An Ontology-Driven Approach Applied to Information Security

Artem Vorobiev

Australian Graduate School of Entrepreneurship
Swinburne University of Technology, Hawthorn, Vic 3122 Australia
Email: dr9r3y@gmail.com

Nargiza Bekmamedova[1]

Faculty of Information and Communication Technologies
Swinburne University of Technology, Hawthorn, Vic 3122 Australia
Email: nbekmamedova@groupwise.swin.edu.au

Software systems have become highly distributed and complex involving independent components working together towards achieving systems' goals. Meanwhile, security attacks against such systems have increased to become more sophisticated and difficult to detect and withstand. In this paper, we argue that the collaboration of a system's constituent components is a better way to detect and withstand this new generation of security attacks including multi-phased distributed attacks and various flooding distributed denial of service attacks. In order to achieve the collaborative intrusion detection and defenses in distributed environments, the system and its constituent components should have a common mechanism to share the collected knowledge about security attacks and countermeasures. Thus, we develop and apply security ontologies that will serve as the common vocabulary that is understandable for both humans and software agents to share and analyse the received information. In particular, several security ontologies are introduced including the security attack ontology, the defence ontology, the asset-vulnerability ontology, the algorithm-standard ontology, and the security function ontology. In conclusion, we demonstrate the applicability of our approach with a case study illustrating the Mitnick attack.

Keywords: Information security, Security ontologies, Security attacks, Security defenses.
ACM Classifications: C.2.4, C.2.6, D.2.0, D.2.11, D.2.12, D.3.1, D.4.6, H.1.2.

1. INTRODUCTION

Software systems have become highly distributed and increasingly complicated, applying various components that collaborate with each other in order to achieve system objectives. Meanwhile, new security attacks including multi-phased distributed attacks and flooding distributed denial of service attacks have appeared on the scene that are difficult to detect and mitigate. However, a collaboration of a system's constituent components that dynamically organises and adapts their composition will enable identification to resist these attacks. For such purpose, these components (represented by humans or software agents) need a common vocabulary to exchange security related information for proper and effective communications.

[1] Corresponding author

Manuscript received: 13 June 2008
Communicating Editors: Aileen Cater-Steel and Mark Toleman

In recent years the development and evolution of ontologies, i.e. explicit formal specifications of the basic categories and relations among them, have been moving from the realm of Artificial-Intelligence laboratories to the desktops of domain experts (Noy and McGuinness, 2007). In particular, the WWW Consortium (W3C) developed the Resource Description Framework (RDF) (Brickley and Guha, 1999), a language for encoding knowledge on Web pages to make them understandable to electronic agents searching for information, while the Defence Advanced Research Projects Agency (DARPA) along with the W3C created DARPA Agent Markup Language (DAML) and its successor Web Ontology Language (OWL) by extending RDF with more expressive constructs to facilitate agent interaction on the Web (Hendler and McGuinness, 2000). Nowadays, many disciplines develop standardized ontologies that enable the domain experts to share and annotate information in their fields. For example, large standardized structured vocabularies such as SNOMED (Price and Spackman, 2000) are likely to be seen in medicine and affiliated areas. Ontology (with Greek etymology that connotes a 'theory of being') is the study of the nature of being, existence and/or reality in general and of its basic categories and their relations, with particular emphasis on defining what entities exist and how these are grouped and related with 'an ontology' (i.e. a hierarchy subdivided according to similarities and differences). In the computer science discipline, ontology is often considered to be a common vocabulary for field experts who need to share information in the domain that encompasses machine-interpretable definitions of basic concepts within the domain and (inter-)relations among them.

While there are several similar definitions of ontologies (Noy and McGuinness, 2007), for this current study we follow the definition provided by Uschold *et al* (1998) who specify *ontology* as '… a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms'.

Given the above, many research works are based on the ontological approach and applied to different knowledge domains due to its numerous advantages over taxonomies and other classification schemes. Accordingly, we identify the following features of ontologies: 1) ability to achieve the shared understanding of the structured information that can be reasoned and analysed automatically among both humans and software agents; 2) ability to specify various semantic relationships among different concepts; 3) ability to solve interoperability problems; and 4) reusability and ability to evolve over time.

In this paper, we present prior developed security ontologies which specify information security issues and cover basic security concepts such as attacks, defenses, functions, vulnerabilities, etc. In particular, the main security ontology known as the security asset-vulnerability ontology (SAVO) illustrates how vulnerabilities are exploited by intruders in order to perform attacks against hosts (or systems) that may affect their assets safeguarded by defensive components. More specifically, SAVO links high-level concepts such as security policies and security goals with low-level technical countermeasures in order to separate security requirements ("What") from their implementations ("How") matched on the required actions ("Do"). Also SAVO binds other security concepts, mechanisms and ontologies including the security attack ontology (SAO), the security defence ontology (SDO), the security algorithm-standard ontology (SASO), and the security function ontology (SFO) for defining information security issues and assisting developers to create better and more efficient protection against system attacks. In particular, SASO links a system's security objectives and high-level security policies with low-level technical solutions, and specifies security algorithms, standards, concepts, credentials, assurance levels, and security objectives as ontological classes employed by SFO. SAO is utilised as the common vocabulary by a coalition of various defensive components (e.g. intrusion detection components) which interact with each other and share a common understanding of

information about attacks and defenses to ensure a better protection. Further, SAO closely correlates with SDO, which is mainly used as a specification of defensive mechanisms to resist certain security attacks and define dependences between the security algorithms and standards expressed in SASO and SFO. However, it is worthwhile to note that such division of ontologies has been done for simplification. Each class of SAO has a property which relates to a certain class of SDO. So that using a certain property, it is possible to specify the rules of anti-correlation (i.e. ways of selecting the proper countermeasure against the occurred security attack). Moreover, every class of the two ontologies has a property which describes the definitions of security attacks and defences. In case of SAO, this property may also include the rules of correlation that allow identifying security attacks which are possibly related to each other. All the described security ontologies will be discussed and reasoned further in the paper, while the applicability of the ontological approach will be exemplified with a case study simulating a Mitnick attack. In conclusion, we outline directions for future research and provide practical implications of such ontologies in information security settings.

2. BACKGROUND AND MOTIVATION

The number of distributed denial service attacks is growing day by day, while the prevention manners against those attacks are only underway. Many industries using distributed computer systems suffer from those attacks as they are real danger to the life of their business. As an inspiration, we take a game industry as the explicit example of the industry suffering from those attacks and losing the clients' private information and credit card credentials which is a significant harm to the reputation of the company's market position in the online game market where the competition is very fierce.

The computer game industry has grown rather fast over the past two or three decades, especially in such market fields as online games for desktops, game consoles and mobile phones. Currently, the Chinese online game market is one of the biggest with revenues at US$298 million in 2004, US$467 million in 2005 and US$1.3-2 billion in 2009 (Nystedt, 2005). This does not even include the volume of sales of virtual gear for the game characters, such as clothing for example, which is also a fast growing business in China. The mobile game market has become a 'secret weapon' for many companies in China. Faced with huge potential, industry key players try to take advantage of the expected growth. However, due to complexity of online gaming systems, piracy and other software security-related threats such as attacks on game servers and game players, game companies were forced to start developing strategies and technologies to prevent security attacks and protect their clients. Thereby, they aim to build trust-based relations between players and game providers while deriving a healthy profit. Further, since small game companies have limited budgets and cannot fairly compete with large game providers, they have to develop strategies to implement and maintain secure and robust gaming systems with a limited number of available resources.

2.1. Mitnick Atacks aAgainst a Gaming System

In this section, we present the modified Mitnick attack known as the WS Mitnick attack and explain it using an example.

A software system's constituent components may be highly distributed and vulnerable to various security attacks, especially distributed attacks such as Mitnick attacks (Undercoffer *et al*, 2004) which are subclasses of multi-phased distributed attacks related to the man-in-the-middle attack.

Imagine that there is a component-based gaming system as illustrated in Figure 1 that consists of game servers (hosts or peers) hosting software components that need to be communicated securely with each other over the network in order to allow online gamers to play games and also store information about them such as their credit card details and home addresses. This information

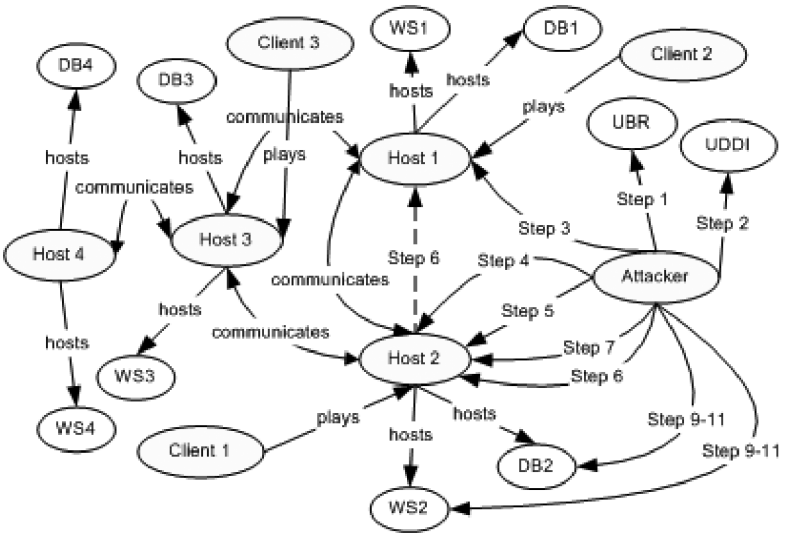

Figure 1: Mitnick attack against a gaming system

is required because a game provider needs to be sure that the game players are real people who satisfy certain criteria (e.g. age).

In this case, game servers are represented by Host 1 (**H1**), Host 2 (**H2**), Host 3 (**H3**), and Host 4 (**H4**), while game players are **Client 1**, **Client 2**, and **Client 3**. Initially, **H1**, **H2**, **H3**, and **H4** have the same configuration by default, however they can be further customized to support users from different regions of the world. So that **H1** and **H2** deliver services to game players from the USA, while **H3** and **H4** support users from Australia. And, even if the hosts are responsible for different geographical regions, they still need to synchronize game data for delivering reliability. Moreover, **H1**, **H2**, **H3**, and **H4** host Web services (**WS1**, **WS2**, **WS3**, **WS4**) to allow game players to play games through the use of Web service interfaces and databases such as **DB1** (hosted by **H1**) and **DB2** (hosted by **H2**) that store information about game players from the USA, while **DB3** (hosted by **H3**) and **DB4** (hosted by **H4**) contain data about users from Australia. Also, there is always a manager who controls the whole system's functionality.

Further, an attacker (**A)** intends to get credit card details of all game players from the USA (**Client 1** and **Client 2**). After registering as a game player from the USA by using stolen credentials and after analysing the network traffic, **A** reveals that hosts **H1** and **H2** are responsible for this region. One of the ways for **A** to get credit card details is to perform the Mitnick attack against both **H1** and **H2**. Initially, the Mitnick attack takes place sequentially in a number of performed activities, if a TCP SYN attack, a TCP sequence number prediction attack, an IP spoofing attack, and a remote access attack follow one after another. Besides, the classical Mitnick attack can be performed as the first stage of game server penetration. Then, the next stage is to perform the XML injection attack against **WS1** that occurs when user input is passed to the XML stream without proper data verification. Also, **A** does not need to scan the Web in order to find Web service targets. **A** just goes to UDDI Business Registry and gets all the required information to perform an attack against Web services such as discovering weaknesses in WSDL documents. As the Mitnick attack is related to the man-in-the-middle attack which exploits weakness of the design of a TCP protocol

in making a TCP connection (i.e. three-way handshake) between hosts, while in this case, the WS Mitnick attack consists of several stages. First, **A** tries to attack **H2** that trusts **H1** using the TCP SYN attack that initiates many partial TCP connections, and second, it penetrates the application layer using attacks against **WS1** or **DB1** including XML injection and WS probing attacks. The described attack tree is illustrated in Figure 1 and is structured as follows:

1–2. **A** navigates to UBR and requests for a website, attached to UDDI and requests WSDL files;
3. To block connections between **H1** and **H2**, **A** pretends to be **H2** (by spoofing **H2**'s IP address) and starts the TCP SYN attack against H1 by opening many partial TCP connections;
4. **A** sends multiple TCP packets to **H2** for predicting a TCP sequence number generated by **H2**;
5. **A** pretends to be **H1** and sends **H2** a SYN packet trying to establish a TCP session between **H1** and **H2**;
6. **H2** replies to **H1** by sending a SYN/ACK packet which has not seen by **A**, however **H1** cannot respond because of many partially opened connections caused by the TCP SYN attack;
7. While pretending to be **H1**, **A** sends a SYN/ACK packet to **H2** with a predicted TCP sequence number (from Step 4) and **H1**'s IP address;
8-10. Based on the premise that **H2** thinks that a TCP session is established with a trusted **H1** now, **A** performs an attack against **WS2** hosted by **H2**, by inspecting **WS2**'s WSDL files and testing vulnerable methods with XML Injection attack; if successful, **A** will apply the XML injection to change **A**'s ID and get privileges;
11. If the XML injection attack is not successful, **A** may try an SQL injection attack against **DB2** or another WS attack, as **H2** still believes that it is connected to **H1**.

Finally, the attacker can get remote access and try to execute some commands. For example, if **H1** and **H2** do not cooperate and do not constantly exchange data about attacks, then at the initial stage of the attack, **H1** registers just a short TCP SYN attack, while **H2** identifies only attempts to predict a TCP sequence number. Also, **H2** does not know that it has been penetrated through the application layer via XML injection and WS probing attacks. So, several questions have been raised: 1) how can such multi-phased distributed attacks be detected? 2) how should components collaborate with each other in order to identify, resist and mitigate such attacks? 3) how can countermeasures be devised? 4) how information about attacks and defenses can be shared and distributed among components? In the next section, we attempt to answer these questions.

3. SECURITY ONTOLOGIES

3.1 The Security Asset-Vulnerability Ontology

In this section, we briefly introduce several security terms used in our ontologies and explain their relations, as illustrated in Figure 2. However, due to the limit of space a more complete description of the security concepts can be found in the work by Stewart *et al* (2005).

The Security Asset-Vulnerability Ontology (SAVO) is built on top of other security ontologies including the security attack ontology (SAO), the security defence ontology (SDO), the security algorithm-standard ontology (SASO), and the security function ontology (SFO). SAVO is a high level security ontology that depicts information security in a simplified manner especially for non-security professionals. The design of SAVO is based upon our expertise of the domain of information security and the collected knowledge derived from various sources (e.g. Denker *et al*, 2004;

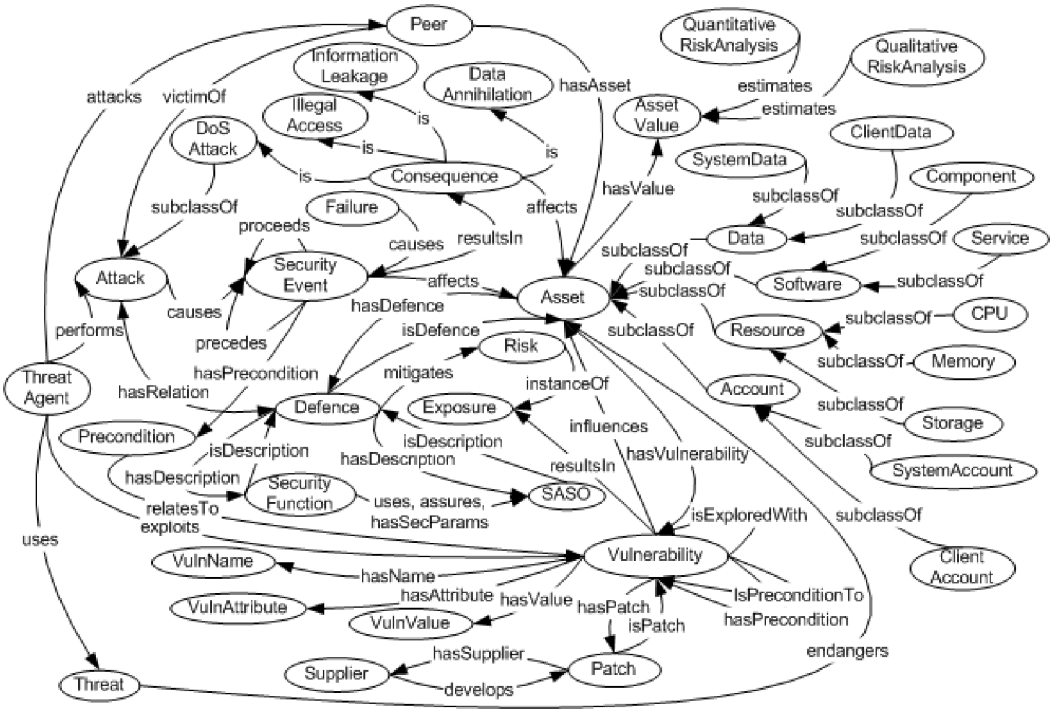

Figure 2: Security Asset-Vulnerability Ontology

Kim *et al*, 2005; Martimiano and Moreira, 2006; Seacord and Householder, 2005). In general, the described ontologies such as SAO, SDO, and SFO are represented by classes '*Attack*', '*Defence*', and '*SecurityFunction*', while the interdependencies between those ontologies will be discussed further in Section 3.4.

First, we introduce a number of basic security concepts required for a better understanding of the provided information. To begin with, the term asset ('*Asset*') connotes anything within an environment that should be protected such as data ('*Data*'), software ('*Software*'), accounts ('*Account*'), and resources ('*Resource*'). The class '*Asset*' consists of several subclasses including '*ClientData*' and '*SystemData*' that refer to client and system data; '*Component*' and '*Service*' that are related to the software implementation; '*CPU*', '*Memory*' and '*Storage*' that present resources; '*Client-Account*' and '*SystemAccount*' that are the subclasses of the class '*Account*'. The term threat ('*Threat*') is any occurrence which may cause any unwanted outcome for a company. A threat agent ('*ThreatAgent*') is an agent that can use a threat in order to exploit vulnerability, while vulnerability ('*Vulnerability*') in turn is the absence or the weakness of defence (error, flaw, etc). Further, risk ('*Risk*') is the possibility that a threat agent will exploit vulnerability to damage an asset. Exposure ('*Exposure*') reveals the possibility that '*Vulnerability*' will be exploited by '*ThreatAgent*'. Also '*Vulnerability*' may result in '*Exposure*'. Further, a single instance of '*Exposure*' is a risk ('*Risk*') which can be mitigated by using a safeguard ('*Defence*') utilised to resist attacks ('*Attack*'). Also '*Defence*' applies security techniques and mechanisms (the classes '*SecurityFunction*' and '*SecurityAlgorithmStandard*' represented by SFO and SASO accordingly). Again the more detailed report on the classes '*Attack*' and '*Defence*' can be found further in Section 3.5.

A threat agent may use a threat to perform '*Attack*' and endanger an asset. A threat agent exploits vulnerability for attacking a host or peer ('*Peer*') which hosts assets (a peer is a victim of an attack). A security attack, which has a precondition ('*Precondition*') related to vulnerability, will cause a security event ('*SecurityEvent*'). A security event can be triggered further by a system failure ('*Failure*'). Besides, a security event may affect an asset and result in a consequence ('*Consequence*') which affects an asset too. In turn, a consequence may be a destruction of data ('*DataAnnihilation*'), an information leakage ('*InformationLeakage*'), an illegal access ('*IllegalAccess*') or a DoS attack ('*DoSAttack*'), as depicted in Figure 2.

Further, an asset has a value ('*AssetValue*') which can be estimated through exploiting both the quantitative risk analysis ('*QuantitativeRiskAnalysis*') and the qualitative risk analysis ('*QualitativeRiskAnalysis*'). The asset might contain multiple vulnerabilities which can be explored with other vulnerabilities. The class '*Vulnerability*' has properties including a vulnerability name ('*VulnName*'), an attribute ('*VulnAttribute*') and associated values ('*VulnValue*'). A patch ('*Patch*') removes vulnerability and is developed usually by a supplier ('*Supplier*').

To exemplify the specific techniques that support SAVO, we introduce the quantitative and qualitative risk analysis formulas. In particular, the quantitative risk analysis provides methods to estimate a concrete probability percentage. There are several functions associated with the quantitative risk analysis (ISACA, 2008):

- The exposure factor (**EF**) shows the percentage of loss that a company would experience if certain assets are violated by the realized risk. The exposure factor is calculated in percentage value (%);
- The single loss expectancy (**SLE**) is the cost of loss and is calculated in dollar value ($) via the formula:

  SLE [$] =AV {Asset Value} * EF

- The annual rate of occurrence (ARO) is the expected frequency whereby a certain risk or threat occurs in one year. ARO is calculated using the formula below:

  ARO= Q_0 {Occurrence Frequency}/Y{year}

- The annual loss expectancy (**ALE**) is the possible cost per year of all instances of the realised threat against a certain asset. ALE is calculated using the following formula:

  ALE = SLE * ARO = AV * EF * ARO

However, the pure quantitative risk analysis cannot be performed at all reported cases. Thus, there is a qualitative risk analysis which is also used to rank threats. The process of qualitative risk analysis includes such assessment factors as experience, judgment, intuition, and so on.

To conclude, the described security asset-vulnerability ontology can also be easily modified and extended via adding supplementary subclasses to the core classes such as classes '*Asset*' or '*Vulnerability*'.

3.2 The Security Algorithm-Standard Ontology

This section of the paper aims to introduce the Security Algorithm-Standard Ontology (SASO) that encompasses security algorithms, standards, concepts, credentials, objectives, assurance levels, etc. In addition, the classes of this ontology usually serve as 'building blocks' for other ontologies such as the Security Function Ontology (SFO).

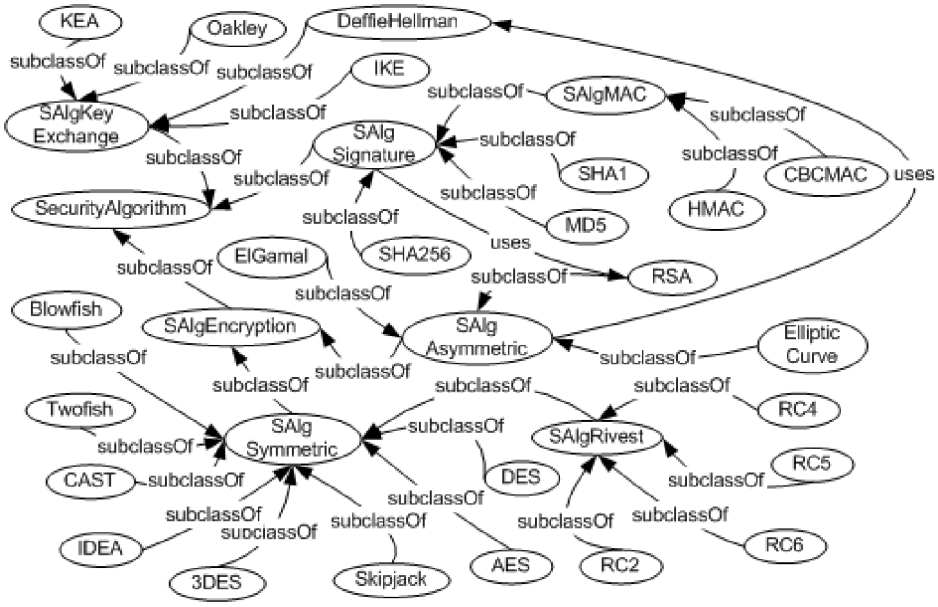

Figure 3: The class '*SecurityAlgorithm*'

SASO consists of several classes[2] including '*SecurityAlgorithm*', '*SecurityConcept*', '*SecurityAssurance*', '*SecurityCredential*', and '*SecurityObjective*'. The class '*SecurityAlgorithm*' is used to define various security algorithms. The structure of this class is illustrated in Figure 3, where the main class called '*SecurityAlgorithm*' has three major subclasses including '*SAlgEncryption*', '*SAlgSignature*', and '*SAlgKeyExchange*'. The class '*SAlgEncryption*' defines classes that deliver encryption/decryption capabilities and consists of two subclasses (the classes '*SAlgSymmetric*' and '*SAlgAsymmetric*') that represent symmetric and asymmetric cryptographic algorithms. The class '*SAlgKeyExchange*' specifies cryptographic key exchange algorithms, while the class '*SAlgSignature*' includes classes responsible for digital signatures and cryptographic hash functions.

The class '*SecurityConcept*' consists of three subclasses that include subclasses called '*SConMechanism*', '*SConProtocol*' and '*SConSecurityPolicy*'. Herein, the similar structure is inherited as provided in the class '*SecurityConcept*' of the main security ontology taken from Kim *et al* (2005). However, there is a difference between security protocols and security mechanisms, where the former class (the class '*SConProtocol*') defines how to fulfil certain tasks using certain steps, while the latter class (the class '*SConMechanism*') is the only implementation of security protocols. The class '*SConMechanism*' mainly specifies various types of firewalls, proxies and virtual machines (VMs), whereas the class '*SConProtocol*' has many subclasses that define different security protocols used in authentication (the class '*SConAuthentication*'), encryption (the class '*SConEncryption*'), key management (the class '*SConKeyMngmt*'), digital signatures (the class '*SConSignature*'), secure e-mail systems (the class '*SConEmail*'), and secure communications (the class '*SConSecureCommunication*'). The class '*SConSecurityPolicy*' describes the rules and

[2] We would like to draw the reader's attention to the fact that this section discusses only the main ontology classes as we see no point in presenting a lengthy report on the functionality of each subclass of the main classes of the ontology.

policies used in access control models. The class '*SecurityAssurance*' specifies assurance methods for security algorithms, protocols, and mechanisms. Currently it has only one subclass called '*CommonCriteria*' (CC) (CC IS, 2006), which is the international standard for the computer security evaluation and is used to evaluate security measures applied to software applications. However, new classes such as '*TCSEC*' (Trusted Computer System Evaluation Criteria) for example, can be easily added into the ontology. The class '*SecurityCredential*' is commonly used for identification and authentication purposes. Finally, the class '*SecurityObjective*' is used to define security objectives including confidentiality, authorisation, integrity, availability, non-repudiation, authenticity, auditability, identification, accountability, and survivability.

3.3 The Security Function Ontology

In this section, we introduce the Security Function Ontology (SFO) which is combined with the security algorithm-standard ontology (SASO). There are six main classes of security functions (the class 'SecurityFunction') extracted from Khan (2005) and derived from merging 11 Common Criteria classes including '*PrivacyFunc*', '*CryptoSupportFunc*', '*IdentificationAuthorisationFunc*', '*UserDataProtectionFunc*', '*TrustedChannelFunc*', and '*SecurityAuditResourceUtilisationFunc*'. In addition, we develop two additional classes, '*TimeFunc*' and '*ProbabilityFunc*' that are used together with security functions to specify the dynamic and unpredictable nature of security in distributed software systems, as illustrated in Figure 4.

Each of these classes has subclasses with the property '*hasSecurityObjective*' which in turn, are mapped onto the security functions with parameters from the classes '*SecurityAlgorithm*', '*SecurityConcept*', and '*SecurityObjective*'. In detail, the class of security functions for cryptographic support ('*CryptoSupportFunc*') is designed to support encryption/decryption algorithms. It consists of two families, cryptographic key management which is used to address the management aspects of cryptographic keys, and cryptographic operations which relates to the operational use of

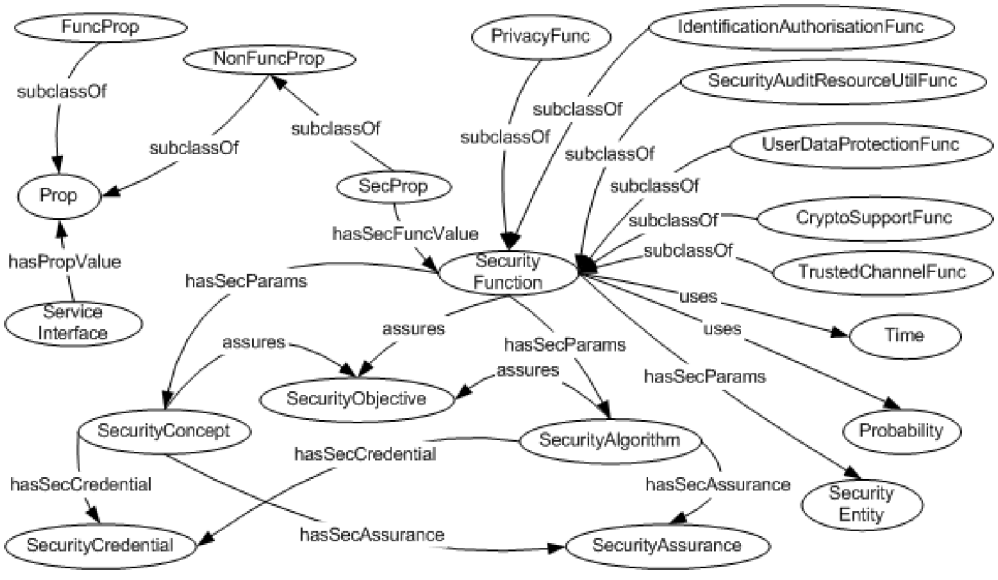

Figure 4: Interrelations among various security concepts

cryptographic keys. Security functions or their sets assure certain security goals such as confidentiality, integrity, availability, etc. The class of security functions for user data protection (‘*UserDataProtectionFunc*’) is utilised to protect user data during its import, export, and storage. The class of security functions for identification and authorisation (‘*IdentificationAuthorisationFunc*’) is used to address requirements for security functions to establish and check identities claimed by users. The class of security functions for security audit and resource utilisation (‘*SecurityAudit-ResourceUtilFunc*’) is regarded more as a security service. Usually, it is used for detecting, analysing, and storing the security-related activities. The resulting information can be used for the audit and identification of the type of security activities. The class of security functions for privacy (‘*PrivacyFunc*’) specifies the privacy requirements of users and it is associated with the security functions for trusted channels (‘*TrustedChannelFunc*’) that aim to protect communications between users or other entities from modification or disclosure. The classes ‘*TimeFunc*’ and ‘*ProbabilityFunc*’ are utilised as parameters for the class ‘SecurityFunction’. They enable describing of the dynamic and unpredictable nature of distributed systems. Besides, these classes are widely used to specify security properties which are time- and event-dependent with a high probability.

To conclude, there are about forty-four security functions; however only a few of them have been presented here for the purpose of the current paper.

3.4 Interrelationships Between Concept Classes Across Security Ontologies

In this section, we describe interrelationships between different security classes of SASO and SFO, as illustrated in Figure 4.

As shown above, the main class of SFO is the class ‘*SecurityFunction*’ which consists of six subclasses introduced in the previous sections. All these subclasses incorporate security functions with the property named as ‘*hasSecParams*’ which in turn, contains the classes ‘*SecurityConcept*’ (SASO), ‘*SecurityAlgorithm*’ (SASO), and ‘*SecurityEntity*’ as values. These classes are used as values since they define security algorithms and standards as well as security entities such as cryptographic keys or operations. However, such classes as ‘*Time*’ and ‘*Probability*’ might be embedded with the class ‘*SecurityFunction*’ and its subclasses for the purpose of specification of the security properties of software systems. Since many security algorithms and concepts have various assurance levels and use security credentials for the access control, some of the SASO classes (i.e. ‘*SecurityConcept*’ and ‘*SecurityAlgorithm*’) have two properties such as ‘*hasSecCredential*’ and ‘*hasSecAssurance*’ with the appropriate classes ‘*SecurityCredential*’ and ‘*SecurityAssurance*’ as values.

3.5 The Security Attack and Defence Ontologies

In this section, we define the Security Attack Ontology (SAO) and the Security Defence Ontology (SDO) that can be used as a common vocabulary by coalitions of software agents (e.g. coalition of defensive components) and/or human end-users. Many defensive components (DCs) use SAO and SDO to interact with each other and share a common understanding of information about attacks and strategies to resist them. In particular, coalitions might be very useful for detecting distributed multi-phased attacks such as the Mitnick attack (see Undercoffer *et al*, 2004). The detailed description of these security attacks and the relevant ontologies can be found in Vorobiev *et al* (2008). Also, SAO and SDO have ability to evolve over time.

Further, as illustrated in Figure 5, if a DC from a coalition detects a new attack, it will add this attack as a new class to SAO and will further share the ontology with the other members of the coalition. When any of the coalition members develops a new countermeasure against an attack, it will afterwards be added to SDO as a new class and will be distributed among the other members.

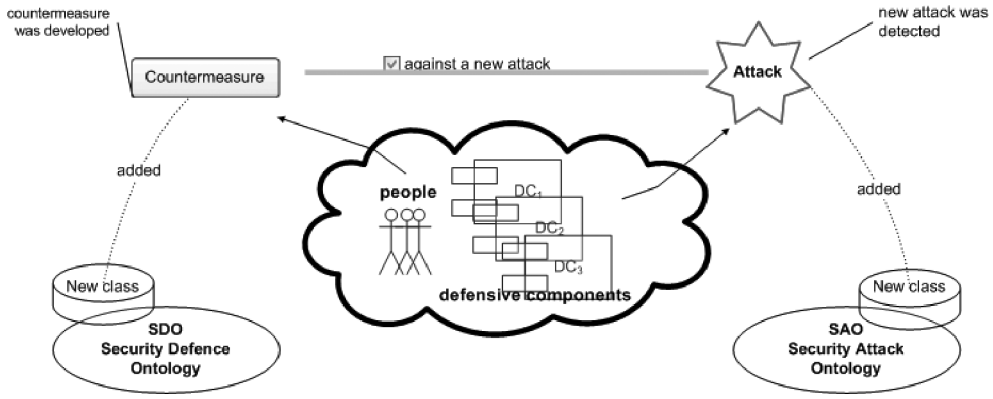

Figure 5: An illustrated example

The main SAO class called '*Attack*' has five subclasses, '*WSAttack*', '*P2PAttack*', '*DoSAttack*', '*SniffingAttack*', and '*MultiPhasedDistributedAttack*' (see Figure 6). In particular, '*MultiPhased-DistributedAttack*' employs security attacks from the first four subclasses. The class '*WSAttack*' defines attacks on Web services, while the class '*P2PAttack*' specifies security attacks against the peer-to-peer (P2P) systems. The class '*DoSAttack*' describes various denial of service (DoS) attacks. Security attacks on communication channels are characterized by the class '*SniffingAttack*' (e.g. the Mitnick attack which is the subclass of the class '*MultiPhasedDistributedAttack*'). The primary reason why we chose multi-phased distributed attacks is the fact that they are rather difficult to identify and protect systems against. In response, the distributed parties should operate as a coalition in order to detect and resist these attacks. The class '*Attack*' (and its subclasses) has a property called '*hasRelation*' which relates to the class '*Defence*' (and its subclasses) from SDO, as illustrated in Figure 6. Besides, while using this property, it is likely to define the rules of anti-correlation. The property '*hasDescription*' is used to specify the definite attacks and defences. In

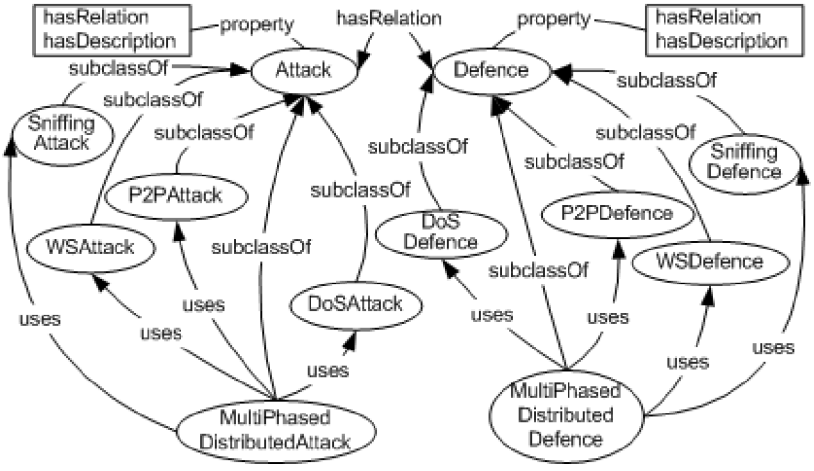

Figure 6: The illustrated classes '*Attack*' and '*Defence*'

case of the attack definitions, this property may also contain the correlation rules, i.e. the rules specifying how to detect security attacks and identify the triggering root attack.

The class hierarchy of SDO is similar to SAO. The main SDO class is called '*Defence*' and it also has subclasses that correspond to SAO classes and contain defenses against attacks on Web service ('*WSDefence*') and P2P systems ('*P2PDefence*') attacks, safeguards against DoS attacks ('*DoSDefence*'), and countermeasures against sniffing attacks ('*SniffingDefence*') and multi-phased distributed attacks ('*MultiPhasedDistributedDefence*'). The class '*Defence*' has two properties '*hasRelation*' which links to the class '*Attack*' and '*hasDescription*' which utilises the security algorithm-standard and security function ontologies to specify countermeasures.

Correspondingly, the class hierarchy of SDO is similar to SAO. The main SDO class is called '*Defence*' and it has five subclasses correlated with five SAO classes including defence against attacks on Web services (the class '*WSDefence*') and P2P systems (the class '*P2PDefence*'), safeguards against DoS attacks (the class '*DoSDefence*'), countermeasures against sniffing attacks (the class '*SniffingDefence*') and multi-phased distributed attacks (the class '*MultiPhased-DistributedDefence*'). The class '*Attack*' has the following two properties called as '*hasRelation*' and '*hasDescription*'. However, the full description[3] of SAO and SDO ontologies is out the scope of this paper.

3.6 Useful Implications of Security Ontologies

Given that ontologies are the representation of the domain knowledge, they explicitly formalise and specify the concepts and their corresponding relationships. Also, they include associated specific instances for the corresponding concepts. These instances contain the actual data used in knowledge-based applications (Patel *et al*, 2003). Thus in this section, we describe how the developed ontologies might be used via deriving the necessary information and how such information can be gathered and embedded via employing these ontologies. Yet, we would like to note that security experts are required (Tsoumas and Gritzalis, 2006) to fill the security ontologies with the necessary information and then to apply them (e.g. security controls) within an organisation. In addition, such management decisions might be influenced directly, for example, by various organisation policies, audit and risk analysis reports, or Service Level Agreements (SLAs). On the other hand, such decisions might be implied indirectly also via the best practices, security standards, independent sources, security mailing lists, and vulnerability catalogues.

The process of embedding the security ontologies with the required data consists of nine stages: 1) gather infrastructure data regarding the network structure, software applications, launched servers, hardware, versions, active services and ports using network scanners such as Nmap (Nmap, 2007) and other tools from one of the Linux distributives, Backtrack (versions 2 and 3); 2) select and evaluate assets using the quantitative and qualitative risk analysis, as previously described; 3) find and rank vulnerabilities using penetration tools such as from Backtrack2/3, CORE IMPACT (CORE IMPACT, 2008) or SAINT (SAINT, 2008); 4) make a trade-off between security-related information collected at the previous stages and the business requirements; 5) collect data related to possible security attacks and its countermeasures; 6) extract information from security policies, evaluate them using the tacit knowledge (based on personal experience and expertise of the security expert) and correct security requirements (if it is required); 7) align security requirements with appropriate security controls via the introduced security functions; 8) embed the collected data into security ontologies; 9) iterate from the first stage to the last stage on a timely basis.

[3] Full descriptions of both the security attack and defense ontology are available upon request via contacting the corresponding author.

4. CASE STUDY: THE WS MITNICK ATTACK

In this section, we explain how combinations of methods described above should be applied to identify and mitigate multi-phased distributed attacks. We demonstrate the applicability of the ontological approach via a gaming system and the initiated Mitnick attack against it. The gaming system's constituent components, described in the previous sections, collaborate, identify distributed attacks, and securely share information about them and then develop countermeasures using the security ontologies. The more detailed description of the WS Mitnick attack is presented below.

4.1. Scenario and Analysis

A game player (**Client 1**) from the USA for the first time wants to play a game called Heroes of Might and Magic 6 (HMM6). The gamer downloads a game client application from a website *www.hmm6.com*. After installing online game software and specifying all requirements, the player starts playing. Besides playing online games, game players can do other activities such as chatting or sharing files. As illustrated in Figure 1, **Client 1** connects to **H2** and plays HMM6. Meantime, **H2** actively communicates with **H1** by synchronising data. However, **A** tries to perform the Mitnick attack in conjunction with WS attacks against **H1** and **H2**. As previously mentioned, Mitnick attacks can be detected by several members of a coalition who are distributed over the network and who cooperate with each other, and therefore **H1** and **H2** cooperate and constantly exchange data about security attacks and defenses.

Now assume that **A** attacks **H1** (whose IP address is *147.202.46.43*) and **H2** (whose IP address is *147.202.46.40*) from a host with IP address *192.2.1.23*. However, all stages of the attack have been explained in detail in the previous sections of the paper. Here, we only describe how **H1** and **H2** respond to the attack. First, **A** spoofs **H2**'s IP address and initiates TCP connections with **H1**. Second, **H1** detects that many partial TCP connections are opened from **H2**'s IP address *147.202.46.40* at *14.40.00* on *08.08.2008*. Third, **H2** detects that someone tries to predict a sequence number from the IP address *192.2.1.23* and spoofs **H1**'s IP address in order to perform a Mitnick attack. To prevent this, **H1** sends a message (the class '*SecurityEvent*' from SAVO) to **H2** to verify if **H2** really has opened all these connections, and if there is no response from **H2**, then **H1** and **H2** take prompt actions, for example sending an alert message regarding a possible attack to other hosts. Properties '*Name*', '*Time*', '*Target*', and '*Source*' specify a name of a security attack, when a security event has occurred, which IP address has been requested, and from which IP address the event has been initiated.

An instance of the class '*SecurityEvent*' called '*SecEv1*' is sent to **H2** and, if **H2** does not reply in time (e.g. three minutes later), then **H1** generates an alert. Simultaneously, **H2** detects that someone from the IP address *190.2.1.23* tries to predict a TCP sequence number. Since **H2** receives '*SecEv1*' and replies to **H1** but in turn, **H2** does not get any acknowledgement from **H1**, the manager concludes that a Mitnick attack is occurring and sends security events containing this information to other hosts. In fact, there are many rules that specify attacks (similar to expert-based systems) stored in the Ontology Repository. In this case, the manager concludes that there is the Mitnick attack when **H1** detects many partial connections from **H2**, while **H2** registers many TCP sequence number prediction and IP spoofing attempts following one after another. Also, **H2** identifies XML injection and WS probing attacks minutes later from **A**'s IP address. Further, the manager concludes that all these attacks are related to each other and performed by one attacker. Consequently, the manager labels the combination of these attacks as the WS Mitnick attack, specifies the rule of this attack and encrypts (e.g. using the AES algorithm) and sends this information to other hosts including **H1**, **H2**, **H3**, and **H4**.

It is very important to treat these attacks as a part of coordinated multi-phased distributed attacks and not as independent attacks, i.e. they should be analysed (e.g. by the manager) as part of one complex attack in order to see, control, and maintain the entire picture of attacks. Such an approach helps to identify complex attacks and develop more comprehensive countermeasures faster.

After the attack has been identified, a group of the game company's software developers and system administrators start developing countermeasures and after finding them, they securely distribute the rules of anti-correlation for this specific attack among other hosts which belong to the gaming system including **H1**, **H2**, **H3**, and **H4**. However, we do not explain here how countermeasures are developed and deployed. If a WS Mitnick attack is detected, then the manager forces the system to close all connections from the spoofed IP address (*147.202.46.40*), then increases the size of the connection queue, decreases the time-out waiting for the three-way handshake, and blocks all connections and ports for **A**'s IP address (*192.2.1.23*). A similar rule for describing countermeasures against TCP SYN attacks has been introduced in the previous section. However, it is worthwhile to mention that Mitnick attacks should be treated as a baseline to verify if the system can really detect coordinated multi-phased distributed attacks.

5. RELATED WORK

Generally, ontologies are mainly utilised to express semantic information and they can be applied to the Web service context. Currently, there are several approaches that allow Web services to publish their semantic data. Among those are OWL (OWL, 2007), OWL-S (OWL-S, 2007), Semantic Web Services Language (SWSL) (SWSL, 2007), etc. However, none of them is specifically designed to express the security requirements and capabilities of Web services. Yet, there is limited research in the area of security ontologies including the NRL security ontology (Kim *et al*, 2005) which is based upon (Denker *et al*, 2004) and expresses types of security information such as security algorithms, mechanisms, protocols for all types of resources. Further, research related to our work (Tsoumas and Gritzalis, 2006) deals with an ontology-based security management. This work proposes a similar approach including countermeasures, however it does not provide security attack and defense ontologies. Another approach called Secure Tropos (Mouratidis and Giorgini, 2007) is a framework with formal semantics in Datalog that allows modelling relationships among goals and actors through the use of permissions, delegations, refinements, and ownerships. It also models security using security ontologies. However, our approach proposes security ontologies that are more comprehensive and rich in description and are specified using a formal language called GIZKA that supports logic programming. Security ontologies such as SAVO and SASO are partially based on Damiani *et al* (2002); Denker *et al* (2004); Martimiano and Moreira (2006); Undercoffer *et al* (2004), while SFO follows the ideas extracted from some efforts (Khan and Han, 2003; Khan, 2005; Vorobiev and Khan, 2006a). Further, SAO and SDO reflect ideas from a number of authors (Denker *et al*, 2004; Kim *et al*, 2005; Lindstrom, 2004; Martimiano and Moreira, 2006; Stewart *et al*, 2005; Undercoffer *et al*, 2004). Previously, we developed the security attack ontology for Web services only (see Vorobiev and Khan, 2006b), while in this paper, we have added four additional classes to SAO to specify attacks against P2P systems, to define DoS attacks, to characterise security attacks against communication channels, and describe multiphased distributed attacks.

In all, our ontologies cover many aspects of information security related to security attacks and defences against them including DoS attacks against Web services and P2P systems and its countermeasures.

6. CONCLUSION AND FUTURE WORK

Software systems have become highly complex and distributed because users demand and expect a high level of functionality. This trend is driven by new technologies which use the increasing power and falling cost of network resources and new hardware. Also, software systems have many vulnerabilities and design flaws due to the fact that system security is usually implemented after its functional requirements have been addressed, and consequently, companies spend much time and money to fix these vulnerabilities. Meanwhile, a new generation of security attacks have appeared such as various multi-phased distributed attacks that currently are difficult to identify, analyse, correlate and anti-correlate. Hence, there is a strong need indeed for a systematic approach to designing and implementing secure and robust distributed systems. In our case, these systems consist of different constituent components that collaboratively identify security attacks and deploy countermeasures.

In this paper, we have demonstrated that through collaboration of a system's constituent components it is possible to detect, correlate, anti-correlate, and withstand such security attacks. However, components should have a common basis (vocabulary) that allows them to exchange information with each other about security threats. Therefore, we have applied an ontological approach to specify information security issues in a way understandable to both humans and software agents. Moreover, we have introduced various security terms, concepts and mechanisms through specified security ontologies including the security asset-vulnerability ontology (SAVO), the security algorithm-standard ontology (SASO), and the security function ontology (SFO), the security attack ontology (SAO), and the security defence ontology (SDO). The applicability of the developed security ontologies specified in OWL has been further illustrated with the Mitnick attack.

In future work, we are planning to develop trust ontologies and investigate how different levels of trust might impact on the security ontologies and consequently, information security settings.

REFERENCES

BRICKLEY, D. and GUHA, R.V. (1999): RDF vocabulary description language 1.0: RDF Schema. *Proposed W3c Recommendation for World Wide Web Consortium*. http://www.w3.org/TR/PR-rdf-schema. Accessed 21-Nov-2001.

CC IS (2006): Common criteria international standard. http://www.niap-ccevs.org/cc-scheme. Accessed 3– Feb-2008.

CORE IMPACT (2008): CORE IMPACT website. http://www.coresecurity.com. Accessed 15-Sept-2008.

DAMIANI, E., De CAPITANI Di VIMERCATI, S., PARABOSCHI, S., SAMARATI, P. and VIOLANTE, F. (2002): A reputation-based approach for choosing reliable resources in peer-to-peer networks. CCS, Washington, USA.

DENKER, G., NGUYEN, S. and TON, A. (2004): OWL-S semantics of security web services: A case study. SRI International, Menlo Park, California, USA.

HENDLER, J. and McGUINNESS, D. (2000): The DARPA agent markup language. *IEEE Intelligent Systems* 16 (6): 67–73.

ISACA (2008): Information systems audit and control association. http://www.isaca.org. Accessed 25-Aug-2008.

KHAN, K. and HAN, J. (2003): A security characterisation framework for trustworthy component based software systems. COMPSAC2003, 164–169.

KHAN, K. (2005): Security characterisation and compositional analysis for component-based software systems. Ph.D. thesis. Monash University at Melbourne.

KIM, A., LUO, J. and KANG, M. (2005): Security ontology for annotating resources. The *4th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE 2005)*.

LINDSTROM, P. (2004): Attacking and defending web services. A Spire Research Report.

MARTIMIANO, L.A. and MOREIRA, E. (2006): The evaluation process of a computer security incident ontology. *The 2nd Workshop on Ontologies and Their Applications*, Ribeirão Preto.

MOURATIDIS, H. and GIORGINI, P. (2007): Secure tropos: A security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific 17(2): 285–309.

Nmap SCANNER (2007): Nmap scanner website. http://www.insecure.org/nmap. Accessed 10-Sept-2008.

NOY, N.F. and MCGUINNESS, D.L. (2007): Ontology development 101: A guide to creating your first ontology. http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html. Accessed 12–Jun–2007.

NYSTEDT, D. (2005): Online gaming growing fast in China. IDG News Service.

OWL (2007): Web ontology language. http:w3.org/TR/owl-features. Accessed 17-May-2007.
OWL-S (2007): Semantic markup for web services. http://www.daml.org/services. Accessed 17-May–2007.
PATEL, C., SUPERKAR, K. and LEE, Y. (2003): OntoGenie: Extracting ontology instances from WWW, *Proc. of Workshop on Human Language Technology for the Semantic Web and Web Services 2003*, Sanibel Island, Florida, 123–126.
PRICE, C. and SPACKMAN, K. (2000): SNOMED: Clinical terms. *British Journal of Healthcare Computing and Information Management* 17(3): 27–31.
SAINT (2008): SAINT website. http://www.saintcorporation.com. Accessed 20-Sept-2008.
SEACORD, R.C. and HOUSEHOLDER, A.D. (2005): A structured approach to classifying security vulnerabilities. CMU/SEI-2005- TN-003.
SWSL (2007): Semantic web services language. http://www.daml.org/services/swsl. Accessed 17–May- 2007.
STEWART, J.M., TITTEL, E. and CHAPPLE, M. (2005): CISSP: Certified information systems security professional: Study Guide. Sybex.
TSOUMAS, B. and GRITZALIS, D. (2006): Towards an ontology-based security management, *Proc. of AINA06*, Austria.
UNDERCOFFER, J., JOSHI, A., FININ, T. and PINKSTON, J. (2004): A target-centric ontology for intrusion detection. *International Joint Conference on Artificial Intelligence*, Mexico.
USCHOLD, M., KING, M., MORALEE, S. and ZORGIOS, Y. (1998): The enterprise ontology. *The Knowledge Engineering Review* 13(1): 31–89.
VOROBIEV, A. and HAN, J. (2006a): Specifying dynamic security properties of web service based systems. *Proc. of SKG06*, China.
VOROBIEV, A., HAN, J. (2006b): Security attack ontology for web services. *Proc. of SKG06*, China.
VOROBIEV, A., HAN, J. and BEKMAMEDOVA, N. (2008): An ontology framework for managing security attacks and defences in component based software systems. *Proc. of the Australian Software Engineering Conference (ASWEC'08)*, Perth, Australia.

BIOGRAPHICAL NOTES

Artem Vorobiev, PhD, CISSP, is a software security engineer with National ICT Australia. He has been an active promoter of his industry-based software development and security experience to be applied to academia. He received his PhD in Software Security Engineering from Swinburne University of Technology, Australia. His research interests include information security, security ontologies and software engineering research. His research interests have appeared in Proceedings of Australian Software Engineering and Information Systems Conferences and also International Conference on Information Systems.


Artem Vorobiev

Nargiza Bekmamedova is currently a PhD candidate at Swinburne University of Technology (Australia). Her current research focuses on exploring the dynamics of the relationship between trust and formal controls in the client-vendor IS outsourcing arrangements. She is also interested in information security and classifications of trust-based ontologies. Her research interests have appeared in Proceedings of Australian and International Information Systems Conferences.


Nargiza Bekmamedova