An Ontology Framework for Managing Security Attacks and Defences in Component Based Software Systems

Artem Vorobiev, Jun Han, and Nargiza Bekmamedova
Faculty of ICT, Swinburne University of Technology, Melbourne, Australia
{avorobiev, jhan, nbekmamedova}@ict.swin.edu.au

Abstract

Software systems become increasingly distributed, involving many independent and collaborating components working towards achieving system goals. At the same time, security attacks on these systems have also grown being more sophisticated and are quite difficult to identify and mitigate, in particular including distributed attacks. In this paper, we argue that one way to detect and resist against such attacks is through the collaboration of a system’s constituent components. To achieve collaborative defense in a distributed component-based system, a common basis (vocabulary) is needed for the components to communicate and work with each other in detecting attacks and devising countermeasures. We adopt an ontological approach to establishing such a common base and introduce ontologies concerning security attacks and defenses. The ontologies specify the security concepts and their relationships in a way understandable to both humans and software agents. We use a case study involving Mitnick attacks to demonstrate how system components use the ontologies to detect and counter attacks.

1. Introduction

Software applications become highly distributed and increasingly complicated, implicating various components that collaborate with each other in order to achieve system objectives. Simultaneously, attackers become smarter in creating new types of attacks, especially distributed attacks, which are quite difficult to identify and mitigate. Thus, we argue that it is possible to detect and resist against such distributed attacks through the collaboration of a system’s constituent components. However, to achieve collaborative defense in distributed environments such as component-based systems, components should have a common vocabulary to allow them to communicate with each other regarding security attacks and countermeasures. For such purpose, we employ an ontological approach that allows sharing a common understanding of information about attacks and defenses among humans and software agents.

The term “ontology” is defined in [29] as follows: “An ontology may take a variety of forms, but necessarily it will include 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.” Many research works apply an ontological approach to different knowledge domains due to its advantages over taxonomies and other classification schemes. Ontologies have the following features: (1) achieving a shared understanding of structured information, which can be reasoned and analysed automatically, among both humans and software agents; (2) specifying various semantic relationships among different concepts; (3) solving interoperability problems; (4) being reused and evolving over time.

In this work, we develop security ontologies which specify information security issues, including security attacks and defenses in particular. The main security ontology called the security asset-vulnerability ontology (SAVO) illustrates how vulnerabilities are exploited by intruders in order to perform attacks against peers or systems that may affect their assets evaluated by using the quantitative and qualitative analysis and protected by defensive components. SAVO binds high-level security policies with 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 and failures. SAO is utilized as a 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 better protection. SAO closely correlates with SDO, which is mainly used as a specification of a number of defensive mechanisms to resist certain security attacks and define dependences between the security algorithms and standards expressed in SASO and SFO. We should mention that such separation of ontologies is done for simplicity. Besides, they can be treated as a core that is extended every time when new security attacks and countermeasures appear. Also, components in our approach can be viewed as servers (peers) that provide a collection of various services.

The paper is organised as follows. In Section 2, we introduce a motivating example involving Mitnick attacks [28] and identify the need to consider security attack and defence ontologies. In Section 3, we briefly describe our framework. In Section 4, we present a description of SAO, SDO, and SAVO. In Section 5, we demonstrate our approach applying it to the example. In Section 6, introduce a prototype to illustrate an applicability of our approach. In Section 7, we introduce related works. Finally, we conclude and outline directions for future research in Section 8.

2. Background and motivation

The computer game industry grows rather fast especially in the area of online games. For example, China’s online gaming market is considered the most promising one in the world with approximately estimated revenue at US$1.3-2 billion in 2009 [18]. Hence, many industry players try to take advantage of such expected growth. However, gaming systems of the future will be highly complex and distributed in order to increase delivering services to game players dispersed over the world who will wish to play online games using not only laptops or desktops but mobile phones as well in any place with access to the Web. One of the possible solutions is to implement gaming systems using the component-based software system (CBSS) and peer-to-peer (P2P) approaches [2,9]. Hosts or peers that host components or Web services (WSs) will deliver flexibility and solve complexity issues. However, many security issues will arise too (e.g. how to identify distributed attacks against Web services or their hosting peers and resist them, how to secure communications among various peers, and so on).

2.1. Mitnick attacks against a gaming system

As previously mentioned, a software system’s constituent components may be highly distributed and vulnerable to various security attacks, especially distributed attacks such as Mitnick attacks which are subclasses of multi-phased distributed attacks.

For example, there is a component-based gaming system, illustrated in Figure 1, which consists of game servers or so called peers that need to communicate securely with each other over the network in order to allow game players to play online games and also store information about them including their credit card details and home addresses. Such information is required because a game provider needs to be sure that game players are real people who satisfy certain criteria (e.g. age). In our case, game servers are represented by Peer 1 (P1), Peer 2 (P2), Peer 3 (P3), and Peer 4 (P4) while game players are Client 1, Client 2, and Client 3. Initially, P1, P2, P3, and P4 have the same configuration by default, however, they can be customised later for supporting users from different regions in the world. P1 and P2 provide services to game players from the U.S.A. while P3 and P4 support users from Australia. And, even if they are responsible for different geographical regions, they still need to synchronise game data for delivering reliability. Moreover, peers host Web services (WS1, WS2, WS3, WS4) to allow game players to play games through the use of Web service interfaces and databases such DB1 (hosted by P1) and DB2 (hosted by P2) which store information about game players from the U.S.A. and DB3 (hosted by P3) and DB4 (hosted by P4) which contain data about users from Australia.

An attacker (A) wants to get credit card details of all game players from the U.S.A. (Client 1 and Client 2). After registering as a game player from the U.S.A. using stolen credentials and social engineering skills (beyond the scope of this paper), and after analysing network traffic, A figures out that P1 and P2 are responsible for this region. One of the ways for A to get credit card details is to perform the Mitnick attack against both P1 and P2. It is essential to mention, that the classical Mitnick attack can be performed at the first stage of penetrating a game server. The next stage can be performing 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 (UBR) and gets all required information for performing an attack against Web services such discovering weakness in WSDL documents in order to get access to mission critical applications and infrastructures (WS probing attacks). Also, since the Mitnick attack is related to the man-in-the-middle attack that exploits weakness of the design of a TCP protocol in making a TCP connection.
(three-way handshake) between peers. We note that a combination of security attacks against different layers such as application, network, and communication layers is rather dangerous because of difficulties with detecting and resisting them.

To better understand, we introduce the three-way handshake between P1 and P2 which is described as follows: 1) P1 tries to organise a connection with P2 by sending a SYN packet to P2, 2) Then, P2 sends a SYN/ACK packet to P1, 3) P1 receives a SYN/ACK packet and sends a SYN/ACK packet (an acknowledgment) to P2. This is the final stage of the handshake when a connection is established.

**Figure 1. Mitnick attacks against a component-based gaming system**

The Mitnick attack can be performed along with various attacks against Web services. We call such attacks involving the Mitnick attack as the WS Mitnick attacks. In our case, the attack consists of several steps when A tries to attack P2 that trusts P1 using the TCP SYN attack that initiates many partial TCP connections, and then penetrates the application layer using attacks against WS1 or DB1 including XML injection and WS probing attacks. The attack tree is illustrated in Figure 1 and is structured as follows:

1. A navigates to UBR and requests for a website.
2. A attaches to UDDI (Universal Description, Discovery, and Integration) and requests WSDL files.
3. For blocking connections between P1 and P2, A starts the TCP SYN attack against P1 by opening many partial TCP connections.
4. A sends multiple TCP packets to P2 for predicting a TCP sequence number generated by P2.
5. A pretends to be P1 by spoofing P1’s IP address and tries to establish a TCP session between P1 and P2 by sending a SYN packet to P2.
6. P2 replies to P1 by sending a SYN/ACK packet which is not seen by A, however, P1 cannot terminate a connection by sending a RST packet because of many partially opened connections caused by the TCP SYN attack.
7. A cannot see a SYN/ACK packet from Step 4, however, A can apply a TCP sequence number from Step 2 and P1’s IP address and send a SYN/ACK packet with a predicted number in response to a SYN/ACK packet sent to P1.
8. Now, P2 thinks that a TCP session is established with a trusted P1. A can perform an attack against WS1 hosted by P2 that believes that has a one way session with P1.
9. A inspects P2’s WSDL files and tries to find vulnerable methods.
10. Then, A tests these methods for performing the XML Injection attack and, if A is successful, A applies the XML injection for changing A’s ID and getting privileges.
11. However, if the XML injection attack is not successful, then A may try the SQL injection attack against a database (DB) hosted by P2 or any other injection attacks against Web services, because P2 still believes that it is connected to P1. However, there are some issues in the described above example. For example, if P1 and P2 do not cooperate and do not constantly exchange data about attacks, then at the initial stage of the attack, P1 registers just a short TCP SYN attack while P2 identifies only attempts to predict a TCP sequence number. Also, P2 does not know that it has been penetrated through the application layer via XML injection and WS probing attacks. So, we raise several questions: 1) How such multi-phased distributed attacks can be detected? 2) How components should collaborate with each other in order to resist and mitigate such attacks? 3) How countermeasures can be devised? 4) How information about attacks and defenses can be shared and distributed among components?

3. Overview of a framework

As referred above, Mitnick attacks can be detected by using a coalition of peers or distributed components, defensive components (DCs) in particular. However, the problem in the component context is that they can be distributed over the Web, controlled by different parties, and developed by different vendors. To identify such multi-phased and distributed attacks, various components from different vendors should have a common vocabulary of attacks and defences in order to communicate with each other. This process is illustrated in Figure 2.
To support our framework, we have developed the Architecture Description Language (ADL) GIZKA and the reference architecture called SECROBAT [32] for creating secure and robust CBSSs, which operate as a coalition and are highly adaptive and reconfigurable in order to resist various types of attacks and failures. SECROBAT has several key features including DCs such as intrusion detection components (IDCs), honeypot components (HCs), key distribution components (KDCs), and the adaptive and reconfigurable architecture with the hybrid P2P [2,9] structure. DCs are integrated parts of such coalitions and can be reconfigured, added or deleted dynamically to protect the system from intrusions, study attacks, and secure connections among normal components. Moreover, coalitions adapt their hybrid P2P structure dynamically when there is an attack or the environment changes. For example, during distributed denial of service (DDoS) attacks a coalition instead of only blocking hosts also changes its structure dynamically from super-peer (normal regime) to pure P2P (during attacks) in order to survive. Actually, pure P2P networks are better at surviving attacks but super-peer networks are perfect for handling failures.

4. Security ontologies

In this section, we describe the security attack (SAO) and defence (SDO) ontologies which are used as a common basis by defensive components (DCs) such as intrusion detection components or honey-pot components for interacting with each other and sharing information about attacks and defenses. SAO and SDO are supported by other security ontologies including the security asset-vulnerability ontology (SAVO), the security algorithm-standard ontology (SASO) and the security function ontology (SFO). However, they are not presented here in details due to the space limit.

4.1. SAO and SDO classes

The main class of SAO, called ‘Attack’, has five subclasses, as illustrated in Figure 3: ‘WSAttack’, ‘P2PAttack’, ‘DoSAttack’, ‘SniffingAttack’, and ‘MultiPhasedDistributedAttack’. The class ‘WSAttack’ specifies attacks against Web services. The class ‘P2PAttack’ describes attacks against peer-to-peer (P2P) systems. The class ‘DoSAttack’ defines various denial of service attacks (DoS). The class ‘SniffingAttack’ characterise security attacks against communications among different parties. The class ‘MultiPhasedDistributedAttack’ specifies multi-phased distributed attacks which is a mixture of attacks derived from ‘DoSAttack’, ‘P2PAttack’, ‘WSAttack’, and ‘SniffingAttack’. We consider only these major classes of attacks because our component-based software systems (CBSSs) can be vulnerable to them. Also, we assume that crypto systems and passwords are fully protected, hence, attacks against them are not taken into account. Moreover, the classes ‘Attack’ and ‘Defence’ have a property called ‘hasRelation’ which ties them together, as depicted in Figure 3. The property ‘hasDescription’ contains specifications of attacks and countermeasures in the low-level syntax.

Figure 2. Illustrated example

If a DC (from a coalition) detects a new attack, it adds the attack as a new class to SAO and shares the ontology with other members of the coalition at runtime. When any coalition member develops a defense against this attack, then it adds a countermeasure to SDO as a new class and distributes this ontology among other coalition members. Finally, such coalitions can be also very helpful for detecting distributed multi-phased attacks such as Mitnick attacks, mitigating and resisting them.

We also have implemented a prototype of our framework to support and demonstrate its key ideas.
‘SniffingDefence’) and multi-phased distributed attacks (the class ‘MultiPhasedDistributedDefence’).

4.1.1. Classes ‘WSAttack’ and ‘WSDefence’
Web services have been deployed recently by many companies including governments, banks, large corporations, etc because of their simplicity of use, platform independence, XML/SOAP support, rich functionality and interoperability. However, Web services also have raised many new unexplored security issues as well as new ways of exploiting inherited old security threats. Attackers do not need to scan the Web in order to find targets. They just go to UDDI Business Registry (UBR) and get all required information to perform attacks against Web services.

![Figure 4. The class ‘WSAttack’](image)

As previously mentioned, the class ‘WSAttack’ defines security attacks against Web services while the class ‘WSDefence’ specifies defenses. The class ‘WSAttack’ has five subclasses, as illustrated in Figure 4: ‘DiscoveryAttack’, ‘ApplicationAttack’, ‘SOAPAttack’, ‘XMLAttack’, and ‘SemanticAttack’. We consider the class ‘ApplicationAttack’ because many application-related security threats have been “reborn” in the Web service context. ‘SemanticAttack’ describes new types of attacks that particular target the new generation of Web services, semantic Web services. For example, the attacker can create a very complicated ontology that can crash an OWL parser. Examples of ‘DiscoveryAttack’, ‘SOAPAttack’ and ‘XMLAttack’ are UDDI attacks, SOAP Replay Attacks and XML Injection respectively. In order to reach their malicious goals, some smart attackers might create new attacks by combining various classes of attacks. All these classes of attacks have many subclasses that are not presented here due to limited space. Also, we do not describe all types of defenses here due to the same reason. However, we mention few of them.

The class ‘WSDefence’ has several subclasses including the class ‘WSDoSDefence’ and ‘ParsingDefence’ which specifies countermeasures against XML parsing attacks such as examining of structural validity of XML documents according to the XML specification, analysing and validating XML Schema Definition (XSD) files that are utilized to describe the structure of XML documents, and blocking XML content-level attacks (restrictions to XML elements).

The class ‘XMLAttack’ has several subclasses including ‘ParsingAttack’, ‘XMLInjectionAttack’ and ‘XPathInjectionAttack’. The XML injection attack (the class ‘XMLInjectionAttack’) occurs when user input is passed to the XML stream [24]. Due to the fact that the XML document can be parsed by the middleware or the database (DB), XML code can be injected to the DB, and when it is retrieved from the DB, it becomes the part of the XML stream. Below we exemplify the XML injection attack:

```xml
<User>
  <ID>12345</ID>
  <Name>Bad Guy</Name>
  <Email>badguy@oops.com</Email>
  <Addr>Bad St</Addr><ID>0</ID><Addr>Bad St</Addr>
</User>
```

At first, a unique user ID equals 12345, however, the attacker enters also a fake street address (Bad St)<Addr ID=0><Addr Bad St). After parsing the address, user ID is rewritten (ID=0) because SAX parsers allow overwriting earlier nodes (SAX attacks). DOM parsers are more complicated and intelligent and can withstand the XML injection attacks, however, they cannot resist against other types of attacks including DoS attacks when hackers can send extremely complicated but legal XML documents (the class ‘DOMAttack’). It forces the system to create huge objects in memory and deplete free memory. The class ‘WSDefence’ is not presented here due to limited space.

4.1.2. Classes ‘P2PAttack’ and ‘P2PDefence’
P2P networks [2,9] have many advantages over the client/server paradigm, however, they also suffer from inherited vulnerabilities because of the dynamic nature of peers. Besides, they face external attacks targeting their nodes, routing mechanisms, etc. The majority of P2P attacks is expressed by the class ‘P2PAttack’ while countermeasures are represented by the class ‘P2PDefence’. We do not introduce the detailed description of both classes here due to limited space.

An example of attacks against P2P systems is an attack on anonymity such as the cache-timing attack [4] (the class ‘P2PCacheTimingAttack’), which aims to determine the contents of a user’s cache and his Internet usage patterns as well. By taking timing measurements, the attacker can determine if the client had to fetch the requested content from the web or it was returned from the user’s cache. To be protected against such attacks, the use of Onion Routing or Crowds [22] is proposed.

4.1.3. DoS attacks and countermeasures
The DoS attack (the class ‘DoSAttack’) refers to such scenario where the attacker tries to make certain resources too busy to answer legitimate requests. The
number of DoS attacks has increased greatly in recent years and new types appear rapidly. The distributed DoS (DDoS) (the class ‘DDoSAttack’) and distributed reflective DoS (DRDoS) attacks (the class ‘DRDoSAttack’) are subclasses of the class ‘DoSAttack’, as illustrated in Figure 5, and employ similar principles. The main difference is that the attacker uses several hosts which are rather difficult to identify and block. Besides, DDoS and DRDoS attacks are critical threats to the Internet. Among the numerous types of the DoS attacks, we could name Teardrop or the land attack, etc. For example, during the land attack (the class ‘LandAttack’) the attacker sends SYN packets to a victim with the spoofed IP address (the same source and destination IP address and port number as the victim). In such situation, the system thinks that a TCP/IP session opening packet was sent to itself (dead circle), which afterwards results in a system failure or crash.

Countermeasures for defending against DoS are defined by the class ‘DoSDefence’ that consists of several subclasses. Actually, there is no fundamental defence against DDoS and DRDoS attacks. The TCP SYN attacks can be mitigated (the class ‘TCP SYN Defence’) by increasing the size of the SYN ACK queue, decreasing the time-out waiting for the three-way handshake, and installing software patches. Other defenses are not presented here due to limited space.

4.1.4. Sniffing attacks and countermeasures

The sniffing (also called snooping) attack (the class ‘SniffingAttack’) includes such attacks that allow the malicious user to get information about computer networks or their network traffic. Sniffing attacks primarily target the initial connections between parties to obtain usernames, passwords, shared keys, and any confidential information. As the rule, well-performed sniffing attacks cannot be detected and usually precede spoofing or hijack attacks. Countermeasures for preventing or stopping sniffing attacks include active monitoring by IDSs for sniffing signatures (e.g. lost or delayed packets) and using encrypted traffic over all network connections.

4.1.5. Multi-phased distributed attacks and defences

The main reason why multi-phased distributed attacks (the class ‘MultiPhasedDistributedAttack’) have gained our attention is the fact that it is rather hard to detect them and protect systems against them. The Mitnick attack and various coordinated attacks are all subclasses of multi-phased distributed attacks. To detect and resist such kind of attacks, different distributed parties should operate as a coalition.

4.2. SAVO classes

The security asset-vulnerability ontology (SAVO), illustrated in Figure 6, is a high level security ontology that depicts information security in simplified manner specifically for humans and software agents and its design is based upon our expertise and knowledge of information security and somehow correlates with [1,13,17,25,28]. SAVO involves other security ontologies such as SASO, SAFO, SAO, and SDO. However, Figure 6 does not show all elements of our ontology (e.g. subclasses of ‘Asset’ or ‘Consequence’) due to limited space.
Figure 6. Graphical description of SAVO

We start from the term “threat” (the class ‘Threat’) which is any occurrence that may cause any unwanted outcome for a company. A threat agent (the class ‘ThreatAgent’) is an agent that can use a threat in order to exploit vulnerability, while vulnerability (the class ‘Vulnerability’) in turn is the absence or the weakness of defence (the class ‘Defence’). Besides, a threat agent may use a threat to perform an attack (the class ‘Attack’) and endanger an asset (the class ‘Asset’). A threat agent exploits vulnerability for attacking a peer or a host (the class ‘Peer’) which hosts assets. An attack causes a security event (the class ‘SecurityEvent’) which can precede or proceed another one and which has a precondition (the class ‘Precondition’) that relates to vulnerability. It is worth mentioning that SAVO is easily modifiable and extendable through adding additional subclasses to the core classes such as the class ‘Asset’ or ‘Vulnerability’.

5. Example

In this section, we explain how a gaming system’s constituent components, described in Section 2, collaborate, identify distributed attacks, and securely share information about them and developed countermeasures using our security ontologies.

5.1. Scenario and analysis

A game player (Client 1) from the U.S.A. wishes to play for the first time a game called Heroes of Might and Magic 6 (HMM6). He downloads a game client application from a website called www.hmm6.com. After all preparations such as installation of game software and specification of required parameters, he starts playing. In addition, game players can perform other activities such as chatting or conducting videoconferences. As illustrated in Figure 1, Client 1 connects to P2 and plays HMM6. At the same time, P2 actively communicates with P1 for data synchronisation. However, A tries to perform the Mitnick attack in conjunction with Web service attacks, as described in Section 2, against P1 and P2. As mentioned above, Mitnick attacks can be detected by several members of a coalition who are distributed over the network and who cooperate with each other. As such, P1 and P2 cooperate and constantly exchange data about security attacks and defenses.

Assume that A attacks P1 (an IP address equals 147.202.46.43) and P2 (IP address is 147.202.46.40) from a host with an IP address 192.2.1.23. All stages of the attack have been explained in Section 2. Here, we only describe how P1 and P2 respond to the attack. At the first stage of the Mitnick attack, A spoofs P2’s IP address and initiates TCP connections with P1. Then, P1 detects that many partial TCP connections are opened from P2’s IP address 147.202.46.40 at the period of time from 14.40.00 to 14.45.00 on 07.07.2007. Hence, P1 sends P2 a message (the class ‘SecurityEvent’ from SAVO) to verify if P2 really has opened all these connections, and if there is no response from P2, then P1 will take prompt actions, e.g., sends an alert message regarding a possible attack to other peers (and a manager represented by a peer-manager as illustrated in Figure 7). Due to limited space we demonstrate only an extract of the description of our security ontologies. The class ‘SecurityEvent’ from SAVO and its data properties are specified as follows:

```java
Class SVO.SecurityEvent {
    Property EVENT;  Property TIME;  Property IP_ADDRESS; }
```

Properties ‘EVENT’, ‘TIME’, and ‘IP_ADDRESS’ specify when a security event has occurred, during what period of time, and from which IP address it has been initiated. The instance of the class ‘SecurityEvent’ called ‘SecEv1’ is defined as follows:

```java
SecEv1 = new SecurityEvent {
    EVENT = MANY_OPENED_CONNECTIONS;
    IP_ADDRESS = 192.2.1.23;
    TIME = 14.40.00 07.07.2007 - 14.45.00 07.07.2007; }
```

Then, ‘SecEv1’ is sent to P2 and, if P2 does not reply after three minutes, P1 generates an alert (SecurityEvent). Simultaneously, P2 detects that someone from the IP address 190.2.1.23 tries to predict a TCP sequence number. This event is specified as:

```java
SecEv2 = new SecurityEvent {
    EVENT = TCP_SEQUENCE_NUMBER_PREDICTION;
    IP_ADDRESS = 192.2.1.23;
    TIME = 14.40.00 07.07.2007 - 14.45.00 07.07.2007; }
```

Since P2 receives ‘SecEv1’ and replies to P1 but in turn P2 does not get any acknowledgement from P1,
P2 concludes that a TCP SYN attack is occurring and sends an alert message containing 'SecEv2' and 'SecEv3' to P1 and other peers (and a peer-manager who can govern peers in a centralised manner). Actually, there are many rules that specify attacks (similar to expert-based systems) stored in the Ontology Repository. In our case, a manager concludes that there is the Mitnick attack when P1 detects many partial connections from P2 while P2 registers many TCP sequence number prediction attempts at the same period of time. Also, P2 identifies XML injection and WS Probing attacks at the same period of time from the same IP address.

A manager concludes that all these attacks are related to each other and performed by one attacker. Consequently, a manager labels the combination of these attacks as the WS Mitnick attack, as introduced in Section 3 and shown in Figure 2, and sends a description of the attack using SAO to all other peers such as P1, P2, and P3. The class for the WS Mitnick attack is shown below:

```
SecEv4 = new SecurityEvent {
  EVENT = SYN_ATTACK; IP_ADDRESS = 192.2.1.23;
  TIME = 14.40.00 07.07.2007 - 14.45.00 07.07.2007;
}
```

```
SecEv5 = new SecurityEvent {
  EVENT = XMLINJECTIONATTACK; IP_ADDRESS = 192.2.1.23;
  TIME = 14.40.00 07.07.2007 - 14.45.00 07.07.2007;
}
```

```
SecEv6 = new SecurityEvent {
  EVENT = WSPROBINGATTACK; IP_ADDRESS = 192.2.1.23;
  TIME = 14.40.00 07.07.2007 - 14.45.00 07.07.2007;
}
```

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

After the attack has been identified, a group of a game company’s software developers and system administrators or intelligent agents start developing countermeasures and after finding them, they distribute them among other components which belong to a gaming system such as P1, P2, P3, and P4. We do not explain here how countermeasures are developed and deployed, we just mention that they are contained in SDO and demonstrate how a description of these defenses can be distributed.

If P1 detects the TCP SYN attack it will close all connections from the spoofed IP address (147.202.46.40), then increase the size of the connection queue, decrease the time-out waiting for the three-way handshake, and block all connections from the A’s IP address (192.2.1.23). This process is illustrated as follows:

```
Class WSMitnick Defence [Array ACTIONS];
WSMitnickDefence_1 = new WSMitnickDefence {
  ACTIONS { CLOSE CONNECTIONS_IP=147.202.46.40;
  INCREASE_CONNECTION_QUEUE=+50;
  DECREASE_TIMEOUT=-1ms; BLOCK_IP=192.2.1.23;}
```

As described in Section 3, new versions of SAO and SDO should be securely distributed. To ensure this, they should be encrypted by using the AES algorithm for delivering confidentiality (we assume that all private keys have been successfully distributed using RSA). Then, these new versions of security ontologies are signed and time-stamped in order to provide integrity, authentication, and non-repudiation using RSA and SHA-256 (the class ‘SHA256’ from SASO) [25]. To conclude, the Mitnick attack should be treated as a baseline to verify if the system really can detect coordinated multi-phased distributed attacks.

6. Prototype implementation

In this section, we briefly introduce a prototype, illustrated in Figure 7, of the gaming system presented above and implemented on the top of our reference architecture called SECROBAT [32] in order to demonstrate the applicability of our approach.

```
Figure 7. Prototype implementation
```

As previously mentioned, components in our approach can be treated as peers that provide various services. The architecture (implemented in Java 6 [7] together with Groovy [5]) of every peer consists of the modules that are connected to a microkernel using message exchange. The gaming system interacts with the external world through the defensive components (DCs) combined with instances of Snort [23], both of which are part of the peers, as depicted in Figure 7. Security ontologies are stored in the Ontology Repository and employed to share information about security attacks and defenses among peers. The manager (also a peer) represented by a human or a software agent controls the system and its constituent components and instantiates ontologies.

Initially, security ontologies are specified in OWL-DL using the Protégé ontology editor [21] and contain information extracted from Snort rules regarding existed security attacks. Instances of Snort are employed to detect simple security attacks such as TCP SYN attacks, however, to identify more complex attacks such as the Mitnick attacks, these instances should operate as a coalition. Hence, they register and
send information about attacks to DCs that in turn forward it to the manager who extracts this information from Snort rules, analyses it, and then converts to OWL-DL, and adds it to the instance (only one) of SAO (only if the signature of the attack has been already added to the ontology) using Jena Framework [8] for building Semantic Web applications that supports OWL and includes a rule-based inference engine. Furthermore, the manager decides how to react and if there is a new type of a security attack, then it is added to SAO which is updated in the repository. When a new countermeasure is developed, information about this is added to SDO, and the manager distributes orders to take prompt action to DCs. Besides, to share all this ontologies in a trusted way, they are encrypted using the AES algorithm, and then signed and time-stamped utilising RSA and SHA-256 (supported by Java Cryptography Architecture (JCA) [7]), distributed among DCs using the mechanism of messaging.

7. Related works

Since, ontologies are mainly utilised to express semantic information, they can also 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 [19], OWL-S [20], Semantic Web Services Language (SWSL) [26]. However, none of them is specifically designed to express the security requirements and capabilities of Web services. Besides, there is a limited number of works in the area of security ontologies including the NRL security ontology [13] which is based upon [1] and expresses types of security information such as security algorithms, mechanisms, protocols, etc, for all types of resources.

Moreover, a research related to our work [27] 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 [15] is a framework with formal semantics in Datalog that allows to model relationships among goals and actors through the use of permissions, delegations, refinements, and ownerships. Besides, it also models security using security ontologies. However, our security ontologies are richer and are specified using a formal language called GIZKA that supports logic programming that is an additional advantage. Security ontologies such as SAVO and SASO are partially based on [1,2,17,28] while SFO applies few ideas extracted from [10,11,31]. SAO and SDO encompass some thoughts taken from a number of works [1,13,14,17,24,25,30,28]. In our previous work [30], we have developed the security attack ontology for Web services only 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 used by different parties, and describe multi-phased distributed attacks. Hence, our ontologies cover many areas of information security related to security attacks and defences against them including but not limited to DoS attacks, attacks against Web services and P2P systems, countermeasures, and so on. Also, collaborative detection and defense systems are presented in [12,34] where techniques for identifying and resisting various types of attacks including coordinated and distributed DoS attacks [16] are introduced.

Currently, there are several WS-* standards (WS-Security, WS-Policy, WS-Trust, WS-Privacy, WS-Federation, WS-SecureConversation, and WS-Authorization, etc) [3,6] used in our approach as building blocks for creating countermeasures against attacks on Web services. The Mitnick attack [28,34], which is used as the core example to demonstrate our approach, is related to the men-in-the-middle attack that exploits weakness of the TCP protocol design in making a TCP connection between two hosts called the three-way handshake. Descriptions of many other security attacks as well as many security terms, concepts, and protocols can be found in [25,33].

8. Conclusion and future work

Nowadays, users of software applications and component-based software systems in particular, require more features, flexibility and better protection. Hence, they become highly complicated and increasingly distributed. At the same time, new attacks, especially distributed multi-phased attacks, which are difficult to identify and mitigate, appear.

In this paper, we have demonstrated that only through the collaboration of a system’s constituent components it is possible to detect and resist against such attacks. However, components should have a common basis (vocabulary) to allow them to exchange information with each other about security attacks and defenses. Therefore, we have utilised an ontological approach, security ontologies in particular, to specify information security issues in a way understandable to both humans and software agents. Moreover, we have analysed reasons why ontologies were chosen over
taxonomies. Then, we have introduced our security ontologies including the security attack ontology (SAO) and the security defence ontology (SDO). SAO and SDO express relationships among security attacks and defenses against them and closely correlate with each other. Both of them have been utilised with other security ontologies with the purpose to provide a common basis for various components in our system. For example, to detect any of the multi-phased distributed attacks such as Mitnick attacks, components should operate as a coalition. As long as the attack is evolving, SAO will also be evolving and will be shared among other members of the coalition. When a new countermeasure is created by any coalition member, it is added directly as a new class to SDO which is then securely distributed among other members of the coalition. Similar to other our security ontologies, SAO and SDO have been specified in OWL. Finally, we have introduced the prototype based on our approach to demonstrate the applicability of our approach to the real world.

In future work, we will develop a trust ontology, then integrate it with our security ontologies, and validate them.

8. References