Avoiding Common Security Flaws in Composed Service-Oriented Systems

Michael Atighetchi, Partha Pal, Joseph Loyall
Raytheon BBN Technologies
Cambridge, MA
{matighet,ppal,jloyall}@bbn.com

Asher Sinclair
US Air Force Research Laboratory
Rome, NY
asher.sinclair@rl.af.mil

***Abstract*— Network-centric information systems are increasingly called upon to support complex tasks and missions that serve multiple communities of interest. As a result, existing capabilities are exposed as services in a service-oriented system, and newer capabilities are derived by discovering and composing available services. While service-orientation enables and facilitates such composition-based system construction, the evolving nature and variety of standards and the varying level of compliance of otherwise feature-rich vendor products has made achieving acceptable level of security and resilience in such systems a daunting and error-prone task. This paper presents a number of factors that contribute to the security of composed service-oriented systems, and outlines ways to avoid common pitfalls and mistakes that stem from these factors and weaken the resiliency and survivability of the composed system.**

Keywords: Service Oriented Architecture, Trustworthy system design, Information Assurance, Survivability

DISTRIBUTION A. Approved for public release; distribution is unlimited. (Approval AFRL PA#88ABW-2012-1859)

This work was supported by the US Air Force Research Laboratory (AFRL) under contract FA8750-09-C-0216.

I. Introduction

There is no systematic and commonly accepted way to analyze the security and resiliency guarantees of a composed system that combines multiple functional as well as defensive services and components. Without proper support for analysis and understanding, system developers may end up constructing a system that introduces some protection, but leaves major vulnerabilities exposed; or a system that introduces new vulnerabilities due to incorrect composition; or a system that impacts performance or increases the resource footprint significantly for only a small increase in protection.

The analysis must take into account (i) the footprint and overhead of the protection services and defense mechanisms, (ii) the security and protection that they provide, and (iii) conditions such as resource constraints and threats of the targeted deployment environment. All three dimensions are important. System owners and administrators often gravitate to shiny new security tools and mechanisms without realizing that they do not fit the available resources of the target environment, and addresses only a subset of their security needs. The result is overloaded resources, significantly reduced performance, potential self denial-of-service and a false sense of security while considerable parts of the composed system remain exposed to security attacks.

Service Oriented Architecture (SOA) based systems are inherently composition-based. New functionality is expressed as orchestrations over services. While this approach is useful for quick integration and realizing new services and functionality based on well-defined workflows, it is prone to configuration errors and lacks support for systemic properties, such as security, resiliency, and efficiency.

The contribution of this paper is to identify a number of factors that contribute to the security and resiliency of composed systems. We provide goodness criteria—conditions or constraints stemming from and involving these factors that avoid the common composition pitfalls. This work serves as a precursor for an experimental approach to validate that the composed system indeed satisfies the goodness criteria.

II. Related Work

The Secure Content Automation Protocol (SCAP) [1] NIST standard specifies a multi-purpose framework for automated configuration, vulnerability and patch checking, technical control compliance activities, and security measurement. SCAP tends to express security assertions in very concrete terms and focuses mostly on the configuration state of devices, while the constraints presented in this paper are more abstract and also include interactions between multiple software components (e.g., services).

The patterns community has identified a number of security patterns [2] [3]. While similar in terms of the abstraction level of presentation, our work differs from the patterns work because we focus on adaptive composed systems and lay out the foundation for future analytical and experimental validation of constraints on software structure and interaction patterns.

III. Path to Safe & Secure Compositions

Based on our ongoing research in adaptive cyber security and advanced middleware [4,5,6] , we observe that successful engineering of a survivable SOA-based system i.e., system that combines multiple defenses or composes multiple services with different defensive capabilities, depends on at least the following factors:

A. **Coupling**: Interactions and components that need to be treated together
B. **Separation**: Aspects that need to be kept separate
C. **Ordering**: Functions that need to be performed in a certain order
D. **Keeping Secrets Secret**: Information that needs to remain hard to guess

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JUN 2012**	2. REPORT TYPE	3. DATES COVERED **00-00-2012 to 00-00-2012**

4. TITLE AND SUBTITLE **Avoiding Common Security Flaws in Composed Service-Oriented Systems**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Raytheon BBN Technologies,10 Moulton Street,Cambridge,MA,02138**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
To be presented at the 6th Workshop on Recent Advances in Intrusion Tolerance and reSilience (WRAITS 2012), Boston, MA, June 25, 2012

14. ABSTRACT
Network-centric information systems are increas-ingly called upon to support complex tasks and missions that serve multiple communities of interest. As a result, existing capabilities are exposed as services in a service-oriented system, and newer capabilities are derived by discovering and compos-ing available services. While service-orientation enables and facilitates such composition-based system construction, the evolving nature and variety of standards and the varying level of compliance of otherwise feature-rich vendor products has made achieving acceptable level of security and resilience in such systems a daunting and error-prone task. This paper pre-sents a number of factors that contribute to the security of composed service-oriented systems, and outlines ways to avoid common pitfalls and mistakes that stem from these factors and weaken the resiliency and survivability of the composed system.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**6**	

E. **Residuals**: Minimizing or at the very least, being aware of the residual vulnerabilities that get introduced when new functional and security elements are integrated into the system.

F. **Coverage**: Whether the composition covers system parts and interactions that were uncovered before (good) or adds unnecessary defenses

G. **Cost**: Adequacy of defenses given a threat model and resource environment

H. **Elasticity:** Ability of a composed system to reconfigure by shedding vulnerable defenses

Consideration of each of the above factors can lead to various configuration instantiations, some secure and some vulnerable, depending on the defenses being composed and the system context to which the composition is applied. The text boxes in each of the following subsections (A through H) represent the initial cut at formalizing the criteria for good composition stemming from the corresponding factor expressed in pseudo propositional logic. In the future, we plan to use such constraints for model checking using a lightweight formal methods tool (e.g., Alloy [7]).

A. *Coupling*

Advertised services can be coupled *physically*, i.e., when service S and T are both exposed for external consumption and both hosted on the same physical node, or *logically*, i.e., when S needs T to operate and without T, S has no value.

For coupled service interactions, protecting one service (e.g., S) and not protecting the other would leave the composed system significantly vulnerable. Consideration of this factor brings to focus the couplings that exist across multiple client-service interaction flows (see Figure 1) when designing defenses for multiple externally visible (i.e., consumed by clients) services.

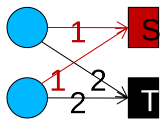

Figure 1. Coupled Interactions

As an example context where this factor influences the success/failure of the composition, consider the fact that service invocation is frequently preceded by service registration and lookup operations. If service invocation is protected by TLS, the service registration and lookup also need to be protected by TLS. Otherwise, an adversary can take over the less protected look up /discovery service and insert itself as a man in the middle in all service registration and look up interactions, which in turn, will break confidentiality and integrity of the TLS protected service.

Another example involves availability. If a service is highly available, i.e., uses redundancy and other availability mechanisms to support high invocation load and remain available under multiple failures, the registry and lookup interactions must also offer comparable level of availability.

The *goodness* criteria for the class of compositions influenced by this factor can be formalized as a constraint over *business information flows* (or *flows* in short) based on the following definitions:

Def. 1: A *flow* is defined as a triple <srcIP, dstIP, dstPort> representing a TCP connection between a client at srcIP and a service offered at dstIP, dstPort. Flows can have associated properties. We are initially considering the following four properties:

- *Encrypted:* a flow is encrypted if the information it carries cannot be observed by a network sniffer.
- *Signed:* a flow is signed if the recipient can detect content tampering in transit.
- *Redundant:* if the service offered by (dstIP, dstPort) of flow x is offered by a different (dstIP, dstPort) pair of at least one other flow y.
- *Diverse:* if the offered services of two redundant flows have different host or AS implementation signature

Def. 2: Two flows x and y are *coupled* if they have the same dstIP, **or** the client's need for service consumed in flow x cannot be fulfilled if flow y is not available, **or** the client's need for service consumed in flow y cannot be fulfilled if flow x is not available.

For a set of coupled flows, all member flows have comparable levels of property p where p is an element of {encrypted, signed, redundant, diverse}.

At the most basic level, if one member flow x in a set of coupled flows is encrypted [signed or redundant or diverse], and another member is not encrypted [signed or redundant or diverse], the composed system has a security risk.

B. *Separation*

In contrast to the previous influencing factor, this factor draws attention to system elements, i.e., components and capabilities, that must be treated separately, i.e., tight coupling or common protection will result in reduced effectiveness and increased security risk. Figure 2 visualizes this factor.

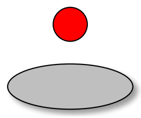

Figure 2. Separation

As a specific instance of composition influenced by this factor, consider the fact that defense mechanisms essentially protect either the information that is being processed and managed by the information system being defended (i.e., as defined earlier, flows) or the business components that process/manage such data. Some defense mechanisms handle the flows inline while others watch over, analyze the behavior of, and manage the life cycle of the business components. The command and control interaction with these defense mechanisms i.e., the reports they issue and the control signal they receive, often share the same path without proper isolation among the *data* and *control*. A good composition will establish isolated paths for control and data messages, so that data load cannot be used to impede the control. The extent of the isolation needs to be commensurate within the cost/resource parameters of the deployment context.

Monitoring a function or a component provides a second instance of composition influenced by this factor. Monitoring is frequently used for security, fault tolerance and system management. In all cases, a level of independence between the *monitor* and the *monitored* must exist in order for the composed system to function as expected. The monitoring mechanism, such as a watchdog [8], and the monitored component such as a proxy, must not be in the same process, and preferably not on the same host; otherwise failure of the monitored component may blind monitoring.

Another example arises in redundancy management. Replicas of components are frequently used for fault and intrusion tolerance. Placing multiple replicas in the same process, in the same virtual machine (VM), or on the same host, is less effective than placing them separately because otherwise a process crash, VM crash, or host crash would take out both replicas. The goodness criterion stemming from this factor can be stated as follows:

If X is a defense mechanism protecting Y, where Y can be a system component or an information flow, then disruption in Y should not cause disruption in X.

C. *Ordering*

It is not uncommon for a survivable system to subject information flowing through it to multiple checks at various inspection points. In many cases, the checks are at different system layers, but it may happen that multiple checks are performed at the same abstraction layer in the same inspection point. For instance, multiple application level checks such as request-rate limiting, request-size checking, and input validation can be performed by a set of proxies operating either in series or in parallel. In such cases, the ordering of the checks (as shown in Figure 3) can make the composition effective or ineffective.

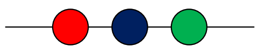
Figure 3. Ordering

Intuitively, it might appear that a strategy like *biggest differentiator first* or *most expensive check last* is the right way to compose the different checks. However, we recommend a strategy that organizes the checks based on how deeply the check needs to interpret or process the information. For example, putting the input-validation check before the request-rate limiting check may make the rate-limiting defense ineffective because input validation involves quite a bit of parsing and interpretation of the request, and can be bogged down by a request flood. Similarly, putting a request-size check before a request-rate check will make the defense against request flood ineffective: the size check involves a significant amount of interpretation and processing of the requests (though less than input validation). The goodness criterion resulting from this factor is stated below:

If two defense mechanisms X and Y are in line with a business information flow and are collocated (in the same process or node), then X should process the flow before Y only if X is at a lower abstraction level (i.e., does not need to parse, process or interpret the flow as high as Y).

D. *Keeping Secrets Secret*

A number of security techniques rely on keeping some designated piece of information secret: if the adversary can find out or guess such information, the security protection provided by these mechanisms will be bypassed. Passwords provide one example, if passwords are discovered or can be guessed, a password-based access control/authentication mechanisms offer no protection. Similarly, in the context of encryption, the decryption key should not be easily guessable or easily discovered. However, in composition-oriented system construction, it is not uncommon to face the need to transmit, share or store passwords and key materials. Storing and transmitting such material in clear text, choosing passwords that are easy to guess, storing clear text passwords in obvious locations are examples of bad system configurations that should be avoided and are easily overlooked.

Influence of this factor however, is not limited to passwords and key material. One of the essential features of a survivable system is adaptability or its ability to mount adaptive defensive response. But merely adding multiple types of adaptive features or even organizing them in defense-in-depth layers may not readily provide the expected effectiveness. What is often overlooked in the design of adaptive system is its predictability to an adversarial observer. The kind of adversary DoD systems are facing are patient and well-motivated, and are willing to invest significant amount of time and resources passively observing the target system or observing how the system responds under gentle fuzzing and probing without drawing significant attention. If the adversary can predict how the system is going to respond, he can pre-plan and position himself at the right location and with the right counter-response. In other words, predictable responses are indicative of an ill-composed system, and make it easier for attackers to plan and execute multi-stage attacks. A specific example of predictable response is when a replica management strategy restarts replicas in well-known places.

While we advocate making adaptive responses unpredictable, this variability comes at a cost. Legitimate clients need to be signaled to adapt to the new configuration, and it is hard to gain a statistically significant amount of "randomness" or "unpredictability". The goodness criterion for this factor can be summarized as:

If defense mechanism M performs a specific action A, where A could be a dynamic reconfiguration of the system, an access control decision or a transformation of data, the trigger for A (i.e., exactly what condition causes the composed system to mount the specific action A) should be hard to guess without significant access and privilege in the system.

E. *Residuals*

Adding new components to a system, even when the new components perform security functions, introduces new risks because changing the system may modify the system's attack surface and vulnerability profile and these changes could potentially be for the worse (as visualized by the red arrow in Figure 4). To illustrate the issue, consider an application firewall introduced to defend services hosted on a Web server. But the application firewall itself now becomes a new attack target. Success of the survivability architecture in composing the defenses with the defended components depends on whether the attack surface and vulnerability profile of the composed system is changed in favor of the defense. Intuitively, a good composition implies that the composed system has a reduced attack surface and smaller vulnerability profile than the individual components being composed. However, we need a few definitions

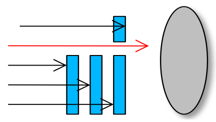
Figure 4. Residuals

first in order to formalize the goodness criteria for this factor.

Def. 3: A defense mechanism X *covers* a component Y if (a) X pre- or post-processes the same information that is processed by Y or (b) X monitors Y.

Def. 4: X *reduces* Y's attack surface if (a) X is a pre-processor and all ingress to Y is through X, or (b) X is a post-processor and all egress from Y is through X, or (c) X detects or responds to failures in Y.

Def. 5: The composition of X and Y has a *reduced attack surface* if X reduces Y's attack surface and X has self protection and no unprotected access.

Def. 6: The composition of X and Y has a *smaller vulnerability profile* than Y if (a) there are attacks that succeed against Y alone, but not against the composed system, and (b) all attacks that succeed against the composed system also succeed against Y alone.

Given these definitions, the goodness criterion for this factor can be stated as:

If defense mechanism X covers a component Y then X composed Y has reduced attack surface and smaller vulnerability profile.

As an illustration, let us consider a simple control loop that composes a mechanism Y looking for attack signatures in single IP packets with a mechanism X blocking connections from a specified source. X will block the source IP I if Y has encountered bad packets from I. The residual vulnerability in this case is that source IPs can be spoofed (e.g., rewriting the source field in TCP SYN packets), and by spoofing IP addresses (i.e., making them appear from legitimate client IP addresses), an adversary can use the control loop to get those client's IP addresses blocked. It is also possible that a corrupt mechanism X can arbitrarily start blocking IP addresses. The composition instance violates the goodness criteria because the composition of X and Y resulted in a wider attack surface (with the possibility of manipulating firewalls), and does not reduce the vulnerability profile (neither X nor Y addresses spoofing). This concern can be addressed, for example, by composing the OODA loop with IPsec protections on the network layer, which can provide anti-spoofing guarantees if configured correctly. Another mitigation approach involves observing packets over time, which makes spoofing more difficult for the attacker (since spoofed responses don't get back to the attacker machine).

F. *Coverage*

The idea behind this factor is to capture the intuitive understanding that defenses need to match the anticipated threats. It is quite common to introduce the latest security technique or tool into the system without a clear understanding of the purpose, mode of operation, and security benefits of the newly introduced mechanism. For example, inserting SQL injection protection into a service enclave that does not use any SQL database is an instance of a bad composition. The unnecessary check not only adds to the memory and performance footprint, but also increases system complexity and therefore unnecessarily increases the attack surface. Figure 5 tries to visualize two situations that are pathological. At the top, a small threat is defended against by a lot of defenses. At the bottom, a large threat is inadequately covered by only one defense.

Formalizing the goodness criteria for this factor assumes that defenses are profiled and annotated, and a threat analysis has been performed on the given system.

Def 7: We define *coverage* of a composed system as the ratio of the number of threats being mitigated by at least one component of the composed system to the total number of threats to the system identified in the threat analysis.

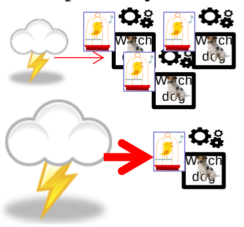
Figure 5. Threat & Coverage

Initially, the undefended system starts with a low coverage score, and as new defenses are added, the coverage score of the composed system increases. The goodness criterion then can be stated as:

Composition of X and Y has coverage greater than the individual coverage of either X or Y.

G. *Cost*

No defense is absolute, and it is understood that we can only strive for an *adequate* level of security. Threat coverage as explained in the previous section is one aspect of adequacy, and the other aspect has to do with cost. While it is easy to proclaim or require that defenses be commensurate with available resources, the practice of defense-enabling a given system faces a number of obstacles and often results in costly and inefficient solutions. The objective of this influencing factor is to draw attention to the cost of composing multiple defenses and help guide designers and system developers towards a more appropriate and more efficient solution.

In the context of evaluating the cost of composed systems, we are restricting ourselves to resource and performance cost. Considerations include the initial upfront cost of additional resource requirements, and runtime operating cost in terms of performance penalty and resource usage.

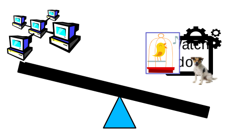
Figure 6. Cost & Performance

An example of a bad composition includes deploying a sophisticated event interpretation technology that requires multiple dedicated hosts and still requires minutes to complete in a tactical environment. First, the requirement of additional nodes puts undue (and possibly unachievable) pressure on the deployment context. In addition, the response time achieved does not match with the operational tempo of the deployment context. While accuracy might be very high, the delay imposed by the analysis will likely reduce effectiveness against attacks that spread quickly. Instead, a better choice might involve the use of *anytime* algorithms that can compute some answers early on and refine the results if more time is available.

Assuming that the defenses are annotated with resource costs and that the resource restrictions of the deployment context are documented, the goodness criteria for this factor can be formalized as:

If defense mechanism X covers component Y, (a) X's static resource requirements can be accommodated in the deployment context, and (b) the processing and communication delay introduced by Y does not increase the response time of Y beyond the expected optempo of the deployment context (expected response time for business requests, expected response time for defensive actions).

H. *Elasticity*

In the context of cyber-attacks and survival, the ability to shed defenses if necessary (i.e., decomposition), often becomes as critical as composition. If an attack cripples a defense mechanism or component in such a way that the system degrades beyond acceptable levels, it may be better to run without the protection of that defense mechanism. For instance, imagine a zero-day exploit kills or corrupts the single packet authorization (SPA) mechanism that sits at the boundary of the protected service and does not allow any incoming request unless the request provides a valid cryptographic credential. Neither restarting the SPA mechanism nor giving up (i.e., stopping the system) is a good option, but shedding the SPA—i.e., running the system without the SPA would perhaps be acceptable. Similar situation may arise when a critical security advisory is issued, and the system administrators need to balance the risk of running with a highly vulnerable defense and not having that defense. However, it is not always easy or simple to do so in a system that integrates multiple defenses. We introduce the notion of composition elasticity to describe the characteristics of systems where such shedding is possible. In other words, shedding is harder to do in a non-elastic composed system.

Although achieving elasticity is a design and implementation task, it is easy to identify the factors that make the composed system non-elastic. The two major factors that contribute to elasticity are coupling and interface preserving insertions. If two defensive mechanisms are coupled i.e., depend on each other, then it is harder to remove either of them. If the degree of coupling is large, i.e., B is coupled with A1, A2, … An, it becomes even harder. Therefore one goodness criterion stemming from elasticity is:

Elasticity of the composition of X with a system S goes down as the coupling between X and S increases.

Another important point to consider in this context is that much of survivability- and security-focused computation is done on intercepted information flow. In a way, that is akin to subjecting the undefended information flow through a number of security focused layers. If care is not taken to preserve the interfaces of the original communicating party, shedding the defense mechanism becomes more difficult. On the other hand, if the defense X was inserted in an interface preserving manner, when X becomes corrupt or unusable, it could be substituted by another interface compliant variant X1 of the defense mechanism X, which in turn, will force the adversary to start from scratch again against X1. This leads to the other criterion stemming from this factor:

Composition of service X with service Y is more elastic if X and Y have the same interface.

IV. Red Team Evaluation

The factors influencing the safety and security of composed SOA-based systems and the goodness criteria described in this paper are based on project that developed a prototype survivable SOA-based system [4]. The prototype survivable system was subjected to a variety of attacks during a week-long Red Team Exercise [9] at AFRL Rome NY. Apart from evaluating the prototype survivable system's ability to withstand attacks, the exercise also aimed to evaluate the quality of the compositions involved in the construction of the prototype system. The exercise was based on a collaborative testing paradigm in which the independent Red Team was given complete access to source code and an escalating level of privilege by way of different attacker starting points. While a detailed discussion of the results is beyond the scope of this paper (they will be available in a forthcoming paper), we note that the prototype survivable system exhibited significant resilience and adaptability at multiple layers in the red team exercise. In the remainder of this section, we highlight a number of cases in which the factors described in this paper provided direct benefits.

We improved the architecture's *elasticity* by defining several configurations (e.g., with SPA and without SPA, with adaptive response and without) that shed different defenses. In the process of standing up and verifying these configurations, we minimized unnecessary coupling in the system. Also, we achieved *redundancy* and *diversity* (both part of coupling dependency) by standing up two Termination Proxy nodes in the Crumple Zone with different TLS-stream splitting methods. To achieve *separation*, we confined traffic to isolated network zones, resulting in multiple Ethernet and IP networks. Furthermore, we used process boundaries to enforce separation between Termination Proxies that provide endpoints for client interactions and Mechanism Proxy Neighborhoods that perform filtering on application traffic. We manually specified and analyzed *ordering* of firewall rules, e.g., to ensure that single packet authorization checks happen before rate limiting. In addition, we specified orders of application-level checks, e.g., mandating rate limiting before white list checks before simulated execution in canary proxies. To establish *coverage* arguments, we analyzed the network flows for defense-in-depth coverage asserting that flows are constrained by at least one of Netfilter [10], JVM security policies [11], or SELinux [12] policies. Furthermore, SPA and checks running in the crumple zone's Mechanism Proxy Cloud (MPC) added additional protections. Figure 7 shows the resulting defense in

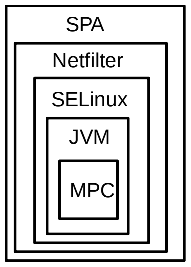

Figure 7. Defense-In-Depth

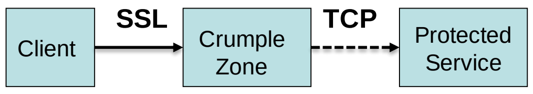

Figure 8. Coupling Violation

depth layout of multiple policy enforcement mechanisms working together.

We also built up models of observed network flows by experimentally determining the type of flow by running regular expressions against network capture files. Analysis of these models for *coupling* constraints revealed that flow from the Crumple Zones to the protected service were not protected by TLS, while flows from the client to the Crumple Zone were protected by TLS. Figure 8 shows a picture of the ordering violation found through analysis of network flows during the Red Team exercise.

V. Conclusion and Next Steps

Effective cyber defense today requires multiple individual mechanisms selected and configured according to specific threats in a specific environment to work together synergistically. The main contribution of this paper is to describe constraints for good compositions, which provides two benefits. First, the factors and the goodness criteria provide the system engineers with guidance for composition orient system construction. Second, the formalization of the criteria using a propositional logic like pseudo-language paves the way for automated analysis and assessment of composed systems in the future.

Software developers can use the constraints described in this paper to assess the architecture and design of distributed systems composed of business logic and security function to discuss design tradeoffs and document decisions, thereby increasing the security of resulting systems.

Formalization of goodness criteria is clearly a first step in the long road to automated model checking for safe and secure composition. Software engineering and system construction techniques are rapidly evolving. New defense mechanisms as well as new attacks are constantly emerging. However, in the rapidly changing landscape, the structural nature of the goodness criteria involving the basic and fundamental aspects of distributed system interactions is a step in the right direction.

Starting with the goodness criteria at design time, it is possible to derive interaction patterns or conditions that can be checked at runtime. A specific next step we are planning to explore is to use domain-specific languages to (a) succinctly describe composed survivable systems, (b) debug described compositions through model checking techniques, e.g., using Alloy [7], and (c) generate lower-level mandatory access control policies, e.g., for SELinux or Netfilter, once the described composition is conflict free.

References

[1] NIST. (2011, June) Security Content Automation Protocol. [Online]. http://scap.nist.gov/

[2] Joseph Yoder and Jeffrey Barcalow, "Architectural Patterns for Enabling Application Security," in *PLoP*, 1997.

[3] Markus Schumacher, *Security Patterns: Integrating Security and Systems Engineering.*: Wiley, 2005.

[4] Michael Atighetchi et al., "Crumple Zones: Absorbing Attack Effects Before They Become a Problem," in *CrossTalk - The Journal Of Defense Software Engineering*, March/April 2011.

[5] Paul Benjamin, Partha Pal, Franklin Webber, and Paul Rubel, "Using A Cognitive Architecture to Automate Cyberdefense Reasoning," in *Proceedings of the 2008 ECSIS Symposium on Bio-inspired, Learning, and Intelligent Systems for Security(BLISS 2008)*, Edinburgh, Scotland, 2008.

[6] J. Chong, P. Pal, M. Atighetchi, P. Rubel, and F. Webber, "Survivability Architecture of a Mission Critical System: The DPASA Example," in *Proceedings of the 21st Annual Computer Security Applications Conference*, 2005, pp. 495-504.

[7] MIT. (2010, September) Alloy Community Web Site. [Online]. http://alloy.mit.edu/community/

[8] (2012, Mar) Wikipedia: Watchdog Timer. [Online]. http://en.wikipedia.org/wiki/Watchdog_timer

[9] B.J Wood and R.A Duggan, "Red Teaming of advanced information assurance concepts ," in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00*, Hilton Head, SC, 2000, pp. 112 - 118.

[10] Netfilter.org. (2010, June) Netfilter home page. [Online]. http://www.netfilter.org/

[11] Peter V. Mikhalenko. (2007, April) Java security: Policies and permission management. [Online]. http://www.techrepublic.com/article/java-security-policies-and-permission-management/6178805

[12] NSA. (2012, Jan) SELinux FAQ. [Online]. http://www.nsa.gov/research/selinux/faqs.shtml