98: 2nd International B Conference, Montpellier, France*, LNCS **1393** (1998).

[3] Barden, R., S. Stepney and D. Cooper, "Z in Practice," BCS Practitioners Series, Prentice Hall, 1994.

[4] Baum, L. F., "The Wonderful Wizard of Oz," George M. Hill, 1900.

[5] Boneh, D., R. DeMillo and R. J. Lipton, *Breaking DES using a molecular computer*, in: *Proceedings of DIMACS workshop on DNA computing* (1995).

[6] Boneh, D., R. Lipton and C. Dunworth, *On the importance of checking computations* (1996). URL http://www.demillo.com/PDF/smart.pdf

[7] Clark, J. A. and J. L. Jacob, *Fault injection and a timing channel on an analysis technique*, in: *Eurocrypt 2002*, 2002.

[8] Derrick, J. and E. Boiten, *Unifying concurrent and relational refinement*, in: *REFINE'2002*, ENTCS **70(3)** (2002).

[9] Gollman, D., "Computer Security," Wiley, 1998.

[10] Handschuh, H., P. Pailer and J. Stren, *Probing attacks on tamper resistant devices*, in: *Cryptographic Hardware and Embedded Systems, CHES 99*, LNCS **1717** (1999), pp. 303–315.

[11] He Jifeng, C. Hoare and J. W. Sanders, *Data refinement refined (resumé)*, in: *ESOP'86*, LNCS **213** (1986), pp. 187–196.

[12] Hoare, C. A. R., "Communicating Sequential Processes," Prentice Hall, 1985.

[13] Jackson, M., "Software Requirements and Specifications," Addison-Wesley, 1995.

[14] Jacob, J. L., *Basic theorems about security*, Journal of Computer Security **1** (1992), pp. 385–411.

[15] Kocher, P., *Timing attacks on implementations of Diffie Hellman, RSA, DSS and other systems*, in: *Advances in Cryptology – Crypto 96 Proceedings*, LNCS **1109** (1996).

[16] Kocher, P., J. Jaffe and B. Jun, *Introduction to differential power analysis and related attacks* (1998).
URL www.cryptography.com/resources/whitepapers/DPATechInfo.pdf

[17] Morgan, C., A. McIver, K. Seidel and J. W. Sanders, *Refinement-oriented probability for CSP*, Formal Aspects of Computing **8** (1996), pp. 617–647.

[18] Nielsen, M. A. and I. L. Chuang, "Quantum Computation and Quantum Information," Cambridge University Press, 2000.

[19] Reed, G. M. and A. W. Roscoe, *A timed model for Communicating Sequential Processes*, TCS **58** (1988), pp. 249–261.

[20] Roscoe, A. W., "The Theory and Practice of Concurrency," Prentice Hall, 1998.

[21] Sanders, J. W. and P. Zuliani, *Quantum programming*, Technical Report TR-5-99, Programming Research Group, Oxford University Computing Laboratory (1999).

[22] Schellhorn, G., *Verification of ASM refinements using generalized forward simulation*, Journal of Universal Computer Science **7** (2001), pp. 952–979.

[23] Shor, P. W., *Polynomial time algorithms for prime-factorisation and discrete logarithms on a quantum computer*, SIAM Journal of Computing **26** (1997), p. 1484.

[24] Spivey, J. M., "The Z Notation: a Reference Manual," Prentice Hall, 1992, 2nd edition.

[25] Stepney, S., D. Cooper and J. Woodcock, *More powerful Z data refinement: pushing the state of the art in industrial refinement*, in: *ZUM'98, Berlin*, LNCS **1493** (1998), pp. 284–307.

[26] Thompson, A., "Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution," Distinguished dissertation series, Springer, 1998.

[27] Woodcock, J. C. P. and J. Davies, "Using Z. Specification, Refinement, and Proof," Prentice–Hall, 1996.