12th ICCRTS
"Adapting C2 to the 21st Century"

Information Assurance for Global Information Grid (GIG) Net-Centric Enterprise Services

Track 8: C2 Technologies and Systems

Rod Fleischer, Brent Goodwin, Alex Lee,
Laura Lee, Ryan Sullivan, Vlad Tsyrklevich

SPARTA, Inc.
13400 Sabre Springs Pkwy. #220
San Diego, CA 92128
(858) 668-3570

Information Assurance for Global Information Grid Net-Centric Enterprise Services

Rod Fleischer et al
SPARTA, Inc.
rodf@sparta.com

ABSTRACT

The capabilities offered by the Internet would be of significant use on the battlefield. The envisioned Global Information Grid (GIG), effectively a military version of the Internet, will be able to provide data-on-demand to soldiers. A well-connected military force could achieve an almost "omniscient" knowledge of a hostile situation and have the ability to resolve it more effectively with less risk to coalition soldiers and equipment.

The Internet was not initially developed with security or assurance in mind at the level needed for the warfighter; attributes that are of paramount importance in situations where technological "glitches" can prove fatal. Despite this, the Internet has been a fantastic success; web services are providing a level of interoperability never before achieved. However, many of these web service protocols still do not provide an adequate level of security or assurance for military use.

In this paper, we discuss the applicability of the current web service protocols for the GIG. We examine the merits and shortcomings of these protocols, and provide recommendations for improving these technologies to provide better levels of security and assurance for military operations.

1. The Global Information Grid

Network-centric operations (NCO) is a new military doctrine of war and peacetime operations in the military and government agencies. The goal is to achieve a competitive advantage through the robust networking of well-informed, geographically-dispersed forces or, in other words, to add "information supremacy" to the military's strategic repertoire.

The Global Information Grid (GIG) is the key enabler of the Department of Defense efforts to develop network-centric operations. This network, or grid, is a globally interconnected, end-to-end set of information capabilities, with associated processes and personnel for collecting, processing, storing, disseminating and managing information on demand to warfighters, policymakers and support personnel. Supporting the GIG are "Net-Centric Enterprise Services (NCES)", designed to be the building blocks for a Service-Oriented Architecture (SOA). Putting it simply, the GIG is a government version of the Internet, which builds on lessons learned from the public Internet, but has been designed from the ground up to be an available, robust, reliable, secure and interoperable service-oriented network.

One of the fundamental problems with today's public Internet is that many of the core Internet protocols that are in use today were not initially designed to be secure or interoperable. These protocols were developed from necessity and have evolved with time, but in many cases they have arisen out of a need to "make it work." While many systems work and often get the job done, putting two existing systems together into a larger system is difficult or impossible. Making legacy programs which have different methods of data communication work together is sometimes only attainable by painstaking acts bordering on sheer will. However, it bears mentioning that this discussion is not intended to be critical of the engineers who built these protocols, as interoperability was not the key concern at the time, nor was the Internet envisioned to grow to the extent that it has. There is an engineering axiom that states "first make it work, then make it better", which is exactly what these engineers did.

1.1. Service-Oriented Architecture (SOA)

A SOA represents the current pinnacle of interoperability, in which resources on a network are expressed as individual, loosely-coupled and independent services. These "web services" are accessible by network clients without any required knowledge of the underlying platform implementation or architecture. Because of the independent, standards-based nature of these services, there is greater interoperability between system components, as well as some protection from a proprietary vendor lock-in. The implementation of a service can change dramatically, but as long as it maintains the same service contract, other services should not even be aware of the change. In some ways, a SOA can be considered as a network analog to object-oriented programming, where all network services are self-contained and provide well-defined interfaces to the outside world.

1.2. Problems with the current Internet

In the years since the development of the early internet protocols, organizations such as the Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) have worked hard to establish standards and improve interoperability. However, it has become clear that future computing needs must be data-centric, not communication-centric. Historically the problem has been about how to move data from one place to another, and with more systems becoming interconnected that has become less of a problem. Now the focus is on designing a data strategy so that end users can find and understand data that is available on the communications back-bone.

The two main problems facing the GIG architects are (1) securing data and (2) cryptographic key management. Data must be shared between applications on the GIG, which can be widely distributed geographically, or on the same local node. In either case, the problem is how to make these different applications understand the same data and, more importantly, how to keep that data secure as it moves from place to place on the network. The other problem is cryptographic key management. Before cryptographic security can be used effectively, the key material must be securely disseminated to all parties involved. It does not matter how strong the algorithms used to protect the data are if the key management is weak, since an attacker who manages to acquire a key has access to all data protected with that key.

To illustrate the problem with weak key management, let us consider the media event in which Paris Hilton's T-Mobile Sidekick was hacked. The reality of the stunt was merely social engineering, where the teenagers in question managed to trick a T-Mobile employee into giving them a store's username and password to the network servers. With valid credentials, they authenticated through proper channels to gain access to sensitive information and embarrass Paris Hilton. The analogy here is the subversion of the credential management scheme (analogous to key management, for our purposes) was far more direct and effective than a brute force attack. In other words, rather than attempting to defeat the cryptographic security of a system, a more effective avenue of attack is generally against the key management scheme (e.g., discovering or hijacking passwords or "becoming an insider"), making it a significant problem that must be addressed to improve overall security. Both data security and cryptographic key management are problems that must be addressed before the GIG will be able to achieve its goals.

1.2.1. Web Services and XML

Web servers advertise services using a well-defined interface, and communicate via an eXtensible Mark-up Language (XML) dialect known as SOAP[1]. By using SOAP over Hypertext Transfer Protocol (HTTP), all communication is done via standard, mature protocols, making interoperability relatively easy to achieve. Another significant benefit is that the web service output can be wrapped in Hyper Text Markup Language (HTML) to make it readily accessible to an end user via a web browser, without requiring vendor-specific software.

Another significant advantage of XML is that it is well-structured, which enables XML-aware firewalls to perform inspection of messages. XML is very flexible, in many schemas the data fields are optional, or can appear in any order. Additionally, XML ignores formatting, so the data fields can be organized in a human-readable tree or simply mashed together. XML has been key to bringing together Communities of Interest (COIs). Using XML, COIs are able to discuss key common terminology, algorithms and concepts without resorting to specific system language or definitions. XML has been instrumental in the move towards SOAs and web services.

1.2.2. Problems with XML

Unfortunately, one can find as many faults with XML as advantages. XML is a very verbose language, where every data item requires an opening tag and a closing tag, resulting in very large messages. Complex types comprised of multiple data items require tags on every contained data item, as well as the tags on the containing type. Scale this up to a document which has a lot of data items in large data structures, and the overwhelming bulk of the document will be tags, instead of data. Larger files imply greater bandwidth requirements. XML also requires significant amounts of processing

[1] SOAP once stood for Simple Object Access Protocol, but the acronym usage has largely fallen out of favor because SOAP is not simple and is no longer used for object access. However, it is a very effective XML data transfer protocol for remote method invocation and is commonly referred to as "SOAP".

power to marshal the data in and out of the XML syntax. Programs need to use software components known as "parsers" to format data with XML tags, or to read the XML-formatted document and extract the data items. The more data an XML file contains, the more tags are required, the greater the processing power needed to parse the document and thus the more processing time required.

Another problem with XML is that many XML documents have the same opening tag structure. With this in mind, an attacker can perform a "known plaintext" attack against an encrypted XML document. A known plaintext attack means that the attacker will know (or at least have a very good idea) what the unencrypted content in a block of encrypted data is, and will be able to determine if he has found the correct decryption key. If data is sufficiently random, then the attacker's job is harder, because even correctly decrypted data may not be immediately recognizable. XML's well-known tag structure, particularly with fixed and known military processes, may make this process slightly easier for attackers.

2. Security Challenges with SOAs

SOAs are a new way of operating a network system and, as with all new technologies, they come with their share of challenges. Of particular difficulty is the challenge of securing a service-oriented system. Due to the intentionally decentralized nature of this system, data flows in all directions and needs to be protected at all times. Additionally, to implement an access control policy, it must be first defined somewhere, and then the rest of the system needs to be aware of the rules and respect them. Since there are many resources in such a system, it becomes cumbersome to require users to authenticate themselves every time they attempt to access a new resource, so "Single Sign On" (SSO) functionality, where a user's credentials are promulgated throughout the GIG to reach all desired services, is extremely desirable. And in a SOA, all of these security functions are implemented in XML, which brings its own set of problems.

2.1. XML is not inherently secure

To understand the problem with security and XML, we must diverge for a moment to explore the evolution of XML and its design philosophy.

In the mid-1990s, the then-new World Wide Web took the world by storm. Web pages were written using HTML, which described to a web browser how to render the data contained in the document. Data was enclosed by tags which described formatting: bold, underline, indent, color, and so forth. This was revolutionary to anyone who had never heard of the Standard Generalized Markup Language (SGML) from which HTML was derived. One problem encountered in the Internet by the rapidly growing web-based businesses of the time was a need for a way to pass data in a neutral format, which could be understood by any browser or any program. Initial attempts were made to use HTML, but it was quickly discarded as being inappropriate since HTML was designed to represent the desired **formatting** of data, not the data **content** or meaning. XML, another derivative of SGML, was developed by the W3C as a general-purpose markup language for creating special-purpose markup languages, or a language used to describe data. Put simply, XML is a way of describing data, without regard for formatting or intended use,

in an application-neutral way. XML works well with HTML in that both content and meaning (via XML) and formatting (via HTML) can be specified.

2.2. W3C security protocols for XML

Since the rise of web service technology and the prolific explosion of XML usage, the W3C put forth XML-Signature and XML-Encryption, two standards for applying cryptographic security to XML documents. XML-Signature is used to digitally sign portions, or all, of a document to provide integrity of the data and the identity of the sender. This guarantees that the document has not been altered in transit, and uniquely identifies the sending party, who cannot repudiate sending the message. XML-Encryption is used to encrypt portions, or all, of a document to provide confidentiality of the data, or to guarantee that it isn't read by unauthorized parties during transit.

2.2.1. XML's Achilles' heel

XML's greatest strengths, its flexibility and complete application-neutrality, are also its Achilles' heel from a security standpoint. The problem with XML-Signature is in the formatting of the XML-tagged data or, more specifically, the lack of any required format. XML parsers are free to format or re-format data in any way they choose, a flexibility which is at odds with the concept of a digital signature. A digital signature is generated across a block of data by hashing, or generating a unique fingerprint of the data, and then transforming the hash value with a private key value. A signature is then validated by taking the transformed value, applying a second transformation with the public key, and comparing the resulting value to a freshly-generated hash value of the content data. If the two are equal, the signature is valid and the data hasn't been modified in transit. The problem arises from XML parsers being free to change the formatting of the data. This results in the hash functions generating different values from differently formed data, because a good hash function will generate a wildly different result if even only one bit is different in the underlying data, making differences in the data very apparent. In short, if two parsers format the same message differently, which is very likely (and currently happens) given the vendor options and approaches, XML-Signature validation will fail. The proposed solution to this is to have the signature generation and validation include an extra step before generating the hash value. This extra step, which re-formats the data according to specific requirements, is a process known as canonicalization. The process guarantees that two versions of the same message, formatted differently, will yield an identical result whose formatting may or may not match either of the originals. The security problem with inserting this additional processing of the data is that the signature value that is generated corresponds to the canonicalized version of the data, and not the original data which was submitted to the signature function. A malicious person with the requisite access could alter the canonicalization process to insert or modify data before generating the signature value. The result is that it could be used to send malicious data through the system as though it had been sent by an authorized user.

XML-Encryption suffers similar vulnerabilities due to the flexibility of XML. Specifically, XML does not mandate that data fields appear in a document in a specific order. The encrypted XML data is not guaranteed to be ordered in a way that supports one-pass processing of the data. If the cryptographic parameters required for successful

decryption appear after the encrypted data payload, then the entire payload (which can be large) needs to be buffered until the document is completely transferred. In these instances, memory requirements to buffer the message may exceed the available memory, or processing time for a large message may become exorbitant. Both of these problems are a recipe for a denial of service (DoS) attack. Additionally, encrypted data must be encoded in Base-64 before being embedded in XML, which generally increases message sizes by approximately 30%. This makes encrypted communication require even more bandwidth, which is a problem since XML is very verbose and not bandwidth-friendly to begin with.

Another significant issue with XML-Encryption is a recursive processing attack. [NSAWSVA]. In this attack, encrypted data may reference an encrypted key, which refers to another encrypted key, which refers back to the first key. Alternatively, the encrypted data could reference network resources that are cyclical. In these cases, there is a DoS problem due to the endless processing of this malformed data. In a later section, we will discuss methods of solving these problems with XML.

2.2.2. Server-side access control policy

The eXtensible Access Control Markup Language (XACML) [XACML] is an XML dialect for the server-side representation of access control policy and access control decisions. These rules can be expressed in an application-independent manner, making it versatile. XACML polices can reference other policies, and can intelligently combine policies with competing or overlapping rule sets. If the provided combination algorithms are not sufficient, application developers can define their own as needed.

XACML can be used to implement Attribute Based Access Control (ABAC). Traditional access control methods such as Identity Based Access Control (IBAC), or the newer Role Based Access Control (RBAC), associate access permissions directly with a subject identity, or with the role that subject is attempting to perform. IBAC, in which an access policy needs to be defined for every identity, is a method which does not scale well and is repetitive and redundant. RBAC requires that access policies be defined for all roles in the system, and then subject identities are mapped to those roles. This scales better, but still has limitations from this one-dimensional view. RBAC generally requires a centralized management of the user-to-role and permission-to-role assignments, which is not well suited to a highly distributed environment, or to an environment with subjects and resources belonging to different security domains. ABAC is a newer method in which policy rules are defined on attributes of subjects (users, applications, processes, etc.), resources (web service, data, etc.), and environment (time, threat level, security classification, etc.). This allows for a much finer-grained access control policy than what can be achieved with RBAC. Of particular note is the ability to use security classification labels to create rules, allowing for XACML policies to be used in conjunction with the needs for a secure operating system's Mandatory Access Control (MAC) system.

2.2.3. Client-side access control assertions

The complement to XACML for client-side access control is the Security Assertion Markup Language (SAML) [SAML]. This XML dialect is used by the requestor of

policy decisions to assert an identity and request permission to access resources. The server handling the request consults a XACML policy to render a decision, which is returned as a SAML response. SAML can be used in some environments to achieve a SSO capability, where users only need to authenticate themselves once. SSO is a "ticket-granting" authentication mechanism, whereby a requestor is supplied with a token that proves that it has successfully authenticated to an identity server "at a particular time via a particular authentication method." Other servers in the system may accept this token without requiring additional authentication, which simplifies the user authentication process. The problem with this sort of mechanism is that it is often vulnerable to a "replay" attack, where a malicious intruder captures the authentication token and then re-uses it to impersonate a valid user. SAML proponents claim that this protocol is not vulnerable to replay, because of timestamps in the digitally signed token and because the tokens are transmitted via Secure Socket Layer (SSL) connections. There is also a hypothetical "man-in-the-middle" attack, shown in Figure 1, where a corrupted server could receive a valid authentication token, and then replay it against another server as illustrated in the figure below. A "man-in-the-middle" attack emphasizes the potential vulnerability stemming from the abuse of Single-Sign-On security tokens because authentication is done only once and then decoupled from access control.

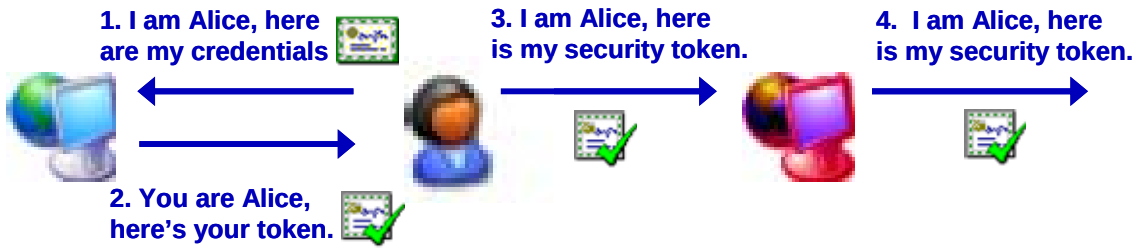

Figure 1. Hypothetical SAML "man-in-the-middle" replay attack.

This attack is unproven, but serves to emphasize the point that decoupling authentication from access control is dangerous, because it is possible for an authentication token to be used by someone other than the authenticated party. This is akin to the problem with online banking where an attacker steals a password and empties someone's bank account. The system wasn't hacked; rather, valid credentials were used inappropriately. But at the time the transaction took place this was difficult, if not impossible, to determine.

3. Strategies for Mitigating SOA Vulnerabilities

Now that we have laid out the vulnerabilities and security problems associated with the XML-related technologies, we can move forward into strategies for mitigating these shortcomings. In this section, we will provide recommendations for the current breed of technologies and, in a later section we will cover strategies for the general evolution of security technology in service-oriented architectures.

3.1. Defending against replay attacks

SAML assertion responses, which contain decisions about whether a particular action is to be authorized or not, should always be digitally signed. This is especially true in the case where SAML is used to implement SSO, because the assertion passes through the

user agent and is thus susceptible to tampering. SAML proponents argue that because the assertion is required to be transmitted to the client via an SSL connection, it is protected against replay. [SAMLMYTH] This argument is naïve because it assumes that the data is safe, just because the connection is encrypted. Unfortunately, the assertion will be decrypted on the user-end of the SSL connection, then encrypted again as it is sent to a new server. This window between the decryption and subsequent re-encryption is a vulnerable point at which the assertion can be tampered with, so a digital signature is required to guarantee the integrity of the assertion data. Additionally, the signed assertion should include a timestamp to prevent the token from being replayed beyond a certain time. Ideally this time window would be fairly small, and the authentication tokens would be re-issued on a somewhat frequent basis. The assertion should also include the IP addresses of the issuing server and the authenticated party, so that attempts to replay the assertion from a different computer will be detected.

SAML also defines an artifact (i.e., nonce) mechanism where the authorization request is too long for an HTTP redirect. This artifact is 40 bytes long and is used by servers to look up an assertion, so the server must maintain a table mapping the artifacts to the assertions. It is claimed that the SAML server will only return the actual assertion to the requestor who received the artifact and that it will remove the table mapping after the first assertion retrieval. [SAMLMYTH] This does not seem particularly useful in the context of SSO, since it can only be used once, but if used, this transaction needs to be similarly protected via the use of an SSL connection and a digital signature across the transmitted data.

Fundamentally, many of the significant vulnerabilities in SAML arise because the mechanisms for preventing these weaknesses are optional, and not enabled by default. [NSAWSVA] The following **optional** security features should be considered **mandatory**:

- Mutually suspicious authentication between web services and clients
- Digital signatures to ensure integrity on all SAML data
- Transport layer (SSL) and/or message payload encryption
- Binding of SAML responses to specific requests
- Use of the optional elements and attributes (a complete list can be found in NSAWSVA reference, section 4.4.2).

It is also recommended that the DoD Public Key Infrastructure (PKI) be used as the basis for all authentications in the NCES / GIG architecture. Strong PKI authentication on all communications would significantly mitigate the “man-in-the-middle” vulnerabilities in the SAML architecture.

3.2. Defending against encryption vulnerabilities

When applying XML-Encryption to data in a SOA framework, certain precautions need to be taken to prevent information from being revealed to unintended recipients. [NSAWSVA] In a symmetric encryption system, where a key is shared among a group of subjects, care must be taken to only encrypt data with that key which is suitable for

dissemination to all subjects who may possess the key. Even if the data is not routed to a recipient who is not authorized for the information, if that recipient somehow intercepts the message, they can decrypt it. Along this same thread, encryption parameters and algorithm identifiers should be checked to ensure they do not reveal any information which may compromise the encryption key.

The processing power required to decrypt data, especially asymmetrically encrypted data such as digital signatures, can be significant. As such, the DoS attacks involving cyclical dependences described above can be particularly damaging. A simple solution is to "time out" an unresponsive web service after a specified interval. An alternate, and perhaps better, solution is for web services to be engineered to restrict arbitrary recursion and the total amount of processing time and network resources that a request may consume. Care should also be taken to perform validity checks on messages in increasing order of complexity, with asymmetric operations (such as signature validation) last, so that a server will be more resistant to being overwhelmed by processing invalid messages.

Encryption can also be used to hide data that a firewall or anti-virus scanner might reject. The strategies for dealing with this include:

- Disallowing encrypted content in contexts where it could carry unsafe data
- Inspecting the decrypted data prior to processing (some firewalls can perform this action)
- Ensuring that the application can process unsafe data without jeopardizing security

If possible, XML-Encryption messages should be formatted to ensure that the decryption parameters appear in the message prior to the encrypted content. However, this may or may not be possible due to the nature of XML parsers.

3.3. Defending against policy vulnerabilities

XACML is well-suited to defining policy for the GIG, but it does not include any requirements for security mechanisms to provide confidentiality, integrity, non-repudiation, authentication or confidentiality requirements for communications between policy points. [NSAWSVA]

If policy is transmitted without protection, it is susceptible to tampering. Such tampering could result in attackers granting themselves access or even disabling security policy altogether. The solution is to ensure that SSL or XML-Encryption is used to provide confidentiality of the policy data in transit, with digital signatures being required to guarantee the authenticity of the policy.

Additionally, the XACML implementation is responsible for correctly combining the policies and making the correct decisions. It is essential that the policies be written correctly and, as XACML continues to mature, there will be more and better tools available to help with this. However, the security of a XACML implementation depends heavily on the security of the underlying components, such as the operating system. A

compromised operating system could lead to policy violations or modification of the policy without detection.

3.4. Bandwidth considerations

Many of the vulnerabilities in the GIG approach are related to DoS from bandwidth starvation. XML is a very verbose language, requiring many tags to present even the smallest amount of data. Cryptographically-secured data would need to be encoded in Base-64 in order to be comprised of characters legal for an XML document, and that encoding typically increases the size of the data by approximately 30%. Available bandwidth will likely be the critical factor for much of the intended infrastructure.

Much of the GIG communication required is going to be in the battlefield where there may be little-to-intermittent connectivity. The Warfighter Information Network – Tactical (WIN-T) networks do not have large bandwidth, and the Mobile Ad-hoc Networks (MANETs) used in the field are challenged to provide adequate wireless coverage to a soldier hiding in a foxhole. Handheld and embedded devices typically have limited memory and processing capability, so cryptographic operations, particularly the Public-Key Infrastructure (PKI) operations required to authenticate data, may be very time-intensive.

4. DISA's Approach to NCES Security

The authorization process for the Defense Information Systems Agency (DISA) Net-Centric Enterprise Services (NCES) architecture is distributed across multiple processes. It is built primarily on SAML and XACML, but also uses the XML Key Management Service to provide distributed use of PKI.

Figure 2 shows the various services employed in DISA's NCES approach, and the protocol interactions between them. The authorization process is shown from the perspective of the fictional user Sam, who tries to access a protected data service. The only direct interactions that Sam has are between his browser and the firewall (which is considered part of the Policy Enforcement Point (PEP) in the GIG architecture) and subsequently between his browser and the service, assuming he is able to successfully authenticate past the firewall.

4.1. Authorization Process

1. Sam attempts to reach the desired service, which could be any user data service. Given that he has to authenticate to reach the service, in this instance it is likely that he is initially attempting to contact the federated search service. The firewall / PEP between Sam and the service requires that he initiate the request via HTTPS and demands his client SSL certificate.
2. The PEP sends Sam's SSL certificate to the Certificate Validation Service (CVS) via an XML Key Management Specification (XKMS) query for validation.
3. The CVS attempts to verify Sam's certificate against the PKI, and returns a response value equivalent to "Good" if validation succeeds, "Revoked" if Sam's identity has been compromised or restricted, or "Unknown" if validation is otherwise unsuccessful.

4. Assuming a "Good" response in Step 3, the PEP sends a SAML assertion with Sam's identity to the Policy Decision Service (PDS) querying whether Sam is authorized to reach his desired service.
5. The PDS does several things at this step. First, it sends a SAML query to the Principal Attribute Service (PrAS) requesting all of Sam's relevant attributes. Also, if necessary, it queries the Policy Retrieval Service (PRS) for the latest policy.
6. The PrAS returns a SAML response to the PDS, the payload of which is Sam's attributes in XACML format.
7. The PDS evaluates Sam's attributes against the network policy and determines whether or not Sam is authorized to access the desired service. This decision is returned to the PEP as a SAML response.
8. Assuming a "Yes" response from the PDS, the PEP allows Sam's HTTPS query to proceed to the desired service.

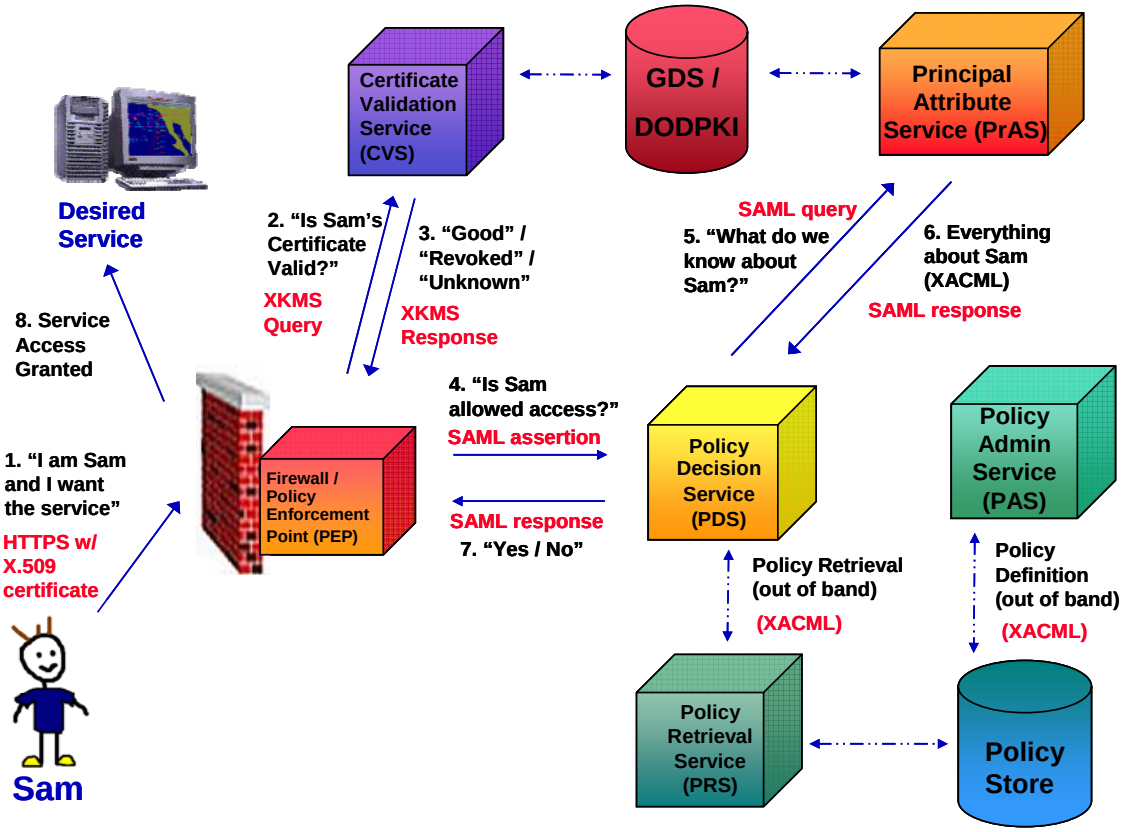

Figure 2. NCES Security: Authorization process for authenticating a user and evaluating his ability to access a desired service.

4.2. Single Sign On (SSO) in a NCES environment

With a system containing multiple web-based applications, where each one requires a different set of credentials, the task of managing all the application logins becomes significant and overwhelming. It becomes even worse when the applications span domains which may or may not immediately communicate personnel or technology

changes, leading to gaps in the system. Unused accounts may continue to have access, or users requiring access may fall through the cracks.

Identity federation is the concept of establishing standards and security models to achieve trust between disparate domains. This dovetails with the concept of an SSO, giving each user one set of credentials which is accepted everywhere that the user has access, even in different trust domains. This works well for the user, since he or she only needs to remember one password (or keep track of one private key), and it works well for the system administrators for the same reason.

Additionally, SSO could significantly reduce the network workload since authentication is done once, and the user is provided a security token which can be used to bypass additional authentication. Following our earlier diagram, assuming that Sam was granted a valid security token in his initial attempt to access the service as described in section 4.1, let us explore how a subsequent authentication attempt would work.

1. Sam attempts to access the desired service, and contacts the PEP via HTTPS.
2. Sam presents his digital certificate and security token to the PEP.
3. The PEP verifies Sam's certificate with CVS.
4. The PEP verifies the PDS signature on Sam's security token, and then evaluates the permissions in the token against the service Sam is attempting to access.
5. If the permissions match, Sam is allowed to access the service.

This approach is quite obviously more streamlined and efficient, but it has potential vulnerabilities that must be addressed. First and foremost, the security token is allowed to pass through the user agent and is thus susceptible to tampering. It is imperative that the PDS digitally sign the security token, and the signed data must include an expiration time stamp. The token should also explicitly state what services the user is authorized to access for the duration of the token's value. In addition, the token should state the IP address from which the user has authenticated, to prevent fraudulent use of the security token by intercepting parties. The use of IP address is not an ideal solution, as some systems may involve Network Address Translation (NAT) and multiple users may appear to be from the same IP address, but it still provides some security against interception to users whose connections are not using NAT, or while the connections are outside the NAT. Identity verification with the digital certificate of the user agent should always be performed before accepting a security token.

Ideally, PKI should be used for authenticating users within NCES. This can be accomplished with a Common Access Card (CAC), which contains the user's private key and certificate on a tamper-resistant chip. A Personal Identification Number (PIN) is required to access the private key, which is never released from the card, and all computations involving the key are done on-chip. From the user perspective, his single sign-on is accomplished by entering the PIN for his CAC. His digital identity will then be evaluated by the PEP/PDS as described in section 4.1, at which point his user agent will be issued a security token. As with the security tokens, the CAC activation should only be valid for a specific period of time, at which point it must be re-authenticated. For

the user, this means re-entering the PIN. However, NCES policies to date are unclear about how often this re-entry should be required.

Username/password authentication could also be used, but this is more complex to implement and manage, and harder to secure. This is for several reasons, but a prime vulnerability of password authentication is that passwords must be comprised of characters which can be typed on a keyboard. On a US keyboard, there are 94 possible characters. To try all possible permutations of an 8-character password would take approximately 9.8×10^7 years if we could try one permutation per millisecond. However, a 2048-bit private key value, which would be used for PKI authentication, would have 2^{2048} possible permutations. At a rate of one permutation per millisecond, it would take 10^{606} years to try them all. (This is longer than the expected lifetime of the universe.) It is much, *much* harder to guess a private key value. Additionally, passwords can be forgotten, written down, lost, etc., which requires central management of passwords to mitigate these problems. The storage of a password is less secure than a private key, which resides on a CAC and never leaves it. A compromise of the password database would affect every user on the system, whereas a compromise of a user's private key affects only that user. Encryption must also be employed to securely send passwords to the PEP for evaluation, which generally requires PKI anyway. If password authentication is to be supported by NCES, guidance from the National Security Agency (NSA) should be solicited to ensure that it is done with minimal vulnerability.

4.3. Multi-Level Security

In order to be truly effective, the security architecture needs to address the entire system. If we consider the system in the scope of the Open Systems Interconnection (OSI) 7-Layer model, SOA is an Application layer protocol. SOA uses encryption at the Application layer, possibly SSL at the Transport layer, and firewalls generally act at the Network layer, as illustrated in Figure 3.

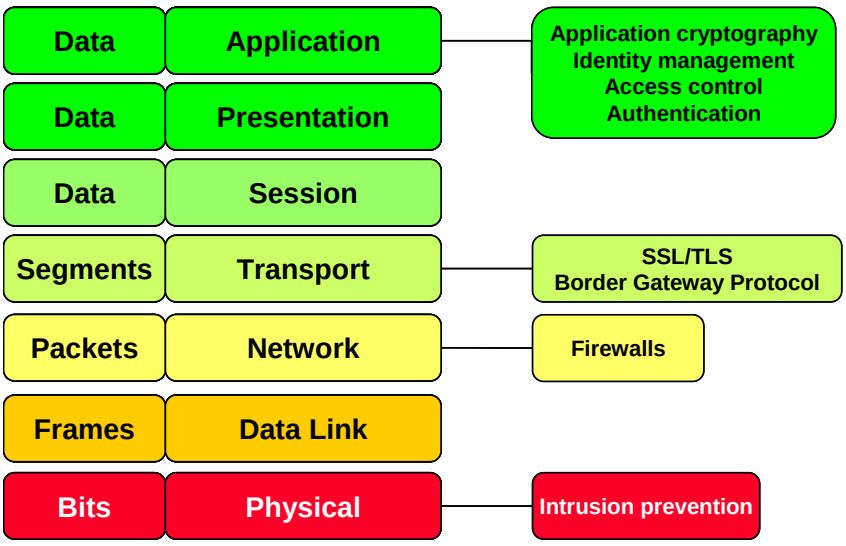

Figure 3. The 7-Layer OSI Network model, showing security features which should be implemented at various levels of the network system.

One obvious application of Physical layer security is to keep the servers under lock and key. This is, of course, an important security measure in any network context, but it is also important to leverage the hardware for assurance in the operations. A Trusted Platform Module (TPM) is a tamper-resistant hardware component designed to handle cryptographic keys, generate random numbers, and perform cryptographic operations using the contained keys. A Trusted Operating System (OS) should make use of a TPM (now commercially available) for the requisite security operations. Because the TPM is used to control access to the private keys, they are kept more secure than in application space where they may be compromised. However, this also relies upon an SOA to make use of the functional hooks provided by a Trusted OS, instead of continuing to handle security operations at the Application layer.

Building on the concept of a Trusted OS is Multi-Level Security (MLS). MLS is different from Multiple Single-Level (MSL), although the two terms are quite similar. MSL relies upon a network being divided into distinct and separated networks, each with its own security level. The best known examples of these are Non-Classified Internet Protocol Router Network (NIPRNET), Secret Internet Protocol Router Network (SIPRNET), and Joint Worldwide Intelligence Communications System (JWICS) for Top Secret processing. While this strategy works in some instances, it introduces multiple problems with moving data (even allowable data) between these networks, and is simply an unwieldy solution in a crisis situation requiring information sharing. MLS is a much more elegant answer to this problem. The difference lies in that MLS may have multiple network security levels connected to the same machine, with a Trusted OS maintaining the boundaries between these layers and preventing unauthorized data flow. Such a system allows acceptable data to be brokered easily between the multiple security levels. This is advantageous when attempting to handle emergency situations requiring quick responses. An example of when this would have been valuable is the recent tsunami in Indonesia. The first responders on the scene certainly didn't have access to SIPRNET; they probably had, at best, a wireless connection to the plain old Internet. If they had SIPRNET computer systems, they would not have been able to connect to the Internet due to the MSL requirements (only Secret systems can be connected to SIPRNET, and nothing classified above or below Secret). However, an MLS system could contain secure data and be connected to the Internet at the same time, because it is capable of safely compartmenting data due to the trusted OS and hardware.

5. Recommendations

DISA has put a lot of thought into the approach which they have adopted for GIG NCES. Their strategy, on the whole, will result in a working system. As is quite typical in the history of networking, once the system is working, it will have room for improvement. In this section, we will elaborate on our recommendations for iterative improvements to the GIG NCES architecture.

5.1. Incorporate Multiple Forms of Access Control

Security needs to be applied at multiple levels to be effective. Access control is an important element of security that should also be applied in multiple ways, as well as at

multiple levels. A coarse-grained access control implemented in a high assurance manner should be used to control general access to the system, but access to specific information should require a second access control evaluation.

A coarse-grained access control can be implemented as RBAC, which associates subject identities with roles to be performed. For example, this might define a system administrator role, and allow significant access to the users associated with performing that role. However, a general user role would carry much lower system privileges. A second, finer-grained access control should evaluate specific user attributes in order to render a decision using a process known as ABAC. In this instance, we can achieve an even finer access control allowing more specific assignment of privilege based on user attributes. For example, these attributes could reflect specific clearances which should be evaluated to access compartmented data. The analogy here is that of giving someone with a Secret clearance access to a specific compartment for which they have a need-to-know, instead of just granting them a blanket Top Secret clearance.

5.2. Perform Architecture-Based End-to-End Workflow Analysis

Any proposed architecture needs to be evaluated for its effectiveness against security challenges. There are tools and techniques available which can greatly assist in both qualitative and quantitative analysis. A particularly useful technique is workflow analysis. This approach decomposes the problem into smaller and smaller sub-problems, addressing each in terms of inputs, processes, and outputs. This is repeated until the sub-problems are at the appropriate granularity required for the analysis. More advanced workflow tools allow this process to go even further, providing a means for deploying and monitoring the execution of working versions of the modeled processes. This effectively accomplishes a complete circle, from modeling a process in order to analyze it, and then monitoring the process with the results of the modeling analysis. As an example, consider a model of the XACML Policy Authoring Process shown in Figure 4.

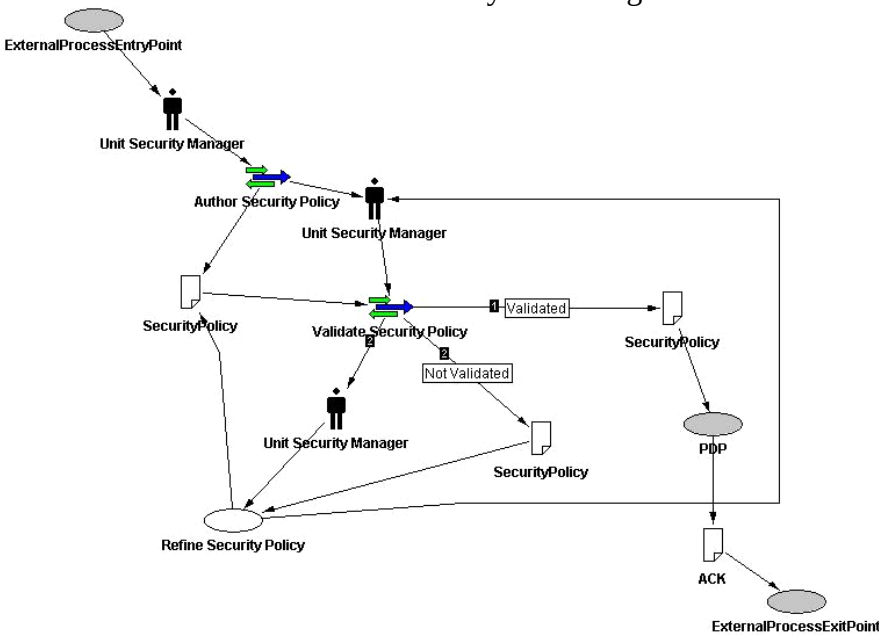

Figure 4. The XACML Policy Authoring Process Model

This process shows the Unit Security Manager initially authoring the security policy, then entering a refinement loop until the policy is validated, at which point it is submitted to the PDP, which transmits the policy into the Policy Store. The policy authoring, validation, refinement, and PDP storage procedures are detailed separately as sub-processes.

Workflow engines allow an analyst to simulate the execution of these process models in order to validate the end-to-end architecture and reveal potential issues and roadblocks which may not be readily apparent. Once the process is deployed, its activity can be monitored to assess its performance against the expected model behavior to ensure that it remains within the projected limits.

5.3. Use Cryptographic Message Syntax

XML-Encryption and XML-Signature, while somewhat effective at what they provide, are classic examples of forcing a square peg into a round hole. The flexibility of XML is fundamentally at odds with the specific exactness required for cryptographic data. In their enthusiasm to use XML data formats, the designers of these protocols left their system open to DoS attacks which are relatively easy to achieve.

XML-Encryption does not provide any guarantees that the security parameters will arrive in the correct order to guarantee one-pass processing. It would be trivial to ensure that an XML document is formed such that the parameters follow the content data, thus requiring that the content data be buffered in memory while waiting for the parameters. With multiple large messages, this can easily create an “out of memory” condition.

Digital signatures require that the content data be exact, as even a difference of a single bit will cause verification to fail. XML-Signature requires that all data be canonicalized to a known accepted format before signature values are generated or verified. Omitting or damaging the canonicalization process will lead to a DoS scenario where all messages are dropped because their signature values cannot be verified.

These vulnerabilities are due to the flexibility of XML, which is its greatest strength except where cryptographic data is concerned. A better solution would be to continue to use XML for the content data, but to apply security through other means.

Cryptographic Message Syntax [RFC3852] was developed specifically for the purpose of transmitting cryptographically-protected data in a known, accepted format that guarantees optimal parameter ordering in order to enable one-pass processing of the message. Cryptographic Message Syntax (CMS) was developed by the Internet Engineering Task Force (IETF) Information Assurance (IA) community for use with applications requiring high degrees of assurance. The High Assurance community has already accepted CMS as a suitable data format, as the CMS specification underwent tremendous peer review and revision before it was published as an Internet Standard. As such, CMS is a mature protocol carrying a high degree of assurance, which the XML technologies (XML-Encryption, XML-Signature, SAML, etc.) are too immature to match.

CMS provides multiple data mechanisms, including plaintext content data, hash values, signed content data, symmetrically-encrypted content, symmetrically-encrypted content with an asymmetrically-encrypted session key, and authenticated data using a Message Authentication Code (MAC). Additionally, all of these data formats can be nested, leading to combinations of formats for increased protection. It is perfectly valid for a symmetrically-encrypted data envelope to contain an authenticated data envelope as its content type, which in turn contains the plaintext content.

CMS presents additional benefits to bandwidth utilization, which is a significant consideration for end users on the battlefield. CMS message formats use Abstract Syntax Notation One (ASN.1) Distinguished Encoding Rules (DER), a very tightly packed binary format. ASN.1 is much more compact than the extremely verbose XML, yielding a major win with respect to packet size on a low-bandwidth connection. Additionally, ASN.1 is an extremely mature data format with parsers readily available.

The data format is not new and is, in fact, very prevalent in many modern technologies. Secure / Multipurpose Internet Mail Extensions (S/MIME), a protocol for sending cryptographically protected email, uses CMS as its data formats. The NSA's firmware update protocols communicate with CMS, and the Group Security Policy Token [RFC4534] top-level data format is a CMS signed-data envelope. The Trust Anchor Management Protocol (TAMP) also makes use of CMS for its communications. Additionally, the DODPKI, the public key infrastructure which is fundamental to the U.S. Government and military security infrastructure, uses X.509 cryptographic certificates. The X.509 specification requires the use of CMS as the certificate data format, so naturally CMS structures are extremely compatible with X.509 PKI operations.

Additionally, CMS is not tied to any particular key management scheme, so any key management scheme can be used. There are built-in interactions to handle both shared symmetric key as well as asymmetric PKI schemes. CMS is also compatible with the Group Secure Association Key Management Protocol (GSAKMP) [RFC4535].

5.4. Employ Group Secure Association Key Management Protocol (GSAKMP) for Cryptographic Groups

A fundamental design concept of GSAKMP [RFC4535] is that access control through key management provides higher assurance than policy enforcement alone. Put another way, access control must be properly negotiated to acquire cryptographic keys, which are in turn required to access sensitive data. Even if an attacker somehow circumvents the system and manages to receive sensitive communication, the data is encrypted with keys that are only granted upon successful negotiation of the access control.

GSAKMP provides management for cryptographic groups. Encryption and authentication keys are securely disseminated to all group members, and the key servers act as both policy decision and enforcement points as they evaluate potential group member identities against the access control policy. The group policy is also distributed via the Group Security Policy Token [RFC4534] to new members, who are in turn empowered to act as enforcement points for application policy.

The Group Security Policy Token, itself a CMS signed-data structure, provides rules for every aspect of group operation. These include general access control rules for group membership, rules for acting as a key server or group controller, protocols required to access the group for management, protocols required to access group communication, and security mechanism requirements for all the aforementioned protocols. The current version of the Token format specifies access rules as regular expressions against the X.509 subject and issuer identifiers, and does not currently support rule languages such as XACML for its data format, although this is planned for future enhancements.

5.5. Protect the Data, Not Just the Network

While network security is important and should not be treated lightly or ignored, the critical aspect of security is the data. The data is all-important, and the network is merely a delivery vehicle. If the data is secured independently from the network infrastructure, then the system is less complicated, has less points of vulnerability, is more robust, and is easier to accredit. Additionally, the separation of security and communication frameworks makes it much easier to change one without impacting the other. If the data is secure, it doesn't matter how it gets to its destination; it is protected regardless of its path through the network.

Security for data largely falls into two distinct categories: **data-in-transit** and **data-at-rest**. While in transit, data can be encrypted with session keys negotiated between the sender and the receiver, which are only used for the duration of the communication. When at rest, in a file-system or database, protections on data must be more permanent than in session-oriented communication. Keys must be protected, and access control to the data must be rigidly enforced.

For data-in-transit, an ideal solution for the key management is GSAKMP [RFC4535]. As discussed above, GSAKMP provides management for ad-hoc cryptographic groups. These groups can be created as needed in a decentralized peer-to-peer environment with strong policy enforcement, and are ideal for group-oriented collaboration and coalition interoperability. GSAKMP provides symmetric keys to groups without communication requirements, so the encrypted data can be distributed on any desired communication channel without any loss of data protection. For better interoperability, CMS data structures should be used to envelope the data and ensure that it is distributed in recognized format, although this is merely a strong recommendation and not a strict requirement.

Data-at-rest presents an altogether different problem. Because the data is staying in the same place for an indeterminate amount of time, it cannot be encrypted in the temporary "session" fashion that data-in-transit is. The decryption key must be stored locally so that the data can be retrieved, so the true problem becomes one of key management and secure storage.

5.6. Utilize Trusted Platform Module (TPM)

Logically, it follows that if the data to be stored is sensitive, the decryption key for that data should be at the same sensitivity level. The use of a Trusted OS provides the assurance that we require to store sensitive data, and the use of a Trusted Platform Module (TPM) provides the assurance to store sensitive key material. A TPM offers the capabilities to securely generate cryptographic keys and restrict the usage of keys to specific functions. TPM also provides remote attestation, an unforgeable summary of software contained on the computer, allowing a third party (such as an auditing agency) to verify that the software has not been compromised. Other capabilities are sealing, where data is encrypted such that it can only be decrypted by the same computer running the same software where it was encrypted, and binding, where the data is encrypted using a unique asymmetric key located on the TPM chip or another "trusted" key. Where the idea of binding is used to implement Digital Rights Management (DRM) in the consumer world, it is also quite useful in a MLS system.

Let us pause for a moment and explain the data storage concept in more familiar terms. When a song is purchased from iTunes, the downloaded music file is encrypted. The key used is the key belonging to the purchaser, effectively "binding" the music file to the computer authorized by the purchaser. When the purchaser wants to move his iTunes library to another computer, that second computer must be authorized in order to play the music. This means the user must authenticate himself to the iTunes Store, which will download his decryption key to the second computer. The user is unable to access the music files on any computer that has not been authorized to play his music, so he cannot just distribute the music to his friends, because they won't have access to his decryption key. This functionality is known as DRM, and the analogous TPM function is binding. The access to the music files is controlled by the key management; without access to the keys, there is no access to the music. This is an example of access control through key management providing higher assurance than policy enforcement alone, and applying security at multiple levels, as discussed in Section 5.1.

A trusted operating system with TPM dovetails nicely with the concept of a MLS system, as described in Section 4.3. Data stored on a computer can be "bound" to a particular security compartment. The TPM can enforce access to the keys, only allowing users authorized for that compartment to decrypt data for which they have the proper rights. Users cannot access data in other compartments because they cannot access the decryption keys required to view that data. This is much more useful than the Multiple Single-Level solution, which requires that all data and all systems be at the highest classification level of the data stored on the network (e.g. SIPRNET). Because the trusted OS and hardware will rigidly enforce the compartmental separation of the data, a MLS system can be connected to different networks with different classification levels, perhaps even to systems without any classification, such as the public Internet.

5.7. Create Secure Group Objects (SGOs) for data at rest

GSAKMP is very useful for creating ad-hoc cryptographic groups and securing data-in-transit, but it is also capable of securing data-at-rest. Conceptually, a Secure Group Object (SGO) is defined as a group resource (typically a file) which is encrypted with a

GSAKMP group key. The encrypted data is enveloped with metadata about the GSAKMP group to which the data belongs. Since the data content is encrypted, the SGO can be published, transmitted, or stored in any desired fashion. Only authorized users with the ability to participate in the GSAKMP group will be able to retrieve the necessary decryption key to access the SGO content data.

This is very similar to the TPM solution; the only difference is that GSAKMP will maintain access to the group key material instead of TPM. The GSAKMP key server can be located on another computer, so multiple data repositories could store data with GSAKMP keys. For a very widely distributed system, such as the GIG, it may be quite desirable to have replicated databases all over the world which use GSAKMP to coordinate the decryption keys for the stored data.

6. Conclusions

The Global Information Grid (GIG) architecture will benefit significantly from leveraging Service-Oriented Architecture (SOA) principles and technologies. The existing web service protocols can be used to achieve a working prototype if certain precautions are taken; however, the security and assurance of the system can be significantly improved in a future spiral by replacing these protocols with those published by the Information Assurance (IA) community.

Cryptographic Message Syntax (CMS) [RFC3852] should be used instead of XML-Encryption and XML-Signature, as it has been accepted by the High Assurance community. The XML security protocols carry denial of service vulnerabilities due to the flexibility of XML, a trait which is not shared by CMS, which requires exact ordering of the data at all times. CMS-protected data payloads can be sent through SOAP messages to achieve a higher level of security and assurance than that which can be achieved by the XML security protocols.

The Group Secure Association Key Management Protocol (GSAKMP) [RFC4535] should be used to provide robust and reliable cryptographic key management. Coupled with the Group Security Policy Token, GSAKMP provides distributed access control via the key management scheme, and achieves higher assurance than simply enforcing policy. GSAKMP is useful in ad-hoc groups to provide key management for data-in-transit, but it is also useful for providing protection for data-at-rest. It is possible to have a replicated database of encrypted Secure Group Objects (SGO) (files encrypted with GSAKMP keys) which uses GSAKMP for key management and access control.

Trusted Multi-Level Security (MLS) systems should be used for the compartmenting of data at rest and, in the cases where GSAKMP is not applicable to secure the data, a Trusted Platform Module (TPM) should be used to bind the data to the appropriate compartment. The additional benefits of using MLS systems to broker data between systems at different classification levels make it an even more desirable solution.

With these recommendations, future spirals of development on the GIG will be able to achieve far greater levels of security and assurance than those reachable today.

References

1. [NSAWSVA] NSA, *Web Services Vulnerability Assessment*, 2004.
2. [SAMLMYTH] IBM, *Debunking SAML Myths and Misunderstandings*, http://www-128.ibm.com/developerworks/xml/library/x-samlmyth.html
3. [RFC3852] *Cryptographic Message Syntax*, Housley, 2004. http://www.ietf.org/rfc/rfc3852.txt
4. [RFC4534] *Group Security Policy Token v1*, Colegrove, et al., 2006. http://www.ietf.org/rfc/rfc4534.txt
5. [RFC4535] *Group Secure Association Key Management Protocol*, Harney, et al., 2006. http://www.ietf.org/rfc/rfc4535.txt
6. [XACML] eXtensible Access Control Markup Language v2.0, OASIS, 2005. http://www.oasis-open.org/specs/index.php#xacmlv2.0
7. [SAML] Security Assertion Markup Language v2.0, OASIS, 2005. http://www.oasis-open.org/specs/index.php#samlv2.0