

Secrobat: Secure and Robust Component-based Architectures

Artem Vorobiev and Jun Han

Faculty of ICT, Swinburne University of Technology, Melbourne, Australia
{avorobiev, jhan}@ict.swin.edu.au

Abstract

Software systems, component-based systems (CBS) in particular, have a lot of vulnerabilities that may be exploited by intruders. Companies spend much time and money to “patch” them up. It is partly due to the fact that a system’s security features are often added to the system after its functional requirements have been addressed. As such, system security features are not systematically designed into the system, and consequently the system has inherent security “holes”. Therefore, there is a strong need for a systematic engineering approach to developing secure and robust systems, especially distributed systems, by considering functional and security requirements at the same time. In particular, these systems should be highly adaptive and reconfigurable in order to resist different types of attacks and failures. This paper introduces a reference architecture, called Secrobat, for creating secure and robust CBS. It has several key features including defensive components and the adaptive and reconfigurable architecture with the hybrid peer/super-peer structure. The reference architecture is illustrated with an example gaming system.

1. Introduction

Many software applications have design flaws and “bugs” that may be exploited by attackers, and companies spend much time and money to fix them up. This is due to the fact that a system’s security features are often added to the system after its functional requirements have been addressed [5,6,23]. Hence, security is not systematically incorporated into the system, and consequently the system has inherent security problems. As such, there is a strong need for a systematic engineering approach that facilitates developing secure and robust distributed software systems by considering security and functional requirements simultaneously. Furthermore, these

systems should be adaptive and reconfigurable dynamically and should support traditional security mechanisms including intrusion detection [18], honeypots [27], key distribution [22] etc in order to resist different types of attacks and failures.

In this paper, we present a reference architecture called Secrobat for creating secure and robust component-based (and service-based) systems (CBS) that are highly adaptive and reconfigurable dynamically and can resist different types of attacks and failures. For example, if there is a Distributed Denial of Service (DDoS) brute-force flooding attack [18] the system changes its architecture from a pure P2P structure [10] to a super-peer structure [33]. Against other types of attacks such as sniffing or DoS Syn-Flood attacks defensive components can be used, including intrusion detection components (IDC), honeypot components (HC) and key distribution components (KDC). They help Secrobat-based systems to be stable and attack-protected by providing traditional information security services such as registering, resisting, and preventing attacks, trace-backing and confusing intruders, protecting components, and distributing and updating security keys. Secrobat can be implemented using different types of distributed systems such as Web Service (WS) based systems [2,11], Peer-to-Peer (P2P) systems [10,33], and Grid computing systems [32].

The reference architecture can also be utilised as a reference guide for researching security problems. For example, it can be used to investigate how the components of a system can be organized in such an effective way that components with strong security properties can protect components with weak security properties and as a result the whole system can be more secure.

The paper is organised as follows. In section 2, we present an example gaming system and raise several questions about its security and robustness. In section 3, we present our reference architecture called Secrobat. In section 4, we describe how Secrobat can be applied to the gaming system. In section 5, we

introduce an implementation strategy for our reference architecture. Section 6 reviews related work. Finally, section 7 draws some conclusions and point to some future directions.

2. Motivation

In this section, we present a hypothetical gaming system and identify several important questions about its security and robustness.

2.1. Example

The computer game industry has grown rather fast for the last several years especially in such areas as online games for desktops, game consoles and mobile phones. One of the biggest markets is China's online gaming market with revenue at US\$298 million in 2004, US\$467 million in 2005 and US\$1.3-2 billion in 2009 [19]. It does not even include the sale of virtual gear for game characters, such as clothing, which is also a fast growing business in China. The mobile gaming market has become the "secret weapon" for many companies in China. Faced with huge potential, industry players try to take advantages of the expected growth. However, because of the complexity of online gaming systems, piracy and other software security threats such as attacks on game servers and game players, game companies have to develop strategies and technologies to prevent attacks and protect their clients. They aim to help build trust relations between players and game providers while deriving a healthy profit. Furthermore, since small game companies have limited budgets and cannot compete well with large game providers, they have to develop strategies to implement and maintain secure and robust game systems with limited resources.

2.2. Main issues

Firstly, because of computer piracy, it is very difficult to make a profit by selling copies of online games. Hence, one strategy would be to distribute them free while game players would pay only for game time. Therefore, online payments have to be secured because users do not want to lose their money.

What parts of the distributed gaming system should be responsible for protecting online payments? How should they be organised?

Secondly, if a game becomes quite popular it attracts a large number of game players which can overwhelm game servers. Also, game servers as well as some other game system components can be

attacked by using different types of attacks such as sniffing or brute-force DoS/DDoS attacks which deplete network bandwidth. It can be a major problem for small game companies as well as for large one. It can be not economically feasible only by adding additional hardware. Hence, there should be other approaches to overcome this issue. One of them is to allow the gaming system to reconfigure dynamically in order to resist attacks and distribute load among game components. Some system components (defensive components) should be specifically designed to protect the gaming system.

What approaches can game providers use in order to solve the problems described above? How should the system be organised in order to withstand attacks and system failures? How should game components be designed and organised?

Finally, game providers need a lot of economic investments, especially in the beginning. Small companies with small budgets simply are not able to start the business because, except of paying for game licenses, they have to pay for hardware, additional software applications such firewalls and intrusion detection systems, technical staff, etc. Hence, there should be a way to run game servers that support as many users as possible while using a small amount of hardware resources. The game system should be easily manageable by a small number of technical specialists. Also the system should be secure and robust and utilise security standards that are free and used by many companies because the system has to interact with many parties such as game players or finance organisations.

How can game providers satisfy such different requirements? What kinds of security techniques should be used in order to allow game providers to easily manage, analyse and reason about a system's security?

In order to be attack-protected and robust, the distributed gaming systems should be designed and implemented using proven security techniques and principles, incorporating appropriately defensive components in system design and adopting adaptive and reconfigurable architectures. In section 4, we present the gaming system with such features, designed following the Secrobat reference architecture.

3. Secrobat

Currently, many accept the argument that a system's security is as strong as its weakest component or link, or even weaker. However, in the human society and biological systems, such an observation

does not hold. In such a system, for example, the strong can protect the weak, resulting in a system that is stronger than the weakest individual, and even than the strongest individual. Biological systems [15] and autonomic computing principles [12] serve as an inspiration to our approach, and we argue that software and security engineers can learn from biologists regarding living systems in order to solve many security problems in computer systems. For example, corals can organise colonies where individual organisms protect each other. Also, such colonies can change their structure if the environment changes or predators attack.

Our reference architecture called Secrobat supports security strategies that resist attacks and avoid failures, including defensive components and reconfigurable architecture with a hybrid structure.

1) Defensive components such as intrusion detection components (IDC), honeypot components (HC), and key distribution components (KDC) allow studying attacks and keeping secure connections between components. They use traditional information security mechanisms but they are exploited in different ways. For example, traditionally, firewalls/intrusion detection systems (IDS) can detect and block attacks but they are stand-alone applications for undertaking particular tasks and are not integrated parts of the whole system. However, IDC, HC, and KDC are integrated parts of Secrobat-based systems and they can be reconfigured, added, deleted or healed dynamically.

2) Secrobat-based systems adapt their hybrid peer/super-peer structure dynamically during an attack or when the environment changes. For example, during DDoS brute-force attacks traditional approaches use IDS [18] together with firewalls to detect and protect the system. They are utilised ineffectively and try to resist attacks by blocking malicious hosts. However, Secrobat-based systems can not only block hosts but also change their structure from super-peer (normal regime) to pure P2P (attacks) dynamically in order to allow the system to survive during DDoS attacks.

3.1. Basic design principles

Many current software systems have vulnerabilities due to the lack of proper design and implementation. They are developed with only functionality in mind [5,6], hence, in our approach we take into account both security and functionality during the whole software development cycle. We employ traditional information security techniques including cryptographic mechanisms, intrusion detection techniques and key distribution schemes and software security engineering

approaches such as AOP for security [28] and security patterns [23,24,25] to integrate security and functionality. We adopt several existing traditional information security principles in Secrobat: cryptographic keys are distributed and assigned; all connections and messages are encrypted (helps to prevent sniffing of traffic); messages are also signed; single points of responsibilities are avoided. Several types of defensive components are introduced to improve security: IDC, HC and KDC. Messages that need the highest level of security are protected with steganography methods. For example, very important information can be inserted in a message body. Because of a large amount of such messages there is a quite small chance that an attacker will get any important information even if he/she sniffs network traffic and then decrypts messages. Mirroring and replication of components are implemented to restore systems after attacks.

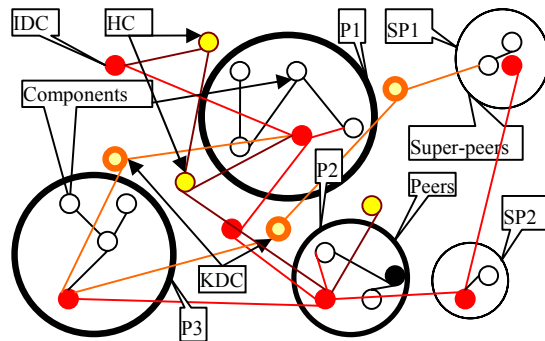


Figure 1. The structure

As shown in Figure 1, components are grouped together to create peers (nodes P1, P2, P3, SP1 and SP2). Some of peers can be super-peers (SP1 and SP2), which act as centralized servers to a set of clients and as peers to each other [33]. In fact, a single component also can be a peer.

Every peer in Secrobat has a plan about what to do during and after attacks, which can be described in an ADL such as Dynamic WRIGHT [1,8]. For example, some peers can imitate that they are “dead” and some can try to resist attacks or forward all malicious network traffic to IDC or HC. Peers also know how to recover after attacks because super-peers store states of peers and the system.

3.2. Defensive components

There are two types of components in Secrobat, normal and defensive. Normal components such as storage are responsible for the functionality of the

system while defensive components such as HC, IDC, and KDC provide security functionality. Defensive components apply traditional security techniques used in intrusion detection systems (IDSs), honeypots and key distribution (KD) systems, but they are utilised differently. They are integrated parts of the entire system while in traditional systems IDSs are used as stand-alone applications. Such an approach allows us to have better control over the whole system.

Honeypot components. HC or fake components are used to confuse attackers. They pretend to be normal components but they really shadow them, create honeynets and register attacks. HC also trace-back to intruders, behave as spies, study attacks etc. HC are linked to the system through IDC and used only during attacks while IDC are utilised all the time and can forward malicious traffic to a honeynet (one or several HC) during attacks.

Key Distribution Components. KDC are responsible for providing and updating cryptographic keys in the system and they are protected by IDC. Keys are used to encrypt and sign messages.

Intrusion Detection Components. IDC (opposite to HC) do not pretend to be normal components and do not create honeynets. They register attacks in source networks, victim networks and intermediate networks; resist and prevent attacks; trace-back intruders; and protect components. IDC are distributed in the network and analyze network traffic and behavior of components, and work as spies or wrappers to cover and protect “weak” components. Some IDC operate as ordinary firewalls and filters by blocking or filtering network traffic from suspicious peers or components. Other IDC use antivirus techniques for protecting Secrobat from network pathogens such as viruses, worms, Trojans etc. Also every IDC looks after other defensive components because hackers can try to attack protection mechanisms.

After an attack is detected IDC can follow several strategies to minimize and deflect the damage from the attack: send signals to the network to reconfigure it, filter the network, block attack agents etc. IDC log attacks and analyze them to create traffic patterns and trace-back intruders. Also IDC examine protocols and normal components that are used in the system. Since network traffic in Secrobat-based systems goes through IDC, it allows systems to have better control over various system parts.

3.3. The reconfigurable architecture

Well-designed architecture saves a lot of time and money because many application bugs and design flaws can be avoided from the beginning.

The structure. The Secrobat architecture has a hybrid structure where software components are organized into peers. It can change its structure dynamically during the normal regime and during attacks (see below). Secrobat unites two types of architectures: the pure P2P architecture and the super-peer architecture. The pure P2P network [10] is homogeneous where most peers have approximately the same number of links while the super-peer network [33] is inhomogeneous with the majority of the peers have one or two links but a few peers have a large number of links, guaranteeing that the system is fully connected. By combining these two types of network topologies, we have the robustness of super-peer networks to failures coupled with the robustness of pure P2P networks to attacks. Pure P2P networks are better at surviving attacks while a super-peer topology is ideal for handling normal operation including query forwarding and surviving random failures [34].

Temporal servers. Some peers can behave as temporal storage or index servers for a limited period of time. For example, index servers can store a list of neighboring peers that can be used to improve search speed in the network if current search speed is not adequate (search requests can overwhelm a pure P2P system because they are broadcasted to all neighboring peers). Temporal servers may be main targets for attacks because they may store index information and distribute parts of this information among super-peers.

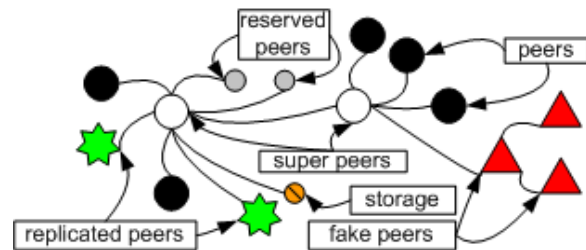


Figure 2. The architecture in a normal regime

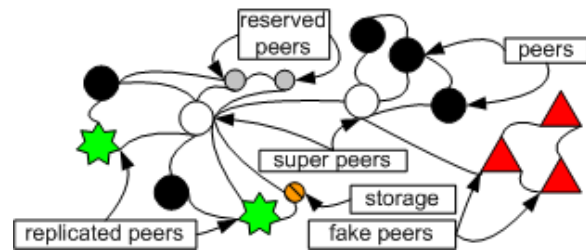


Figure 3. The architecture during attacks

Two regimes. In the normal regime, as illustrated in Figure 2, every peer is connected to one of the super-peers. However, if there is an attack (e.g., a brute-force DDoS attack) on the system the structure changes, as shown in Figure 3. During an attack, peers try to connect to as many peers and super-peers as possible. Instead of connecting to only one of super-peers. For example, the temporal storage server starts behaving as peers, which is still responsible for search in the system but needs to cooperate with other peers (see Figure 3). Malicious traffic is blocked or forwarded by IDC to honeynets for studying. For example, the gaming system can survive during attacks or failures because pure P2P networks are better at surviving attacks but super-peer networks are better for handling failures.

Fake peers. IDC together with HC can be organized into fake peers in order to confuse intruders.

Replication. Peers with important data can be replicated and repaired when some network segments are blocked or destroyed. This functionality is supported by reserved peers that “sleep” and do not participate in the network activity. Each peer has a significance number and if this number is 0 it means that the peer does not need to be replicated. If this number equals 1 or 2 it means that the peer should be replicated once or twice. The case of super-peers is the same. In the case of the gaming system, the replication mechanism helps the system to survive during attacks or network failures. Peers that are responsible for storing information about user accounts will be replicated and available all the time. However, the replication cost (computer resources) can be high.

All peers have tables of reserved peers which are utilised when peers need to be replicated. Every super-peer stores states of its peers and copies of states from other super-peers. These states are used during and after attacks to restore “dead” peers. For example, upon detecting DDoS attacks on the gaming system, peers begin replacing their “dead” neighbors with replicated peers. States of “dead” peers are taken from super-peers and then copied to replicated peers. This process of replacements continues till the end of the attack or till the number of replicated peers becomes 0. At the same time, if the difference between the significance number of the attacked peer and the real number of its replicated peer is more than 0, the system tries to create new copies of the attacked peers and give time to IDC to block malicious network traffic. Once the system detects no more attacks, it returns to the normal regime and tries to restore “dead” peers. As the part of the normal regime, it rebuilds a list of reserved and replicated peers to use them during attacks in the future.

4. Secrobot and the gaming system

In this section, we apply Secrobot to the gaming system and address the issues described in section 2.

4.1. A gaming scenario

If a game player (Client1), as shown in Figure 4, wants to play a game called Heroes of Might and Magic (HMM3) for the first time, he downloads a game client software program from the website hmm3.com. After installing it, he specifies that he wants to play HMM3 with game players from Oceania in their 20th from 9am till 5pm. A system manager (Manager1) checks if Client1 has enough funds to play and decides what security policies should be enforced for Client1, and what security techniques (IPSec and 3DES) should be used because security measures may reduce performance or network bandwidth. Manager1 stores all its data on a storage server (SS1).

In addition to playing games, game players can do other collaborative activities including chatting or conducting videoconferences. Every game player can make a payment that goes through the nearest IDC and is checked and forwarded to the bank.

4.2. System architecture

The Secrobot-based gaming system in the normal regime, as illustrated in Figure 4, looks like a super-peer system where the majority of peers have one or two links but a few peers have a large number of links. All connections between game players (clients) and the system (game servers (GS1,GS2,GS3), storage servers (SS1,SS2,SS3) etc) are controlled by intrusion detection components (IDC1 and IDC2). A key distribution component (KDC1) is responsible for distributing keys and certificates in order to maintain secure connections among different parts of the system. System components as well as game players link to each other through IDC (super-peers). A honeynet (HC1) is used to collect new data for IDC attack patterns, study attacks and attract intruders by exposing well-known vulnerabilities. Storage servers (SS) are responsible for storing personal game players' data, system security properties and policies, etc. Game servers (GS) provide game services and store game states. They are connected to other GS directly (P2P), however, they are linked to game players through IDC. Reserved (Reserved1) peers have been presented in section 3.3. The main task of the manager (Manager1) is to organize other components and peers and manage the system.

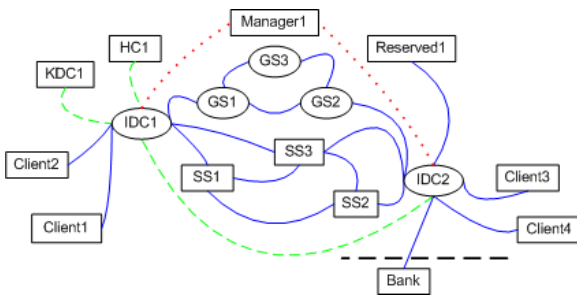


Figure 4. The Secrobat-based gaming system in the normal regime

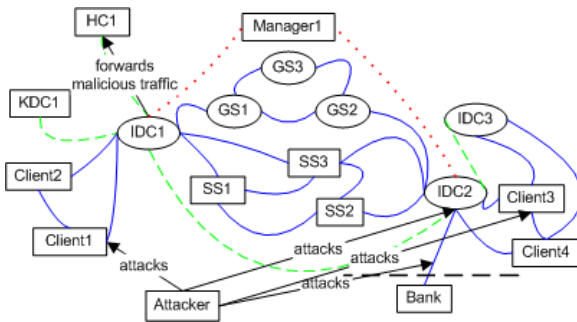


Figure 5. The Secrobat-based gaming system during attacks

4.3. Analysis of security features

In this section, we explain how the above system design addresses the security issues highlighted in section 2.2.

Firstly, the network connection between the system and the external bank is encrypted and inspected by IDC2 in order to protect online payments of game players. It is difficult for Attacker to hack communications between Client1 and other peers because IDC1 and IDC2 detect cases of intrusion, study log files, and use information about the studied attacks from HC1. Messages are also encrypted using cryptographic keys provided by KDC1.

Secondly, when under attack, IDC1 tries to protect Client1 and forwards malicious traffic to HC1 that studies and trace-backs to Attacker. To withstand brute-force DDoS attacks, the system changes its structure, as illustrated in Figure 5. The additional IDC3 is added using the Reserved1 peer in order to resist attacks and protect system components and clients. Also IDC1, IDC2 and IDC3 look after each other and HC1. Defensive mechanisms described in section 3 are active. Clients connect to other clients directly. For example, Client1 and Client2 can still

play the game even during attacks. Also, if a game provider needs more resources, the system can try to distribute load dynamically and evenly. To minimize expenses, a game provider uses a manager (Manager1) to operate the system and utilise Client3 and Client4 as game servers. However, system data stored on these clients should be encrypted.

Finally, the system can use a security description and reasoning scheme that supports security properties expressed in SCL/E-SCL [13,14,29]. If a peer hosts Web services, these SCL/E-SCL security properties can be expressed in the form of OWL/OWL-S [20,21] and SWRL [26]. (See also section 5 below.)

5. The implementation strategy

In this section, we describe one of the possible implementation strategies that can be used to develop our reference architecture. It consists of several aspects.

First, to implement our reference architecture and defensive components, we utilise the ROAD (Role-oriented adaptive design) framework [3], which makes systems more adaptable and adaptive through the following three main features:

- Role-player relationship. ROAD separates and relates the role from the object or agent that plays that role.
- Role-role relationship. Functional roles are treated as first-order entities and associated by *contracts* that mediate the interactions between them. As such, the system's explicit organisation structure is created.
- Management roles. Management roles (managers) manage the systems' structures, including roles, contracts, role-player bindings, and role-role contracts, to realise system adaptation.

The ROAD framework is developed using Java and utilises the AOP approach [28] in order to integrate the functional and non-functional aspects of the systems. However, it does not directly deal with the security characteristics of components and systems, even though ROAD contracts can cater for various non-functional properties.

Second, we incorporate security contracts into ROAD contracts and specify them using SCL/E-SCL (the Security Characterisation language) [13,14,29]. SCL/E-SCL allows the component developers to specify the security properties of the component in the form of logic program clauses, and the compatibility of interacting components regarding their security properties can therefore be checked automatically. It can describe security properties in a static manner and

also it can specify dynamic adaptive software systems where the security properties of the components and system may depend on time, have a particular probability, or even change in nature.

Third, we choose Web services (WS) [2,11] as an implementation platform for our reference architecture. WS can be hosted by peers and super-peers and also can be organised into complex structures. Similarly, other technology platforms can be considered.

Fourth, as mentioned above, to specify the security properties of WS and to describe how WS and the entire system should reconfigure and adapt, we adapt and incorporate SCL/E-SCL into WS descriptions, utilising the Semantic Web languages OWL/OWL-S [20,21] and SWRL [26].

Fifth, to interoperate with each other, WS use a general security vocabulary represented by ontologies. In particular, defensive WS utilise security attack and defence ontologies [30] in order to detect and protect the system against various attacks, especially distributed multi-phased attacks that can be detected only by distributed intrusion detection systems.

Finally, security patterns [23,24,25] can be applied to develop better defensive mechanisms and complex network structures that withstand different attacks.

6. Related work

Traditional information security approaches [4,22] are usually used as vulnerabilities are discovered and after an application has been developed. However, it is less expensive to fix software “holes” up by considering functional and security requirements at the same time [6,5,23]. To do so, we have proposed the reference architecture that utilises the Aspect-Oriented Programming (AOP) approach [28,16], security patterns [23,24,25] and antipatterns [17], and traditional information security principles [4,22]. Secrobot’s basic design principles are based upon principles of software security engineering (SSE) [7,13,31], that lies in the intersection of software engineering and information security.

The specification and analysis of dynamic software architectures are studied in [1] where WRIGHT is extended with dynamic behaviour. In [8] WRIGHT is extended with non-functional properties. Dynamic self-organising distributed component software architectures are described in [9] using Darwin. However, these approaches do not take into account security while the main goal of our approach is to secure and protect component based systems. In implementing our reference architecture, we use the ROAD framework [3], SCL/E-SCL (the Security

Characterisation language) [13,14,29], Web service (WS) standards [2,11] and Semantic Web approaches [20,21,26].

Our hybrid pure-peer/super-peer structure uses ideas from P2P [10,33] and Grid computing systems [32]. They are utilised to achieve the security objectives in Secrobot by reconfiguring and adapting the hybrid structure. Defensive components in Secrobot utilise methods widely adopted in intrusion detection systems (IDS) [18] and honeynets [27].

The final inspiration to Secrobot has been from biological systems [15] and autonomic computing principles [12], which combine four typical features of biological systems: self-configuring, self-optimizing, self-healing and self-protecting.

7. Conclusion and future work

In order to create robust and attack protected software systems, especially distributed component-based systems (CBS), their functional and security requirements should be considered during the whole software development life cycle. These systems should also be highly adaptive and reconfigurable.

In this paper, we have presented our approach, called Secrobot, that allows Secrobot-based software systems to be robust and attack protected, and described its key features including basic design principles, the reference architecture, defensive components (intrusion detection components, honeypot components, key distribution components), and the hybrid pure-peer/super-peer structure. We have introduced an implementation strategy for our approach using Web services (WS), Semantic Web languages [20,21,26], the ROAD framework [3], security properties [13,14,29] and security ontologies for WS.

In future work, we will utilise the security attack ontology [30] and further develop the security defence ontology in order to allow intrusion detection components to register and resist different types of attacks, especially distributed multi-phased attacks. We intend to investigate how WS with “strong” security properties can protect “weak” WS using security patterns [23] and reconfiguration mechanisms.

8. References

- [1] R. Allen, R. Douence, and D. Garlan, “Specifying and analyzing dynamic software architectures”, *FASE’98*, pp. 21-37, Lisbon, Portugal, 1998.
- [2] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, “Web Services Architecture”, *W3C Working Group Note*, 2004.

- [3] A. Colman and J. Han, "Adaptive Service-Oriented Systems: An Organisational Approach", *Computer Systems Science & Engineering*, May 2006. To appear.
- [4] Cryptography. <http://www.ssh.fi/support/cryptography>.
- [5] S. Doshi, "Software Engineering and Security: Towards Architecting Secure Software", *a graduate term paper for ICS 221 - Seminar in Software Engineering*, University of California, Irvine, 2001.
- [6] P. Devanbu and S. Stubblebine, "Software Engineering for Security: a Roadmap", *ICSE2000*, pp. 227-239, Limerick, Ireland, June 2000.
- [7] Y. Deng, J. Wang, J. J.P. Tsai, and K. Beznosov, "An Approach for Modeling and Analysis of Security System Architectures", *IEEE Transactions on Knowledge and Data Engineering*, September/October 2003.
- [8] C. Eenoo, O. Hyllooz, and K. Khan, "Addressing Non-Functional Properties in Software Architecture using ADL", *AWSA 2005*, Brisbane, March 2005.
- [9] I. Georgiadis, "Self-Organising Distributed Component Software Architectures," *PhD Thesis*, Imperial College, 2002.
- [10] Gnutella. <http://www.gnutella.com>.
- [11] IBM and Microsoft, "Security in a Web Services World: A Proposed Architecture and Roadmap", *A joint security whitepaper from IBM Corporation and Microsoft Corporation*, April 2002.
- [12] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing", *IEEE Computer*, pp41-50, January 2003.
- [13] K. Khan and J. Han, "A Security Characterisation Framework for Trustworthy Component Based Software Systems", *COMPSAC2003*, USA, November 2003.
- [14] K. Khan, "Security Characterisation and Compositional Analysis for Component-based Software Systems", *PhD thesis*, Monash University, April 2005.
- [15] T. R. De Kievit and B. H. Iglewski, "Bacterial Quorum Sensing in Pathogenic Relationships", *Infection and Immunity*, 68:4839-4849, 2000.
- [16] G. Kiczales, J. Irwin, J. Lamping, J. Loingtier, C. V. Lopes, C. Maeda, and A. Mendhekar, "Aspect-Oriented Programming", *A Position Paper from Xerox PARC*, 1997.
- [17] M. Kis, "Information Security Antipatterns in Software Requirements Engineering", *PLoP02*, 2002.
- [18] J. Mirkovic, "D-WARD: Source-End Defence Against Distributed Denial-of-Service Attacks", *PhD thesis*, UCLA, 2003.
- [19] D. Nystedt, "Online gaming growing fast in China", *IDG News Service*, April 2005. <http://www.macworld.com/news/2005/04/04/chinagaming/>
- [20] OWL. <http://w3.org/TR/owl-features/>.
- [21] OWL-S. <http://www.daml.org/services/>.
- [22] RSA Lab. <http://www.rsasecurity.com>.
- [23] C. Steel, R. Nagappan, and R. Lai, "Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management," *Prentice Hall*, 2005.
- [24] Security Patterns. <http://www.securitypatterns.org>.
- [25] M. Schumacher and U. Roedig, "Security Engineering with Patterns", *PLoP 2001*, September 2001.
- [26] SWRL. <http://www.w3.org/Submission/2004/03/>.
- [27] The HoneyNet Project & Research Alliance. <http://www.honeynet.org>.
- [28] J. Viega, J.T. Bloch, and P. Chandra, "Applying Aspect-Oriented Programming to Security", *Cutter IT Journal*, February, 2001.
- [29] A. Vorobiev and J. Han, "Specifying Dynamic Security Properties of Web Service Based Systems", *SKG2006*, China, 2006. To appear.
- [30] A. Vorobiev and J. Han, "Security Attack Ontology for Web Services", *SKG 2006 Workshop on Semantic Grid*, China, 2006. To appear.
- [31] B. De Win, "Engineering application-level security through aspect-object software development", *Ph.D. Thesis*, Department of Computer Science, K.U.Leuven, March 2004.
- [32] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for Grid Services", *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.
- [33] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network", *ICDE2003*, 2003.
- [34] R. Albert, H. Jeong, and A. Barabasi, "Error and attack tolerance of complex networks", *Nature* 406, 378-382 (2000).