A POLICY-BASED FRAMEWORK FOR THE AUTHORISATION OF SOFTWARE DOWNLOADS IN A MOBILE ENVIRONMENT

Eimear Gallery (Mobile VCE Research Group, Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX; E.M.Gallery@rhul.ac.uk)

ABSTRACT

Software defined radio is a potentially important paradigm for the future communications industry, allowing the mitigation of many problems encountered when wireless networking infrastructure and terminals are implemented completely in hardware. SDR technology allows radio functions to be implemented as software modules running on generic hardware platforms, and thus the expense and frustration involved in user migration to alternate handsets after the deployment of a new network standard can be alleviated. Problems arising when travelling between countries which have dissimilar air interfaces and link layer protocols deployed, potentially inhibiting roaming between wireless networks, can also be avoided. With the deployment of SDR these issues can be tackled via the upload of software modules onto hardware platforms, implementing different or upgraded standards, allowing many different network technologies to happily co-exist.

This exciting technology, however, brings with it an assortment of security problems. The risk of software with malicious intent damaging any system/device on which it is executed, becomes a very real danger and in this respect, determining whether or not received code should be executed on a particular hardware platform is a very serious issue. This problem becomes especially critical in a mobile environment, where bandwidth and processing power may often be limited; restricting the capabilities of a mobile node either to contact the originator of the software or to perform detailed checking of the code.

1. INTRODUCTION

This paper focuses on the construction of a policy-based authorisation framework for implementation within the mobile environment, with the objective of providing both mobile devices with the ability to assign appropriate privileges to software, according to both where it originates from and the attributes it possesses. This policy-based architecture consists of two fundamental policy models. The first exists within the trusted domain server responsible for a set of mobile devices lying within its protective boundaries, and which is accountable for the production of code attribute credentials for device use based on a series of security checks. The second exists within the mobile device and results in the output of an authorisation decision regarding running an executable based on code attribute assertions output by the domain server and defined policy statements held within the device.

We begin with a brief overview of critical analysis previously completed on a number of architectural authorisation models, to determine the fundamental requirements for an architecture for mobile code authorisation in a mobile environment as regards the framework, and policy expression of that framework. Following this, a brief overview of various policy specification language assessments is given to justify our choice of language for the definitive model. The definitive framework is then given, followed closely by policy expression and policy engine description for the domain server and the end device.

2. ARCHITECTURAL ANALYSIS

We will begin, as stated, with a brief overview of critical analysis previously completed on five possible architectural authorisation models [1]. In the first scenario code authors ask device manufacturers for consent to produce code, and in return signed attribute statements consisting of an identity element and an authorisation status element are sent to the author. Code authors can then sign and distribute code with the attribute statement. This system is basic, and the authorisation technique is very generic. An author is labelled either safe or unsafe, with no allowance for the possibility that not all code coming from a particular source is of a similar standard. Much responsibility is placed on the device manufacturer – it is not clear why the device manufacturer should be made the sole trusted point.

The fundamental concepts in the next scenario closely mirror those described in MExE, where code is allocated to an execution domain depending on the code author identity. This scenario is also rather restrictive because authorisation is based solely on the identity of the code author. It also leads to questions as to why the device manufacturer is trustworthier than a network operator.

In an alternate scenario the code author submits code segments to a chosen TTP for testing, and the type and result of successful tests are subsequently recorded in a signed credential. Here, should a problem arise with malicious code, responsibility can be assigned to the TTP in question if testing was not carried out accurately or correctly and this may deter fraudulent TTPs. This model does, however, require much more processing in the device. In addition to signature verification, proofs of code may have to be verified. It may also be difficult to assign the code to a particular domain based on test results, as no standard test methods currently exist.

By putting ACLs, containing the identities of trusted code producers, in devices, certain executables may be authorized to bypass security checks. This modification improves efficiency but may jeopardise the security of the network should a user be permitted to add identities to the 'trusted author' ACL. Alternatively ACLs can be used to increase the security of the system where code must be sent with an attribute statement signed by a TTP, but must also be signed by a trusted code producer whose identity is stored in the ACL. This set-up may however necessitate the storage of large ACLs in resource-restricted devices.

In the next scenario, code, on its receipt by a proxy server and after the signature of the code author is verified, is executed in a simulator mirroring the destination mobile device. If malicious behaviour is attempted by the code in the simulator, the code is discarded and a record of the code failure is made in the profile of the code author. Conversely, if the code behaves as expected, a note is made of this in the profile of the code author. This method, rather then confining the examination of code to specific tests, allows for the discovery of any violations that may be attempted. It does, however, mean that all code must be executed twice, which may lead to efficiency problems. However the profile database may allow domain severs to authorise code without simulated execution, after a certain positive profile level has been achieved.

3. POLICY EXPRESSION TECHNIQUE ANALYSIS

Selecting a security policy specification technique involved the examination of a large group of policy specification languages, including PLAS, ASL, Keynote, Nereus, SPL, Ponder, XML, SAML, XACML, and TPL, so that languages meriting detailed consideration could be identified. Keynote, Ponder, SAML, TPL and generic XML were then chosen for detailed consideration.

Keynote [2] was found to be a simple and flexible language, easy to read and write. One unified language is useable for both credential and policy expression which is practical and convenient. The documentation, outlining the features and attributes of the language, is also clear and precise. Signed credentials are a major attraction of this technique, in conjunction with the fact that credential chains can be used. It must also be noted that, with Keynote, the necessity for the definition of external semantics is alleviated but in order to ensure interoperability the name of the application domain, over which action attributes should be interpreted, may be cited in the attribute named "app_domain" and responsibility assigned to a suitable authority to provide a registry of reserved app_domain names, which lists the names and meanings of each application's attributes [2].

The second policy specification language investigated was Ponder [3]. The language allows for the expression of many different policy types from authorisation policies to refrain policies. Rules for composite policy construction are also explicitly defined, making simple the management of large systems. The probable occurrence of conflict among policy statements is also considered and meta-policies can be implemented to alleviate this. However, problems arose in relation to our predefined requirements, for example, this particular language assumes prior authentication, and acts merely as an access control mechanism. Ponder is also a language designed for policy statement specification but not attribute statement specification. In conjunction with this, although this language appears readable, construction of policy statements is not easy. Constraints are expressed using a subset of OMG's OCL, which is not very user-friendly to the nonprogrammer.

The third specification language investigated was SAML [4]. The language itself is no more complex then generic XML, is easy to read and write, and is both clear and unambiguous. Standard assertion and protocol schemas allow for the definition of the majority of authentication, attribute and policy decision statements, with graceful extensibility of schema and the definition of additional namespaces made possible if required.

TPL is "used to define the mapping of strangers to predefined roles, based on certificates issued by third parties" and is explored in a paper by Herzberg, Mass, Michaeli, Naor and Ravid [5]. This language poses some interesting concepts and should a role-based access control policy specification language be chosen for policy specification, TPL provides a tidy intermediary between attribute credential expression and mapping of entities, by the use of particular credentials, to roles. It is XML based, which makes it both flexible, extendible and user friendly.

Finally we considered generic XML [6], from which languages such as SAML and TPL were created. XML is a toolkit for creating and using markup languages and defines two document model types, the DTD or XML schema.

4. POLICY MODEL REQUIREMENTS

Analysis of the above scenarios enables attributes required of the final model to be extracted. As regards the mobile environment these include: minimum use of the device CPU; minimum use of device disk space for storage of security controls (e.g. ACLs); authorisation based on the code and not merely the producer id; comprehensive test sets using proved, reliable technologies such as proofs of code; and use of TTPs or preferably domain servers.

As regards system components we must consider the inclusion and adoption of: a mechanism for the protection of original code; a way in which the identities of the code author can be verified; and a means of assuring code quality. As regards policy expression, SAML was chosen for credential expression and our knowledge of TPL and XML DTD definition allowed us to define a DTD for policy statement expression.

5. THE LEGACY SYSTEM AND THE BUSINESS MODEL

As regards SDR, we focus on the protection of mobile devices within an operating network as should one device within a network be contaminated the fall of that network could result. For this particular framework the requirement is that all devices/ potential hosts within each mobile network lie within the protective boundaries of a domain server responsible for authorisation. From investigation into mobile terminated routeing protocols, I envisage the integration of the security proxy server into the GPRS servers or the border gateways that exist between PLMNs.

As regards the mobile architecture, parties include mobile devices, network operators, code consumers and software houses. Software houses produce code and sell it in order to make a profit and therefore should accept responsibility and cost for the production of code quality guarantees. The consumer, who purchases code from the software house and gains functionality from the code, will also be willing to pay code authors, to ensure that code comes with security controls, and trusted domain servers, for completion of independent checks. Finally the mobile operators have a vested interest in ensuring that their customers remain satisfied and that their network remains functional. Circulation of these executables also boosts network traffic and mobile functionality, a profitable result for the operators.

6. SECURITY FRAMEWORK

The initial step in constructing a policy-based framework involves the development of the underlying architecture; the assignment of roles and responsibilities to each of the identified participants; the selection of the state of the art security mechanisms to be deployed; and the definition of protocols associated with the architecture.

6.1 Entity Roles and Responsibilities

Entities include the software provider houses, responsible for the manufacture and production of signed code in conjunction with proofs of code; the code consumer; the mobile device, which must contain the relevant policy statements and policy engine such that incoming code can be assessed and either discarded or authorised to varying degrees; trusted security domain servers, responsible for the verification of code safety; and certification authorities, responsible for the verification of key ownership and the creation of public key certificates.

6.2 Digital Signatures

As regards security system components, the first to be deployed is the certificate-based asymmetric digital signature, which provides protection of the original code and unequivocal evidence of the author's identity. There are many security issues however that must be considered as regards the implementation of this mechanism. We will focus on RSA and DSA schemes on some occasions so as to illustrate the importance of correct parameter selection in relation to particular signature schemes.

Problems may arise in relation prime generation. In the case of both RSA and DSA, attacks have been launched against schemes that utilise weak primes. It is true to say that, "choosing a strong prime is like locking one door, but leaving others unlocked, choosing large primes is like locking all the doors" [7]. In both of the schemes, moduli of larger sizes, implied by choosing large prime numbers negates the need to use specially devised prime generation processes to avoid weak primes.

In randomised schemes, random number generation is also proved critical and whether truly random numbers, pseudorandom numbers, or cryptographically generated numbers are used, the numerical output must be comprised of truly random and unpredictable numbers which are of uniform distribution and independent.

In order that the hash function doesn't lead to any security breaches it should possess the following attributes: it can be applied to a block of data of any size; it produces a fixed length output; the hash of any message is relatively easy to compute so that both hardware and software implementations are practical, one-way property and either weak collision resistance or preferably strong collision resistance [8].

In relation to digital signatures with message recovery, a good redundancy adding function must be chosen, as the choice of function is critical to the security of the system, where a redundancy function is one in which the message to be sent is usually input into so that it will be in the correct format to be input into the signature generation process.

In both the RSA and DSA schemes, among others, the danger of using a common modulus also exists. In the RSA scheme, this is a proven threat, and there are a variety of attacks on systems of this nature, one of which for example involves a probabilistic method of factoring n. In relation to the DSA, while there has been no proven attack on such a system where moduli are the same, it is suggests that using such a system is only an invitation for cryptanalysis.

6.3 PKI

PKI is defined as the 'set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke public key certificates based on public key cryptography' [9]. Managerial issues to be considered before implementation include the following.

In March 2001, Verisign discovered that in January 2001 they had issued two class-3 certificates to an impostor who claimed to be an employee of Microsoft. This was the result of the certification authority failing to correctly authenticate the recipient of the certificate and it led to the to a situation where malicious code could have been distributed by the attacker and accepted by any user it was sent to without hesitation due to the 'legitimate' certificate that accompanied it. So, as stated by the Vice President and General Manager of Applied Services at the Mountain View, California, "Due to human error we did not detect that the individual concerned misrepresented that they worked for Microsoft when, in fact, they did not" [10]. We see a system based on complex mathematics and a highly intricate trust infrastructure fall over due to a slip in the original authentication of the requester of the certificate.

In order to implement this digital signature technology, software implementations, such as secure operating systems or alternatively hardware devices such as processor cards or crypto boxes have been proposed, all of which can be attacked by Trojan horses, programs which overtly do one thing while covertly doing another [11]. Some of the first evidence illustrating that data could be signed that was actually different from the displayed data the signatory thought he was signing was illustrated by Rossnagel, 1994 [12] in systems which used chip cards that could be manipulated by third parties at operating system level. There have been numerous opinions on this matter as to how easy it is for insiders to distribute these Trojan horses, such as those employed as software distributors or as personnel involved in installation or maintenance.

A further issue, which may lead to concerns, is that of key generation and the closely linked concept of key recovery. Although one of the attractive attributes of CA-based asymmetric signature schemes is that each participating individual can generate their own key pair, many individuals and organisations alike do not possess the expertise to do so and therefore rely on the services of a trusted key generation facility to do this for them. Although it would be good practise on the part of the third party involved to generate the key pairs, distribute them to the parties concerned and then ensure that no record of the key pair is held on any internal database of the company, one must be aware of the situation that could arise when third parties become involved and all individuals must ensure that they are using respected and trusted key generation services. It is important to distribute the key pairs securely once they have been generated so that they are free from the danger of interception and compromise.

The Revocation Problem is also one that should be considered in relation to PKI use. Whether CRLs, certificate distribution points, delta CRLs, indirect CRL, OCSP or SCVP is utilised, timely and effective revocation must be achieved.

One final issue, which may seem to be an obvious one, but one that mustn't be overlooked, is that of physical security and access control.

6.4 Virus Scanning

A selection of virus scanning techniques will also be used within the domain server on all incoming code prior to the completion of any other security checks in order to ensure a basic level of security. These mechanisms include first-generation simple scanners, which merely scan code for what are labelled virus signatures, bit patterns and structured pieces of code known to be and indication of malicious activity and second-generation heuristic scanners, which make use of heuristic rules in order to detect probable virus infection such as looking for fragments of code that are often associated with viruses or verifying checksums, appended to each program.

6.5 Trust Relationships

We will now move on to investigate the trust bonds in existence within the framework. The first of these trust bonds is that of the implicit trust between all host devices and the particular trusted domain server to which they are bound; the second represents the trust relationships that may be arranged between domain servers and software houses, and there are also the trust bonds constructed between various domain servers who may place varying degrees of trust in one another. As regards trust ratings, assignment may depend on quality of service of code generally received, audit and accountability, compliance with accepted industry standards and all relevant regulation, contract, liability, policy statement, performance and reputation or through transitive trust and a code author may be assigned a high, medium or low trust level and a domain server may be assigned a high or medium level of trust in defined policy identity lists. A code author may also be blacklisted. These trust levels will then impact the checks completed on code; the efficiency of the authorisation process; and the eventual privileges assigned to the code as regards execution on the device, as illustrated below.

Tampering is an issue that must be considered with respect to these policy identity lists. In order to handle this problem we introduce the concept of the trusted computing base which encompasses; the abstract concept of the reference monitor, which mediates all accesses to objects; the security kernel, which includes the hardware, software and firmware of the TCB which implements the reference monitor; and the trusted computing base which includes the security kernel among other protection mechanisms. This security kernel must mediate all accesses, be protected from modification and be verifiable as correct. It is here that identity lists should be stored.

It is at this stage also that I will discuss the issue of domain server denial of service "characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service" [13]. Denial-of-service attacks come in a variety of forms, consumption of limited, or non-renewable resources; destruction or alteration of configuration information; or physical destruction or alteration of network components, each of which administrators must be vigilant for. Distributed denial-of-service attacks which involve floods of packets that originate from hundreds of other victims whose integrity has been compromised by criminal hackers and the final victims are the sites receiving a flood of fraudulent packets that can crash servers and saturate inbound bandwidth must also be considered. The implementation of widely publicised preventative measures on this topic is vital with respect to trusted domain servers.

The services of an accreditation authority may also be required if incoming code is received from an unknown author and has minimally trusted or no assertions accompanying it. Rather than discarding all code that falls into this category a national accreditation authority may exist to which code authors can register and undergo a series of generic tests to ensure legitimacy. The name of a code author on this registry may indicate to the domain server that code originating from this source should be given very minimal privileges rather than being discarded**.**

6.6 PCC

We will now move on to a mechanism described by Necula and Lee in [14] labelled proof carrying code, and it is used in order to solve the problem of establishing trust between code authors and code consumers without any reliance on cryptographic protocols. Essentially this method requires that a "safety proof that attests to the fact that the code respects a formally defined safety policy" is created and sent in conjunction with the associated code segment. On receipt of the code and the proof the code consumer can simply and efficiently check the validity of the proof with a proof validator and make a decision as to whether or not the incoming code is safe to execute.

As with other security mechanisms, this too puts forwards some challenges to be overcome. Correctness proofs depend on the programmer or the logician to translate a program's statements into logical implications and just as programming is prone to errors, so is this translation. Deriving the correctness proof from the initial assertions and the implications of statements is also difficult. In addition logical engines proposed for the generation of proofs run slowly and the speed of the engine degrades as the size of the program increases, so that proofs of correctness are even less appropriate as of yet for large programs. The current state of program verification is less well developed than code production. Questions also arise in relation to how big proofs get in practise and the large bandwidth required in sending them across the network [11].

6.7 Path Histories

Path histories are also put to use within this framework. In our particular situation we do not wish to utilize this mechanism strictly for tracking the path of the executable so as to deter malicious manipulation but to draw on it such that domain servers can see as to whether domain servers they trust have already tested the code in question as should this be the case a request can then be sent from the current domain server host to a previously visited and trusted proxy for a credential specifying the attributes relating to the code. We see possible construction of path history entries as follows until a full track log has been built:

```
signature of current location on (code
identity/hash |assertion references| the previous
location | current location, i.e. security domain
| next location).
```

Let us now consider a general overview of how the system may function, how all entities listed above may interact and communicate such that it works smoothly. We initially request that all code producers and domain servers acquire a public/private key pair and have this key pair certified by a chosen CA. New code authors may also apply to be placed on an accreditation authority register. It all begins when a user requests code from a software house, who assumes responsibility of program manufacture in conjunction with the proofs of code. Alternatively the code producer may only adopt the responsibility of code creation in conjunction with the signing of what they have created. Once the above has been completed, the code is sent to the consumer and it is at this stage that the authorisation mechanisms come into play. In the framework we propose that the first entity code meets on its journey to a destination mobile device contained in a particular network is a domain server, whose responsibility it is to ensure any code with malicious intent does not receive the opportunity to damage the host on which it wishes to execute.

7. DOMAIN SERVER POLICY EXPRESSION

As regards policy declarations, we require the expression of domain server credentials, i.e. assertions created for use by other domain servers by the current proxy acting as an attribute/authentication authority for code which has tested; attribute credentials, sent with the code from the trusted domain server to the end device; policy statements, defined for the trusted domain server in conjunction with those defined for the device itself; and finally the expression of the domain server policy engine, which will output an attribute credential for the end host use; and the device policy engine, which makes the final authorisation decision regarding code based on the attribute credentials received and the policy statements defined.

When code enters a domain server three things must occur: assertions must be either created for other domain servers or requested from trusted domain servers if they already exist; code must be checked for safety and assertions for the end device must be output. When code comes into contact with a trusted domain server the path history is initially checked and either: the code has not been in contact with any other trusted domain servers in which the current domain places its trust; no path history is in existence; or there exists a reference(s) from a TDS to an assertion in the path history. If the code has not been in contact with any other trusted domain server, the security proxy server takes on the role of both authentication and attribute authority. In this instance the incoming code presents the domain server with its credentials, for example the signature of the code author or proofs of code. On completion of checks, authentication and attribute assertions are created by the domain server and their reference is added as part of the trusted domain server's entry to the path history such that subsequent domain server's code visits may be able to request credentials by reference. This credential construction is separate from the process used to create and forward the code's attribute credential to the end host.

The first assertion to be created is the authentication assertion for the code author. In order that we can add the <CodeCondition> element to the assertions, the SAML assertion schema has to be extended. To achieve this, we utilised the element <Conditions>, which serves as an extension point for new conditions and the substitution group mechanism. In this case the extension schema defines a new element <CodeCondition>, which is a member of a substitution group which has <Condition> as a head element. The substitution group then allows the <CodeCondition> element to be used anywhere the SAML <Condition> element can be used.

The following is the schema fragment, which has defined the new type:

```
<!-- CodeCondition –>
<element name = “CodeCondition”
              type = “asamle: CodeConditionType”
              substitutionGroup = “saml: Condition”/>
<complexType name = “CodeConditionType”>
              <complexContent>
                  <extension base = “saml;ConditionAbstractType”>
                    <sequence>
                   <element name = “CodeIdentity” type = “binary” >
                    </sequence>
                  </extension>
              </complexContent>
</complexType>
```

In assertions the code identity element will contain a binary number comprised of the code hash. Before the assertion can be deemed valid by a receiving proxy the hash of the code must be checked against the code identity entry to see if they match. Following this, various code checks are undertaken and an attribute assertion is also created for the incoming code. If on the other hand a reference to an assertion created by a trusted domain server exists in the path history, a query can then be sent to a listed TDS for all the available assertions and the appropriate digitally signed assertions would be sent in return.

The initial step carried out by the domain server policy engine is to put all executables through a virus scanner such that any obvious threats can be identified promptly; the next step involves checking of the path history for security domain server identity and assertion references where domain server may be trusted either to a medium or high degree; If such a domain server exists, the assertion identities are extracted from the path history and a request is sent to the trusted domain server as was explained above; Once the credentials have been received, the signature on them are verified and if received from a highly trusted domain server the identities of the code author is extracted from the credential; If the credentials have been sent by a domain server trusted to only a medium degree or if no suitable credentials exist, the signatures of the code producer on the code hash is verified. If signatures cannot be verified, or if the corresponding certificate has been revoked, the code is immediately discarded.

Check path history: Domain server high	**Check path history: Domain server medium**	**No trusted proxy listed in path history**
Authentication credential as sufficient verification of author id	Verify author signature	Verify author signature

The identity of the code producer is then checked against trusted policy identity lists and depending on the trust values assigned to the code author and the trusted domain server, the code is put through a series of checks.

	Path history: Highly trusted domain server	**Path history: domain server trusted to medium degree**	**Path history: No report**
Code author highly trusted	No checks, code assumed safe	No checks, code assumed safe	No checks, code assumed safe
Code author medium trust	Check credential No proof verification	Check credential PCC verification	PCC verification
Code author no trust	Check credential Try PCC verification	Check credential Try PCC verification	Try PCC verification

Following the results of these checks in conjunction with identity information, a trust value is assigned to the code in an attribute assertion, examples illustrated below:

Trusted domain server: check path history	**Code author**	**Tests actually completed with success**	**Accreditation by a standards body: only considered when code author is unknown**	**highest level of trust allotted to code and the efficiency rates**
High All ok assertion 1	High 4	Code assumed safe	N/A	5 Very fast 1 sig. ver. No pcc ver.
Med/ High report S’thing wrong	N/A	N/A	N/A	Discard Very fast 1 sig. ver.
High All ok assertion 1	Low 0	PCC verified	Accreditation from standard body 1	2 Slow

8. DEVICE POLICY EXPRESSION

We will now examine the credentials, which are passed between the security proxy server and the end host. The credential accompanying the code is an attribute credential. Incorporated into it is an attribute element <CodeTrust Report>, which contains the trust value assigned to the code by the domain server.

```
<Assertion xsi:type= “saml :AttributeAssertionType”
  version= “0100”
   AssertionID = “{6738467-47378dj-hu234832}”
   Issuer = www.DomainServer1.com
   IssueInstant = “2003-05-31T13: 20:00-05:00”
     <Conditions
          Notbefore = “2003-08-31T13: 20:00-05:00”
          NotOnOrAfter = “2004-08-31T13: 20:00-05:00”>
      <asamle:CodeCondition>
       <asamle:CodeIdentity>0001101……010101</asamle:CodeIdentity>
      </asamle:CodeCondition>
     </Conditions>
        <Attribute>
           <AttributeName>CodeTrustReport</AttributeName>
           <AttributeNamespace>
                     http://ns.code-Trust-vocab.org/basic
           </AttributeNamespace>
           <AttributeValue>
                <Trust>3</Trust>
          </AttributeValue>
      </Attribute>
</Assertion>
```

Where the schema for attribute is specified in the following namespace, http://ns.code-Trust-vocab.org/basic and the following schema fragment defines the element <TrustValue>:

```
<element name = "TrustValue">
<simpleType base = "positive-integer">
  <maxExclusive value = 5>
</simpleType>
</element>
```

When the code attempts execution, an access request is sent to the policy decision point in conjunction with the relevant assertion. The policy engine initially forwards the code and the assertion to separate applications which: verify that: the code hash is equal to the bit string in the element code condition; the time is valid; and that the signature on the credential is that of the domain server. If any of these checks fail, the code is discarded.

If all the verification processes are completed successfully the code and credential are returned to the policy decision point, which assigns the code to a role depending on two inputs, the policies defined from a very simple DTD illustrated below, and the assertion content. This particular DTD, which stems from DTD developed by Herzberg et al. in [5], defines a role element comprised of a number of rules, which are made up of requirements consisting of two attributes, the issuer identity, which must be that of the trusted domain server and the trust value which must have a value between 0 and 5 inclusive.

```
Policy language:
<?xml version = 1.0"?>
<!ELEMENT          POLICY   (ROLE)*>
<!ELEMENT          GROUP    (RULE)*>
<!ATTLIST          GROUP
   NAME                     ID          #REQUIRED>
<!ELEMENT          RULE                 (REQUIREMENTS)*>
<!ELEMENT          REQUIRMENTS          EMPTY>
< !ATTLIST         REQUIRMENTS
    ISSUER         "Specific domain server identity"
         #REQUIRED
   TRUSTVALUE     (0¦ 1¦ 2¦ 3¦ 4¦ 5)    #REQUIRED >
```

We follow this with a specific policy DTD instance.

```
<!---->
<!—Code with credential that has issuer=trusted domain server and trust
value 5 will be mapped to the most trusted entity group/role>
<!---->
<?xml version = 1.0"?>
<GROUP NAME = "most trusted entities">
  <RULE>
   <REQUIRMENTS ISSUER = "identity of domain server"
TRUSTVALUE = 5>
  </REQUIRMENTS>
  </RULE>
</GROUP>
```

In this case 'role' represents a "collection of procedures assigned to code, where procedures are highlevel access control methods with a more complex semantic than read or write and procedures can only be applied to objects of certain datatypes" [15]. The assignment of a role to an executable should also lead to the transfer of a resource priority value, which it will use with regard to resource usage such as CPU or memory, while it is executing.

9. CONCLUSIONS

What has been presented is a policy-based architecture for the authorisation of software downloads for use in association with SDR technologies. This architecture is fundamentally based on the deployment of a security proxy server in every predefined domain, which is responsible for the safety verification of incoming code via the use of various security checking mechanisms in conjunction with interactions with other security proxies which are predefined by a specified policy statement. This checking process then results in the output of an attribute assertion which is used by the end host device in conjunction with predefined device policy statements to map a set of privileges to executables via the assignment of roles. As regards future work, it is clear from this paper however, that there are many obstacles to be overcome as regards the smooth implementation of this framework.

10. REFERENCES

[1] E. Gallery, "Towards a Policy-based Framework for Mobile Agent Authorisation in Mobile Systems", 3G 2003, June 2003.
[2] M. Blaze et al., "RFC 2704, The Keynote Trust-Management System Version 2", IETF, 1999.
[3] N. Damianou, N. Dulay, E. Lupu and M. Sloman, "Ponder: A Language for Specifying Security and Management Policies for Distributed Systems", Imperial College Research Report, DoC, 2001/2.
[4] OASIS, "Assertions and Protocol for the OASIS Security Assertion Markup Language V1.1", 2002.
[5] A. Herzberg , Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid, "Access Control Meets PKI, Or: Assigning Roles to Strangers", IEEE Syposium on Security and Privacy, Oakland, CA, May 2000.
[6] E. Ray, "Learning XML", pp326, 2001.
[7] R.L. Rivest and Robert D. Silverman, "Are 'Strong' Primes Needed for RSA?", RSA Data Security, 1998.
[8] W. Stallings, "Cryptography and Network Security", Principles and Practise, 2nd Edition, Prentice-Hall, 1999.
[9] A. Arsenault and S. Turner, PKIX working group, "Internet X.509 Public Key Infrastructure: Roadmap", July 2002.
[10] F. Gomes, "Security Alert: Fraudulent Digital signatures", Sans Institute, Information Reading Room, June 7, 2001.
[11] C.P. Fleeger, "Security in Computing", Second Edition, Prentice-Hall, 2000.
[12] A. Rossnagel, et al., "Die Simulationsstudie Rechtspflege. Eine neue Methode zur Technikgestaltung für Telekooperation", Berlin, 1994.
[13] CERT® Coordination Center, "Denial of Service Attack", Carnegie Mellon, Software Engineering Institute.
[14] G. Necula and P. Lee, "Safe, Untrusted Agents Using Proof-Carrying Code", LNCS 1419, pp61-91, 1998.
[15] D. Gollman, "Computer Security", Wiley, 2001.

11. ACKNOWLEDGEMENTS

The work reported in this paper has formed part of the WA1 area of the Core 2 Research Programme of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, www.mobilevce.com, whose funding support, including that of EPSRC, is gratefully acknowledged. More detailed technical reports on this research are available to Industrial Members of Mobile VCE.