Security-By-Contract for the Future Internet⋆

Fabio Massacci[1], Frank Piessens[2], and Ida Siahaan[1]

[1] Universita' di Trento, Italy `name.surname@disi.unitn.it`
[2] Katholieke Universiteit Leuven, Belgium `name.surname@cs.kuleuven.be`

Abstract. With the advent of the next generation java servlet on the smartcard, the Future Internet will be composed by web servers and clients silently yet busily running on high end smart cards in our phones and our wallets. In this brave new world we can no longer accept the current security model where programs can be downloaded on our machines just because they are vaguely "trusted". We want to know what they do in more precise details.

We claim that the Future Internet needs the notion of *security-by-contract*: In a nutshell, a contract describes the security relevant interactions that the smart internet application could have with the smart devices hosting them. Compliance with contracts should verified at development time, checked at depolyment time and contracts should be accepted by the platform before deployment and possibly their enforcement guaranteed, for instance by in-line monitoring.

In this paper we describe the challenges that must be met in order to develop a security-by-contract framework for the Future Internet and how security research can be changed by it.

1 The End of Trust in the Web

The World Wide Web evolved rapidly in 90's with a highlight in 1995 when the Java Applet enabled secure mobile code for the Web. In this millennium the notion of the Web has changed: rather than a network, the Web has become a platform where people migrate desktop applications. We have richer applications such as WebMail, Social Web sites, Mashups, Web 2.0 applications, etc. this is further supported by technologies such as Asynchronous JavaScript and XML (AJAX), .NET, XML, SOAP (Web Services).

Fact of Life 1 *The security model of the current version of the web is based on a simple assumption: the good guys develop their .NET or Java application, expose it on the web, and then spend the rest of their life letting other good guys using it while stopping bad guys from misusing it.*

The business trend of outsourcing processes [16] or the construction of virtual organizations [18] have slightly complicated this initially simple picture. Now running a "service" means that different service (sub)components can be dynamically chosen and different partners are chosen to offer those (sub)services.

Hence we need different trust establishment mechanisms (see e.g. [23, 22]). A large part of the WS security standards are geared to solve some of these problems: WS-Federation defines the mechanisms for federating trust; WS-Trust enables security token interoperability; WS-Security [3] covers the low level details such as message content integrity and confidentiality; WS-Security Policy [9] details lower level security policies .

Still, the assumption is the same: *the application developer and the platform owner are on the same side*. Traditional books on secure coding [20] or the .NET security handbook [24] are pervaded by this assumption.

Unfortunately, this assumption is no longer true for the brave new world of Web 2.0 and the Future Internet. Already now a user downloads a multitude of communicating applications ranging from P2P clients to desktop search engines, each of them ploughing through the user's platform, and springing back with services from and to the rest of the world. Most of these applications will be developed by people and companies that a lay user had never known they existed (at least before downloading the application).

It looks like we are simply back to the good old security model of Java applets [15] and good confinement would do the job. Nothing could be more wrong: applets are light pieces of code that would not need access to our platform. Indeed, to deal with the untrusted code either .NET [24] or Java [15] can exploit the mechanism of permissions. Permissions are assigned to enable execution of potentially dangerous or costly functionality, such as starting various types of connections. The drawback of permissions is that after assigning a permission the user has very limited control over how the permission is used. Conditional permissions that allow and forbid use of the functionality depending on such factors as bandwidth or the previous actions of the application itself (e.g. access to sensitive files) are also out of reach. Once again the consequence is that either applications are sandboxed (and thus can do almost nothing), or the user decided that they are trusted and then they can do almost everything.

The mechanism of signed assemblies from trusted third parties does not solve the problem either.

Fact of Life 2 *Currently a signature on a piece of code only means that the application comes from the software factory of the signatory, but there is no clear definition of what guarantees it offers. It essentially binds the software with nothing.*

Loosely speaking, the mobile software deployment process is identical to the hiring process of the aristocratic armies. In order to hire an officer you don't ask for his CV, you don't stipulate a contract with him and set targets. You just ask for his father's name and depending on that name you make him lieutenant, major or general. You grant him the privileges of the rank and trust that he'll not betray the name of the family.

The (once) enthusiast installers of UK Channel 4 on demand services 4oD [1] might tell a different story [29]. What is best than download a client that allows you to see almost free movies from your favorite TV channel? After all you are downloading from a reputable and trusted broadcaster. It is not shady software from a hacker web site. Only in the fine print of the legal terms of use (nowhere in the FAQs and only visible after a long scrolling down of legalese) you find something you most likely would like to know beforehand (extracted from the web site on 31st of July 2008):

As one of the many unfortunate users of the system noticed [29], there is no need of contacting your ISP. They will contact you pretty soon and will not be pleasant...

Fact of Life 3 *We end up in a stale-mate. We built our security models on the assumption that we could trust the vendors (or at least some of them). The examples from reputable companies such as Channel 4 (or BBC, Sky TV etc.) show that this is no longer possible. Still we really really want to download a lot of software.*

2   The Smart(Card) Future of the Web

The model that we have described above is essentially the web of the personal computers. We, as world-wide consumer[3], accept the idea that PC applications fails, that PC are ridden with viruses, spyware and so on. So we do not consider this a major threat

Fact of Life 4 *None of the users complaining about 4oD [29] have considered their PC or their Web platform "broken" because it allowed other people to make use of it. They did not consider returning their PC for repair. They considered themselves being gullible users ripped off by an untrusted vendor.*

There is another domain at the opposite side of the psychological spectrum: smart-card technology. This technology enjoyed worldwide deployment in 90's with Java Card Applets and their strict security confinement. At the beginning of the millennium, many applications such as large SIM cards, emerging security and identity management businesses are implemented on smart-cards to address mobile devices security challenges [19]. Still, smart-cards have essentially led a sheltered life from the Web problems we have described. When used in mobile phones they just acted as authenticator and withdrawn from the picture immediately.

(Un)fortunately, the smartcard technology evolved with larger memories, USB and TCP/IP support and the development of the Next-Generation Java Card platform with Servlet engine. This latter technology is a full fledged Java platform for embedded Web applications and opens new Web 2.0 opportunities such as NG Java Card Web 2.0 Applications. It can also serve as alternative to personalized applications on remote servers so that personal data no longer needs to transmitted to remote third-parties.

Prediction 1 *The Future Internet will be composed by those embedded Java Card Platforms running on high end smart cards in our phones and our wallets, each of them connecting to the internet and performing secure transactions with distributed servers and desktop browsers without complicated middleware or special purpose readers.*

We still want to download a huge amount of software on our phones but there is a huge psychological difference from a consumer perspective.

---

[3] We should distinguish between the computer scientist or security expert and the computer, even if savvy, user.

Fact of Life 5 *If our PC is sluggish in responding,* we *did something wrong or downloaded the wrong software, if our phone is sluggish,* it *is broken.*

Idea 1 *In the realm of next generation Java card platforms we cannot just download a software without knowing what it does. The smart card web platform must have a way to check what is downloading.*

3 Security by Contract for the Smart Future Internet

In the past millennium Sekar et al. [32] have proposed the notion of Model Carrying Code (MCC) as the seminal work on which our research agenda for the Smart Future Internet is based. MCC requires the code producer to establish a model regarding the safety of mobile code which captures the *security-relevant behavior* of the code. The code consumers checks their policies against the model associated with untrusted code to determine if this code will violate their policy.

The major limitation was that MCC had not fully developed the whole lifecycle and had limited itself to finite state automata which are too simple to describe realistic policies. Even a simple, basic policy such as "Only access url starting with http" could not be addressed. The *Security-by-Contract (*S×C*)* framework that we have developed for mobile code [11, 10] builds upon the MCC seminal idea to address the *trust relationship* problem of the current security models in which a digital signature binds a contract with nothing.

Idea 2 *In* S×C *we augment mobile code with a claim on its security behavior (an* application's contract*) that could be matched against a mobile* platform's policy *before downloading the code. A digital signature does not just certify the origin of the code but also bind together the code with a contract with the main goal to provide a semantics for digital signatures on mobile code.*

This framework is a step in the transition from trusted code to trustworthy code.

This idea is nice but we must develop it fully in order to really make a significant advance over the initial intuition from model carrying code. So we should consider the full lifecycle. A contract should be negotiated and enforced during development, at time of delivery and loading, and during execution of the application by the mobile platform. Figure 1 summarizes the phases of the application/service life-cycle in which the contract-based security paradigm should be present.

At *development time* the mobile code developers are responsible for providing a description of the security behavior that their code finally provides. Such a code might also undergo a formal certification process by the developer's own company, the smart card provider, a mobile phone operator, or any other third party for which the application has been developed. By using suitable techniques such as static analysis, monitor in-lining, or general theorem proving, the code is certified to comply with the developer's contract. Subsequently, the code and the security claims are sealed together with the evidence for compliance (either a digital signature or a proof) and shipped for deployment.
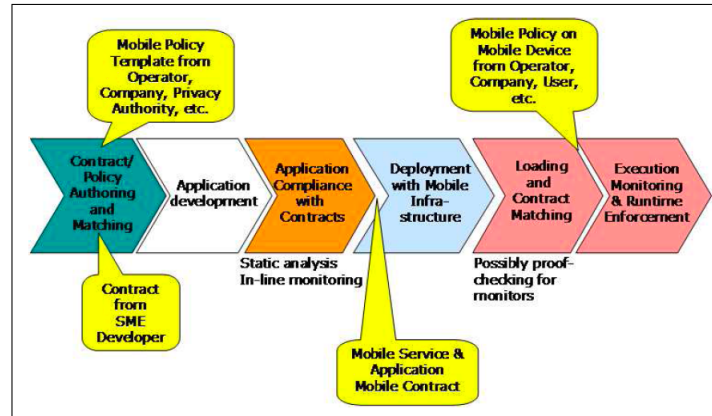
Fig. 1: Application/Service Life-Cycle

At *deployment time*, the target platform follows a workflow similar to the one depicted in Fig.2 (see also [35]). First, it checks that the evidence is correct. Such evidence can be a trusted signature as in standard mobile applications [40]. An alternative evidence can be a proof that the code satisfies the contract (and then one can use PCC techniques to check it [28]).

Once we have evidence that the contract is trustworthy, the platform checks, that the claimed policy is compliant with the policy that our platform wants to enforce. If it is, then the application can be run without further ado. This may be a significant saving from in-lining a security monitor.

At *run-time* we might want to decide to still monitor the application. Then, as with vaccination, we might decide to inline a number of checks into the application so that any undesired behavior can be immediately stopped or corrected.

4   What is a Contract for the Smart Future Internet?

The first challenge that we must address is finding an appropriate language for defining contracts and policies.

Definition 1. *A* contract *is a formal complete and correct specification of the behavior of an application for what concerns relevant security actions (Virtual Machine API Calls,Web Messages etc).*

By signing the code the developer certifies that the code complies with the stated claims on its security-relevant behavior.

On the other side we can see that users and mobile phone operators are interested that all codes that are deployed on their platform are secure. In other words they must declare their security policy:
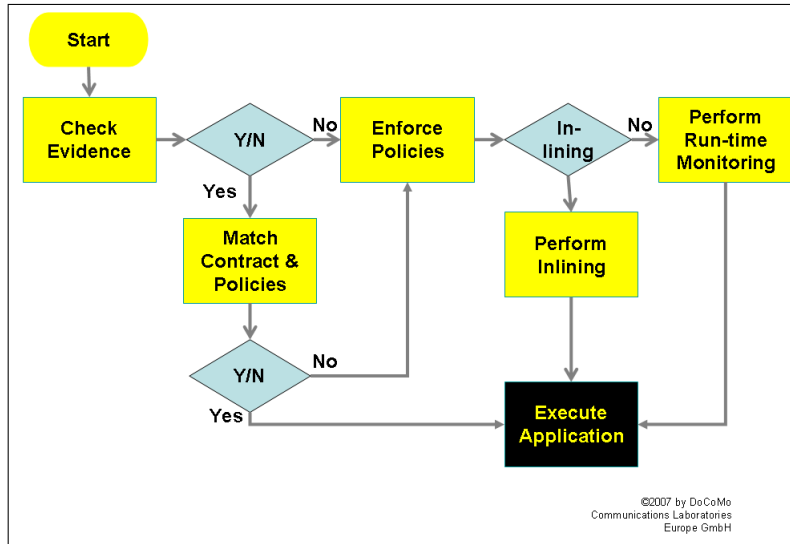
Fig. 2: SxCWorkflow

Definition 2. *A* policy *is a formal complete specification of the acceptable behavior of applications to be executed on the platform for what concerns relevant security actions* .

Technically, a contract can be a security automaton in the sense of Schneider [17], and it specifies an upper bound on the security-relevant behavior of the application: the sequences of security-relevant events that an application can generate are all in the language accepted by the security automaton. We can have a slightly more sophisticated approach using Büchi automata [34] if we also want to cover liveness properties that can be enforced by Edit automata [4]. This definition can be sufficient for theoretical purposes but it is hardly acceptable for any practical use.

State-of-the-Art 1 *All theoretical papers [17, 4, 34] define the security behavior as a set of "actions" as ground terms but real programs are not made by "actions", they have API calls, OS calls, and those calls have a number of parameters. Even the most basic security policy if we simply instantiate the API parameters into ground actions will lead to automata with infinitely many transitions that we cannot even write down.*

Example 1. The policy of smart card provider may require that "After PIM (the Personal Identification Module) APIs were accessed only secure connections can be opened". This policy permits executing the Java `Connector.open(string url)` method only if the started connection is a secure one i.e. `url` starts with "https://".

Fig.3a represents an automaton for Ex. 1. Starting from state p_0, we stay in this state while PIM is not accessed (jop). As PIM is accessed we move to state p_1 and we

(a) Infinite Transitions Security Policies

$$joc(v_{joc,1}) \doteq \texttt{io.Connector.open}(url)$$
$$jop \doteq \texttt{pim.PIM.openPIMList}(\dots)$$
$$q \doteq \texttt{io.Connector}.type$$
$$\text{is protocol type e.g. "http"}$$
$$pr(q) = type \doteq \text{permission } q \text{ is for protocol } type$$
$$p(url) = type \doteq url.\texttt{startsWith}(type)$$
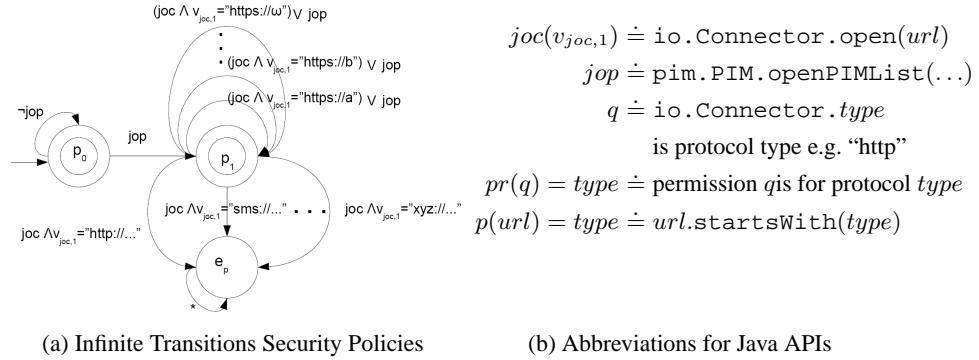
(b) Abbreviations for Java APIs

Fig. 3: Infinite Transitions Security Policies

stay in state p_1 only if the started connection Connector.open(string url) method is a secure one i.e. url starts with "https://" or we keep accessing PIM (jop). We enter state e_p if we start an unsecure connection Connector.open(string url) e.g. url starts with "http://" or "sms://" etc. These examples are from a Java VM. Since we do not consider useful to invent our own names for API calls we use the `javax.microedition` APIs (though a bit verbose) for the notation that is shown in Fig.3b.

Idea 3 *For S×C for mobile code (.NET and Java) a variant of the PSLANG language [2] has been proposed whose formal counterpart is the notion of* automata modulo theory *[25] where atomic actions are replaced by expressions that can finitely capture infinite values of API parameters.*

Challenge 1 *Identify a suitable language for the specification of contracts and policies at a level of abstraction that is suitable for the smart future internet that can be used for* all *phases of the life-cycle (Fig.1) both at development and deployment time (Fig.2)*

It is indeed important that the language is able to be used in all steps. A language perfect for matching that cannot be enforced at run-time or that can only be verified with a costly interactive theorem prover is not going to be very effective.

5 Application-contract compliance

So far we have only defined a language for describing the behavior of smart-card web applications. There is no a-priori guarantee that this statement is correct.

Idea 4 *Static analysis can be used at development time to increase confidence in the contract. With static analysis, program analysis and verification algorithms are used in an attempt to* prove *that the application satisfies its contract.*

The major advantages of static analysis are that it does not impose any runtime overhead, and that it shows that all possible executions of a program comply with the contract. The major disadvantage is that the problem of checking application-contract compliance is in general undecidable, and so automatic static analysis tools will typically only support restricted forms of contracts, or restricted forms of applications, or the tool will be *conservative* in the sense that it will reject applications that are actually compliant, but the tool fails to find a proof for this.

The S3MS project for mobile code has shown that static analysis is feasible for limited forms of contracts (e.g. for contracts that are stateless), or in combination with runtime verification [37].

The programs and services running on the embedded servlet will be significantly more complex and have actions at different level of abstractions whose full security implications can be understood by considering all abstraction levels at once. The challenges for static analysis are many: with expressive notions of security contracts, verifying application-contract compliance is actually as hard as verifying compliance with an arbitrary specification [31].

Prediction 2 *Contracts for applications in the Smart Future Internet will have a complexity that that is comparable to the level of abstractions of current concurrent models that are used for model checking hardware and software systems (in 10^{10} states or transitions and beyond).*

A standard approach to make program verification and analysis algorithms scale to large programs is to make them *modular*: make sure that the algorithm can check parts (classes / methods / ...) of the program independently. This is particularly hard for application-contract compliance checking, because the security state of the contract is typically a global state, and the structure of the contract and its security state might not align with the structure of the application.

State-of-the-Art 2 *For modular verification algorithms, annotations are required on* all *methods to specify how they interact with the security state, and not only on methods that are relevant for the contract at hand.*

Clearly, this annotation overhead is prohibitive, so a key challenge is to look for ways to reduce the annotation burden.

Idea 5 *An interesting research question is whether a program transformation (similar to the security-passing style transformation used for reasoning about programs sandboxed by stack inspection [39, 33]) can improve this situation.*

Idea 6 *A second approach to address scalability is to give up soundness of the analysis, and to use the contract as a model of the application in order to generate security tests by applying techniques from Model Based Testing [38].*

Losing soundness is a major disadvantage: an application may pass all the generated tests and still turn out to violate the contract once fielded. However, the advantages are also important: no annotations on the application source code are needed, and the tests generated from the contract can be easily injected in the standard platform testing phase, thus making this approach very practical.
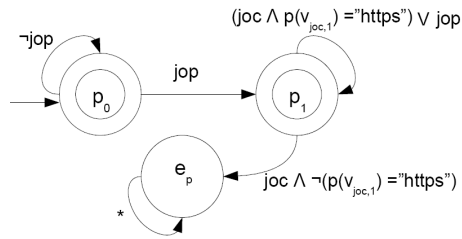
Fig. 4: Example of Automata Modulo Theory for the policy from Ex. 1

Challenge 2 *One particularly interesting research challenge to be addressed here is how to measure the coverage of such security tests. When are there enough tests to give a reasonable assurance about security?*

It is easy to automatically generate a huge amount of tests from the contract. Hence it is important to know how many tests are sufficient, and whether a newly generated test increases the coverage of the testing suite.

6 Matching Contract and Policy on the Smart Future Internet

Suppose our language constructs allowed the developer to provide a verified contract. Now we are at the time of deployment and, as users, we would like precisely to check whether our intriguing application will not use our upward bandwidth as the 4oD unfortunate users. We must therefore identify the next key component of the SxC paradigm, namely *contract-policy matching*.

The operation of matching the application's claim with the platform policy requires that the contract is trustworthy, i.e. the application and the contract are sealed together with a digital signature when shipped for deployment or by shipping a proof that can checked automatically. We will return on this issue in a later section but for the moment let's take it for granted.

Idea 7 *We must show that the behavior described by the contract is acceptable according to our platform policy.*

A simple solution is to build upon automata theory, interpret contract and policy as automata and use language inclusion [6]. Given two such automata Aut^C (representing the contract) and Aut^P (representing the policy), we have a match when the language accepted by Aut^C is a subset of the language accepted by Aut^P.

As we have already shown in Figure3a this cannot be done just be instantiating the variables of API calls in order to obtain the usual set of ground actions. A solution that we have used in the S3MS project has been to introduce the notion of *automata modulo theory* in which actions labelling the transactions of the finite state automata are instead expressions constraining the value of the arguments of an API call [25]. In Figure4 we show the automaton modulo theory corresponding to the infinite automaton in Figure3a.

Once the policy and the contract are represented as automata then one can either use language inclusions [25] or simulation [26] to check whether the contract is acceptable according our platform policy.

Such solution is only partial because the automata that we have envisaged do not store the values of the arguments of allowed/disallowed APIs. As a result the policy below cannot be yet matched.

Example 2. After connecting to a URL X then the application is only allowed to connect to the same URL X".

Such policy would allow us to solve the 4oD problem: once the site connect to Channel4 web site it can only connect to this site again. So no bandwidth can be used while acting as a P2P servent.

In order to do this a potentially promising research could be to build a version of automata modulo theory that is able to exploit the usage of action arguments that is typical of process algebra approaches. For instance, one could exploit the works on history-dependent automata (HD-automata) [27, 14] which extended automata by local names on states and labels.

Challenge 3 *Contracts and policies for the future internet must be history-dependent: the arguments of past allowed actions (API calls, WS invocations, SOAP messages) may influence the evolution of future access control decision in a policy.*

Further, in our current implementation of the matcher that runs on a mobile phone, security states of the automata are represented by variables over finite domains e.g. smsMessagesSent ranges between 0 to 5. [2, 5]. A possible solution could be to extend the work on finite-memory automata [21] by Kaminski and Francez or other works [30] that studied automata and logics on strings over infinite alphabets through register and pebble automata.

Challenge 4 *Define matching for contracts and policies that allows to compactly represent states with potentially infinite state spaces without giving up effective matching.*

The last clause of the challenge ("without giving up effective matching") is essential. Remember that our model of the Future Internet is built up by powerful smartcards running their own web servers and clients and sitting on our mobile phones, devices and cars. We cannot wait for two hours of BBD construction using current model checking technologies before deciding that an interesting travel program discovered at our arriving airport is not good for our phone.

Hence we arrive here in the same corner that we ended up during the discussion on static analysis.

Idea 8 *Another approach to address scalability is to give up soundness of the matching and use algorithms for simulation and testing.*

Also in this case losing soundness is a major disadvantage: a contract may pass the matching and still turn out to violate the policy once fielded. However, the advantages are also important. A quick decision with high probability of correctness is significantly better than no decision due to memory consumption: if users get tired of waiting or the

device has not enough battery to run a full test, users might decide to run the program anyhow and we would end up in a bad situation.

Challenge 5 *One particularly interesting research challenge to be addressed here is how to measure the coverage of approximate matching. Which value should give a reasonable assurance about security? Should it be an absolute value? Should it be in proportion of the number of possible executions? In proportion to the likely executions?*

An interesting approach could be to recall to life a neglected section of the classical paper on model checking by Courcoubetis et al [7] in which they traded off a better performance of the algorithm in change for the possibility of erring with a small probability.

7 Inlining a monitor on Future Internet Applications

What happens if matching fails? or what happens if we do not trust the evidence that the code satisfies the contract? If we look back at Fig.2 monitor inlining of the *contract* can provide strong assurance of compliance. Here, we highlighting the research challenges that still remain.

With *monitor inlining* [13], code rewriting is used to push contract checking functionality into the program itself. The intention is that the inserted code enforces compliance with the contract, and otherwise interferes with the execution of the target program as little as possible. Monitor inlining is a well-established and efficient approach [12], and the S3MS project has shown that inlining can be used today as a contract compliance technique [36].

However a major open question is how to deal with concurrency efficiently.

Prediction 3 *Servents in the Smart Future Internet will need to monitor the concurrent interactions of tens of untrusted multithreaded programs.*

An inliner needs to protect the inlined security state against race conditions. So all accesses to the security state will happen under a lock. A key design choice for an inlining algorithm is whether to lock across security relevant API calls, or to release the lock before doing the API call, and reacquiring it when the API call returns.

The first choice (locking across calls) is easier to get secure, as there is a strong guarantee that the updates to the security state happen in the correct order. This is much trickier for an inliner that releases the lock during API calls. However, an inliner that locks across calls can introduce deadlocks in the inlined program, because some of the security relevant API calls will themselves block. And even if it does not lead to deadlock, acquiring a lock across a potentially blocking method call can cause serious performance penalties.

The S3MS project has provided a partial solution by partitioning the security state into disjoint parts, and replacing the global lock, by per-part locks. This improves efficiency, but depending on application and policy, it can still introduce deadlocks.

Challenge 6 *How to inline a monitor into a concurrent program so that it cannot create a deadlock in future interactions with other unknown programs yet to be downloaded.*

The ability to resist to changes in context (i.e. new concurrent programs downloaded after the inlined program) is essential for usability. The inlined version of 4oD should not get in the way if later on I want to download a (inlined) role-playing game. Of course it is possible that two malicious software downloaded at different instants might try to cooperate in order to steal some data. The security monitor should be able to spot them but not be deadlocked by them.

If inlining is performed by the code producer, or by a third party, the code consumer (the client that actually runs the application) needs to be convinced that inlining has been performed correctly. Without a secure transfer of the guarantees of application-contract compliance to the client, it would be easy for an attacker to modify either the application or the contract, or it would be possible for an application developer to lie about the contract.

Cryptographic signatures by a trusted (third) party is a first solution even if it transfer the risk from the technical to the legal domain. The trusted party vouches for application-contract compliance. Note the difference with the use of signatures in the traditional mobile device security model. In the security-by-contract approach, a signature has a clear semantics [11]: the third party claims that the application respects the supplied contract. Moreover, what is important is the fact that the decision whether the contract is acceptable or not remains with the end user. If an application claims that it will not connect to the internet and instead it does, at least you can bring the signatory to the court for fraudulent commercial claims.

An alternative solution is whether we can use the techniques of Proof-Carrying-Code (PCC) [28] for this. In PCC, the code producer produces a proof that the code has certain properties, and ships this proof together with the code to the client. By verifying the proof, the client can be sure that the code indeed has the properties that it claims to have.

State-of-the-Art 3 *Proof verification is a relatively simple process, so the key issue in PCC systems is how to generate the proofs (and how to keep them compact). Currently proof generation requires essentially PhD students working on an interactive theorem prover for hours or months using complicated logics and type systems. In other words, it is unfeasible.*

The difficulty of the endeavour is that the code has not been produced to be verified compliant against a security property but usually to actually do some business. In other words, the code producer is not aware of the property and the property producer is not aware of the code. In this scenario verification is clearly an uphill path.

Idea 9 *When we inline a contract we know precisely what code we are inlining and also what property the inlined code should satisfy. So, instead of asking a PhD student to annotate the code, we can ask the inliner to do this automatically for us. Indeed we could ask the inliner to generate the proof directly.*

This should make it relatively easy to check that code complies with the contract: the generation of a proof should be easier, and the size of the proof would also be acceptable for inlined programs. Preliminary results from the S3MS project for PCC for inlined sequential Java [8] show that this is indeed the case.

Challenge 7 *Identify automatic inlining mechanisms that inline a monitor for a security contract and generate an easily checkable proof for industrial applications in the Smart Future Internet.*

8 Beyond Micro-Security for the Future Internet

In the discipline of economics there is a traditional distinction between micro-economics and macro-economics. According the Wordnet dictionary at Princeton University, the former is "the branch of economics that studies the economy of consumers or households or individual firms" while the latter is "the branch of economics that studies the overall working of a national economy".

Idea 10 *We can now draw a parallel of the notion of micro- and macro- research into the realm of security research.*

microsecurity *is the branch of IT security research that studies the security of individual digital services, components, or organizations*
macrosecurity *is the branch of IT security research that studies the overall security behavior of a large population of digital entities.*

State-of-the-Art 4 *All our ideas and challenges and the 99.9% of all security research in the world has been in the field of microsecurity. We have been fixing, breaking and proving correct an individual service or protocol or the interaction between N entities discussing the individual interactions between them.*

The picture is slowly changing as epidemiological studies on viruses appeared and research targetting population is starting. For example, researchers at NEC Japan are considering solutions to the problem of SPAM mails that do not focus on better filtering algorithms on the client (i.e. a micro-security solutions) and works if a "population" of servers as a whole adopts the much simpler measures of throttling email invoices to the average rate.

Challenge 8 *In the Future Internet few millions of smart servents will adopt and enforce a type of contract (e.g. by Axalto) and some other millions of servents (e.g. by G&D) might adopt different contracts. What can we say about the population as a whole? How will security incidents spread? What kind of private data will be lost?*

If we are able to meet this challenge, then macro-security will be born.

References

1. Channel 4. 4od. Available on the web http://www.channel4.com/4od/index.html, 2008.
2. I. Aktug and K. Naliuka. Conspec - a formal language for policy specification. In Proc. of the 1st Workshop on Run Time Enforcement for Mobile and Distributed Systems (REM2007), 2007.

3. Bob Atkinson, Giovanni Della-Libera, Satoshi Hada, Maryann Hondo, Phillip Hallam-Baker, Johannes Klein, Brian LaMacchia, Paul Leach, John Manferdelli, Hiroshi Maruyama, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, John Shewchuk, and Dan Simon. *Web Services Security*. Microsoft, IBM, VeriSign, 1.0 edition, April 2002. available via http://www-128.ibm.com/developerworks/webservices/library/ws-secure/ on 25/10/2005.
4. L. Bauer, J. Ligatti, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. *Int. J. of Inform. Sec.*, 4(1-2):2–16, 2005.
5. N. Bielova, M. Dalla Torre, N. Dragoni, and I. Siahaan. Matching policies with security claims of mobile applications. In *Proc. of the 3rd Int. Conf. on Availability, Reliability and Security (ARES'08)*. IEEE Press, 2008.
6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.
7. C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in Sys. Design*, 1(2-3):275–288, 1992.
8. M. Dam and A. Lundblad. A proof carrying code framework for inlined reference monitors in Java bytecode. Submitted for publication, 2008.
9. Giovanni Della-Libera, Martin Gudgin, Phillip Hallam-Bakerand Maryann Hondo, Hans Granqvist, Chris Kaler, Hiroshi Maruyama, Michael McIntosh, Anthony Nadalin, Nataraj Nagaratnam, Rob Philpott, Hemma Prafullchandra, John Shewchuk, Doug Walter, and Riaz Zolfonoon. *Web Services Security Policy Language*. IBM and Microsoft and RSA Security and VeriSign, 2005.
10. L. Desmet, W. Joosen, F. Massacci, P. Philippaerts, F. Piessens, I. Siahaan, and D. Vanoverberghe. Security-by-contract on the .net platform. *Information Security Technical Report*, 13(1):25–32, 2008.
11. N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan. Security-by-Contract: Toward a Semantics for Digital Signatures on Mobile Code. In *Proc. of EuroPKI'07*. Springer-Verlag, 2007.
12. Erlingsson and Schneider. SASI enforcement of security policies: A retrospective. In *WNSP: New Security Paradigms Workshop*. ACM Press, 2000.
13. Ulfar Erlingsson and Fred B. Schneider. IRM enforcement of Java stack inspection. In *IEEE Symposium on Security and Privacy*, pages 246–255, 2000.
14. G.L. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model-checking verification environment for mobile processes. *ACM Trans. Softw. Eng. Methodol.*, 12(4):440–473, 2003.
15. L. Gong, G. Ellison, and M. Dageforde. *Inside Java 2 Platform Security: Architecture, Api Design, and Implementation*. Addison-Wesley Professional, 2003.
16. G. Goth. The ins and outs of it outsourcing. *IT Professional*, 1:11–14, 1999.
17. K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *TOPLAS*, 28(1):175–205, 2006.
18. C. Handy. Trust and the virtual organization. *Harvard Business Review*, 73:40–50, 1995.
19. Mike Hendry. *Smart Card Security and Applications*. Artech House, 2nd edition edition, 2001.
20. Michael Howard and David LeBlanc. *Writing Secure Code*. Microsoft Press, 2nd edition edition, 2002.
21. M. Kaminski and N. Francez. Finite-memory automata. *Theor. al Comp. Sci.*, 134(2):329–363, 1994.
22. Y. Karabulut, F. Kerschbaum, F. Massacci, P. Robinson, and A. Yautsiukhin. Security and trust in it business outsourcing: a manifesto. In S. Etalle and P Samarati, editors, *Proceedings of STM'06*, ENTCS. Elsevier, 2006.
23. Günter Karjoth, Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Service-oriented Assurance - Comprehensive Security by Explicit Assurances. In *Proc. of QoP'05*, 2005.

24. B. LaMacchia and S. Lange. *.NET Framework security*. Addison Wesley, 2002.

25. F. Massacci and I. Siahaan. Matching midlet's security claims with a platform security policy using automata modulo theory. In *Proc. of The 12th Nordic Workshop on Secure IT Systems (NordSec'07)*, 2007.

26. Fabio Massacci and Ida S. R. Siahaan. Simulating midlet's security claims with automata modulo theory. In *Proc. of the 2008 workshop on Prog. Lang. and analysis for security*, pages 1–9, New York, NY, USA, 2008. ACM.

27. U. Montanari and M. Pistore. History-dependent automata. Technical Report TR-98-11, Dip. Informatica, University of Pisa, 5 1998.

28. G.C. Necula. Proof-carrying code. In *Proc. of the 24th ACM SIGPLAN-SIGACT Symp. on Princ. of Prog. Lang.*, pages 106–119. ACM Press, 1997.

29. CNET Networks. Channel 4's 4od: Tv on demand, at a price. *Crave Webzine*, January 2007.

30. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *TOCL*, 5(3):403–435, 2004.

31. F.B. Schneider. Enforceable security policies. *TISSEC*, 3(1):30–50, 2000.

32. R. Sekar, V.N. Venkatakrishnan, S. Basu, S. Bhatkar, and D.C. DuVarney. Model-carrying code: a practical approach for safe execution of untrusted applications. In *Proc. of the 19th ACM Symp. on Operating Sys. Princ.*, pages 15–28. ACM Press, 2003.

33. Jan Smans, Bart Jacobs, and Frank Piessens. Static verification of code access security policy compliance of .net applications. *Journal of Object Technology*, 5(3):35–58, 2006.

34. C. Talhi, N. Tawbi, and M. Debbabi. Execution monitoring enforcement under memory-limitation constraints. *Inform. and Comp.*, 206(2-4):158–184, 2007.

35. D. Vanoverberghe, P. Philippaerts, L. Desmet, W. Joosen, F. Piessens, K. Naliuka, and F. Massacci. A flexible security architecture to support third-party applications on mobile devices. In *Proc. of the 1st ACM Comp. Sec. Arch. Workshop*, 2007.

36. D. Vanoverberghe and F. Piessens. A caller-side inline reference monitor for an object-oriented intermediate language. In *Proceedings of the 10th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'08)*, volume 5051/2008 of *Lecture Notes in Computer Science*, pages 240–258. Springer, 2008.

37. D. Vanoverberghe and F. Piessens. Security enforcement aware software development. *Information and Software Technology*, 2008. doi:10.1016/j.infsof.2008.01.009.

38. Margus Veanes, Colin Campbell, Wolfram Schulte, and Nikolai Tillmann. Online testing with model programs. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 273–282, New York, NY, USA, 2005. ACM.

39. Dan S. Wallach, Andrew W. Appel, and Edward W. Felten. Safkasi: a security mechanism for language-based systems. *ACM Trans. Softw. Eng. Methodol.*, 9(4):341–378, 2000.

40. B.S. Yee. A sanctuary for mobile agents. In J. Vitek and C.D. Jensen, editors, *Secure Internet Programming*, pages 261–273. Springer-Verlag, 1999.