A Game of Microservices: Automated Intrusion Response

Tetiana Yarygina, Christian Otterstad

A Game of Microservices: Automated Intrusion Response

Tetiana Yarygina and Christian Otterstad

Department of Informatics, University of Bergen, Norway
tetiana.yarygina@uib.no
christian.otterstad@uib.no

Abstract. The microservice architecture is a subtype of distributed systems that has been attracting a lot of interest both in the industry and academia. Emerging research recognizes the need for a better understanding of microservice security, and, in particular, mechanisms that enable defence-in-depth and adaptive security. With the continuously growing number of possible attacks and defenses, the choice in the optimal defense strategy becomes non-trivial as well as time critical. We propose a cost-sensitive adaptable intrusion response system for microservices, which uses a game theoretic approach to automatically respond to network attacks in real time. Furthermore, we discuss both the applicable attacks and defense responses specific to microservices.

Keywords: adaptive security, self-protection, game theory, defense-in-depth, SOA, IPS, IDS, minimax

1 Introduction

Microservice architecture is gaining significant attention both by practitioners and in academia [1, 2]. Microservices allow for building flexible systems where the components can be written in different programming languages, use different technologies, scale independently, and can be easily updated and redeployed. Many microservice architectural principles such as modularity, loose coupling, and fail-fast are not new and stem from fundamentals of distributed systems [3]. Microservices are a particular implementation approach to Service-Oriented Architecture (SOA) [4]. However, the scale of microservice adoption is unprecedented and can perhaps be compared with the invention of object-oriented programming (OOP).

A key aspect of microservices is automation. With hundreds or thousands of microservices, manual updates are infeasible. Centralized logging, performance monitoring, service discovery are examples of such automation that are ubiquitously adopted. Such trends as contentious integration, DevOps culture, and need for high scalability and flexibility further increase the importance of automation in microservice networks. Yet, not all the areas of microservice operation are automated.

Self-protection and other self-* properties, such as self-configuration and self-optimization that allow software systems to adapt to the changes in the environment, are actively researched areas [5]. However, the problem of self-protection for microservices has received scant attention in the research literature.

Regarding microservice security, Fetzer [6] discussed deployment of microservices inside secure containers to build critical systems. Sun et al. [7] proposed a virtual network monitoring solution that enforces policies over the network traffic in the cloud. Otterstad & Yarygina [8] pointed out the isolation benefits of microservices, as well as proposed the use of N-version programming for a simplified IDS. Yarygina & Bagge [9] have investigated automation of secure inter-service communication and defense-in-depth principle for microservices.

Intrusion detection systems (IDSs) provide system administrators with information on malicious activity in the system. While intrusion prevention systems (IPSs) attempt to block intrusions that are detected, the handling of complex situations and choice of intrusion responses are often left to humans. Once an intrusion is detected, actions should be taken as fast as possible to stop an attack.

According to Stakhanova et al. [10], an ideal intrusion response system (IRS) should be automatic, proactive, adaptable, and cost-sensitive. One of the possible ways of achieving such an IRS is through security games. Game theory [11] studies mathematical models of how multiple agents act when optimizing their payoffs. While the conventional game theory has a variety of applications in economics, political science, biology, it also received significant attention in the area of network security [12–14].

There is a general lack of research in IRS for microservices. With the continuously growing number of possible attacks and defenses, the choice of the best defense strategy is complicated. A game theoretic approach potentially solves this problem. Our main contribution in this paper is the design of a cost-sensitive automatic IRS for microservices with game-theoretic foundation; we also elaborate on the response actions specific to microservice architecture.

This paper is organized as follows. Section 2 explains the game theory fundamentals and defines the game model used in this paper. In Section 3, the architecture of the proposed IRS is presented. Section 4 evaluates the proposed architecture. Section 5 concludes the paper.

2 Security Games: Assumptions and Solutions

We observe that the microservice architecture readily allows for employing game theory derived algorithms. This is a foundation for the system ability to respond to intrusions. This paper models the strategic interaction between an attacker and a defender as a finite dynamic zero-sum game. Different game theoretic solution concepts exist [11].

2.1 Finite Dynamic Two-player Zero-sum Game

Most security games involve two counteracting parties—an attacker and a defender. While attacker's goal is to exploit the system, a defender is trying to

protect the system, its resources, and data. Each player has a set of actions available to them at given time. Each action has a positive or negative reward associated with it. In the case of attacker and defender, each player's gain or loss is balanced by the losses or gains of the other player, which makes this game *zero-sum*. A compromised microservice node has a negative score for the defender, but a positive score for the attacker.

The attacker and defender take actions and receive rewards in turns, as seen in Figure 1. In this way, the game moves from one security state to the other. Games with more than one stage are called *dynamic* or *extensive*.

There is a limited number of states a given system can be in. Some of the states are the final states, i.e. the leaf nodes on the game graph. The security states where the attacker gains full control of the system can be seen as final states. In such case, the game is called *finite*.

2.2 Minimax

The goal for the defender is to choose the optimal response action in the given context. A common solution to this problem is the minimax algorithm. The minimax strategy for a defender is a strategy that minimizes the maximum payoff of an attacker, i.e. maximizes the benefit of a defender. For two-player finite zero-sum games, the minimax solution is the same as the Nash equilibrium.

In a two-player game, the minimax strategy for a defender i against an attacker $-i$ is $\arg\ \min_{a_i \in A_i} \max_{a_{-i} \in A_{-i}} u_d(a_i, a_{-i})$, where a_i is the response action taken by defender, a_{-i} denotes the attack action taken by adversary, and u_d is the utility function of defender.

Completely analyzing certain games using solely the minimax algorithm is impractical. The minimax algorithm traverses the nodes of a game tree. The number of nodes on the tree increases exponentially with the number of turns that can lead to a combinatorial explosion. The performance of the minimax algorithm can be improved using alpha-beta pruning or other pruning methods.

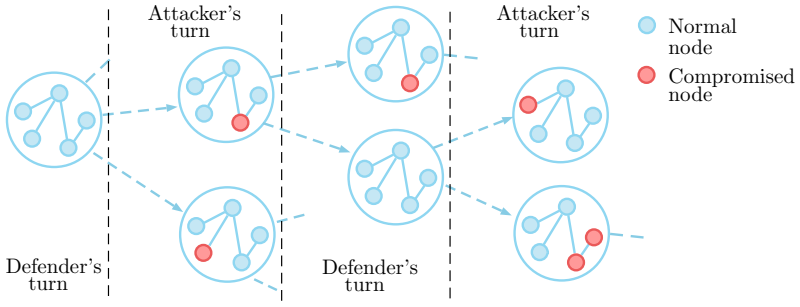

Fig. 1: Security game between the defender and attacker. An attacker decides which microservice to attack. The response actions deployed by the defender may or may not eradicate the attacker. If defense measures are insufficient and/or unsuccessful, the attacker will propagate laterally through the network.

3 Proposed Architecture

Building on this game theoretic foundation, we propose a microservice intrusion response system. The system is cost-sensitive, in that the game allows us to chose the most effective, least costly response to an attack, rather than applying drastic measures in every situation.

The system consists of a distributed set of local monitoring processes and a central entity called the Microservice Game Engine (μGE), as depicted in Figure 2. The purpose of the system is to minimize the damage caused by an attack in real time. The μGE allows the microservice network to dynamically react to threats while taking action costs into consideration. In particular, the μGE exploits the fact that a partially compromised microservice network has not yielded total control to the attacker. A strong separation of the control flow and increased isolation are inherent benefits of microservice architectures [8, 9].

The μGE aggregates information, builds a game tree and takes automated action based on the observed input obtained from the local IDSes running on the respective microservices. In order to facilitate these actions, the μGE relies on several key components discussed below.

3.1 Network Mapping

The ability to maintain an accurate view of the microservice network at all times is a prerequisite for the μGE to respond effectively to malicious activity. The initial microservice network can be mapped in several ways. For example, each microservice can report its incoming and outgoing edges to the μGE.

After the network has been initially built by the μGE, it is ready to start playing the game in anticipation of attacks. However, when the real network changes, its representation inside the μGE should also be updated, and all computation performed thus far will be discarded. Nodes are inserted and removed from the tree as they report to the μGE.

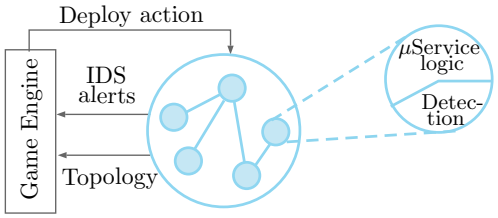

Fig. 2: An overview of the proposed architecture. The detection code in each microservice reports to the game engine, which evaluates the current state of the system and plans a response to any ongoing attack. Responses are deployed by e.g., restarting or reinstalling services, or other response actions.

3.2 Intrusion Detection and Events Reporting

Intrusion detection functionality resides in each microservice, informing the μGE of events of interest. Multiple sources of IDS information, as well as non-IDS information may be used. An IDS such as Snort [15] can detect among other things stealth port scans, operating system fingerprinting, and buffer overflows. Non-IDS information include events in the network, such as a service having stopped sending heartbeats, service registration and deregistration.

Attack actions in the game can be defined through the intrusion detection function. The set of possible attack actions is limited to the knowledge base of the particular IDS in place.

3.3 Event Evaluation Function

The evaluation function takes as its argument a node object representing a state of the microservice network. The state of this network in terms of score is then evaluated and returned as an integer. The most basic and coarse node states may be grouped into three categories. *Benign:* default normal state of all nodes. *Under attack:* an alert for the node has been raised. *Compromised:* a node that conducts malicious activity or is suspected of one.

The μGE can notice that events have happened over time, and/or that events are happening in multiple places of the network at the same time—each event which by itself is not enough to trigger an issue, but as an aggregated result. The aggregated result may be accumulated in a temporal and/or spatial sense. This is similar to what distributed IDSs would do, see the survey by Folino [16].

The evaluation function should be able to aggregate information such that a node can be inferred to be compromised based on its behavior. E.g. if there is a port scan or API probing attempt from an internal node, this is assumed to only be possible if the node is compromised. The node may therefore be flagged as compromised even though no direct detection of an attacker present on the system was detected. If there is only one other node which could communicate with the compromised node, the evaluation function can further infer that this node is also compromised. By extension, the μGE can infer a chain of guaranteed compromised nodes and possibly compromised nodes. As an example, if there are two additional nodes which can communicate with the node, the evaluation function can trivially assume with 50% probability that either of them are compromised, assuming everything else is equal.

3.4 Decision Function

The decision function runs a minimax algorithm with specific pruning mechanisms. If there is no actual malicious action and no network related update taking place, the μGE will populate the tree representation of different possible states. For each particular node of the microservice network, based on the state of the network, there will be a list of possible actions the attacker or defender can perform. The set of possible actions is used to create new states of the same

microservice network, creating new nodes, where the edges from the previous node represent the particular action taken.

Whenever any new information which results in a different state is received, the μGE stops its search, and updates the root of the tree to be that resulting state. This simulates performing the actual operation the attacker did against the real network. It is now the defender's turn. The μGE will also run the evaluation function on all the leaf nodes and compute the optimal move for the defender. We make the conjecture that the time taken to actually run this part of the algorithm is negligible and can be run post attack. The best strategy is chosen based on the available response actions discussed below.

3.5 Intrusion Response: Defender's Actions

Traditional fault tolerance techniques include rollback, rollforward, isolation, reconfiguration, and reinitialization [17]. Microservice based systems, however, allow more actions to be taken. In the case of an assumed compromised node, the defender may opt for the following choices.

Rollback/restart the service. This will destroy the current instance of the service and start a new one from the same configuration. If the problem persists, even older configuration of the service may be used. This allows the defender to hopefully mitigate attacks based on flawed configuration assuming it was ever correct, as well as bugs introduced in the latest version.

Diversification through recompilation or binary rewriting. Introducing randomness into the binaries executing in the microservice may be done by binary rewriting or recompilation with special compiler support [18]. An example of a freely available framework providing such support is the LLVM multicompiler.

Diversification through cloud provider. Moving a microservice to a different host in attempts to mitigate attacks that rely on host characteristics, such as exploits that target hardware or a malicious cloud provider.

Scale up and down n-variant services. The N-variant microservice system was proposed as a security measure by Otterstad&Yarygina [8]. This action uses the existing diversification techniques (compiler diversity/binary rewriting and cloud diversity) to spawn additional microservices, which feed their result to a governor node that compare the results for consistency. This allows nodes that have been tampered with to be detected.

Split or merge services. Requires a tool support for code auto modification that does not currently exist. An extension of this approach is to add dummy hardened services to the path. A node may be split at the function level, this may mitigate certain attacks that rely on there being a binary path of execution which enables the exploit to work, and/or the existence of certain gadgets, which will not be available after a split has been performed.

Isolation/Shutdown. Entails physical exclusion of the faulty service: stop the service permanently. This approach has a high cost associated with it and is unacceptable for the systems with high availability requirements. Noureddine et.al. [14] showed that disconnecting nodes in a response to an attack efficiently delays an attacker that is moving laterally in the simulated network.

4 Evaluation and Discussion

Advantages. The controller system has two main advantages: low latency, and depth. In contrast to a human, who may need time to understand and react to the attack, the μGE can react instantly. The latency is important due to the fact that some attacks may be automated and complete a sequence of steps in the attack very quickly, which would make it critical to be able to respond promptly.

The depth is important for related reasons. The μGE can search deeper into the tree and gain a deeper insight than a casual observation from a human. Some choices may not seem intuitive as they result in a better network state deeper down in the tree. It may be possible that the attacker performs a particular attack to which the naive reaction is what allows the actual attack to proceed.

Algorithm complexity. The depth of the tree is m with b legal moves on average (the branching factor of the tree). The running time of the minimax algorithm is $O(b^m)$, and space complexity (memory) is $O(b*m)$. For large trees, high complexity can make the approach computationally infeasible. For the alpha-beta pruning algorithm, sorting the moves by the result improves performance, such that for the perfect ordering the running time is $O(b^{m/2})$.

Model limitations. So far, we discussed only a subset of all possible attack and defense actions. In a real world scenario there are not only more nodes, but many more operations the attacker and defender could do. This causes an explosion in the complexity of the tree, which greatly limits the depth of the search, consumes much more memory, and CPU time.

The list of attacker and defender operations is a model of the real world. Any such model will have limitations in terms of granularity. The extent to which the defender is willing to exert resources on creating an accurate representation of the real world will affect the effectiveness of the system. However, even for missing classes of attacks, they are likely to result in a state which the system will detect. Lets consider a node compromised with a zero-day exploit that went unnoticed. It is extremely unlikely the attacker has a zero-day for every node in the network. Thus, when the attacker starts to probe the rest of the network from an internal node, looking for well-known vulnerabilities, the μGE will again notice the issue and can consult its graph for the optimal course of action.

This paper assumes that the utility function and reward values are designed by experts offline. Selecting the best defense model is difficult because of a lack of quantifiable security metrics. Despite the multiple attempts to address this problem [19], putting values to parameters is still a human responsibility.

5 Conclusions

This paper presented the design of a cost-sensitive adaptable IRS applicable to microservice networks called μGE. The μGE collects information from the network as well as issues actions based on a search tree of possible outcomes once an attack has been detected. The proposed solution exploits the fact that the microservice network is modular by design and components can be restarted,

permuted, moved, and even in some cases removed, without destroying the entire operation of the network. In general, no mitigation technique provides a guarantee an attacker cannot succeed. However, the μGE enables low latency and far lookahead which is a strong advantage for a defender.

Several open questions remain. An efficient approach to identifying and setting the values that are topical to each defender has not been presented, as this is highly subjective and specific to the assets that are important. Furthermore, for an algorithm of this type to be efficient on big networks, it is likely that a significant amount of aggressive pruning of the search tree must be performed.

References

1. Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J., Josuttis, N.: Microservices in practice: Reality check and service design. IEEE Software 34, 91–98 (2017)
2. Newman, S.: Building Microservices. O'Reilly Media (2015)
3. Tanenbaum, A., van Steen, M.: Distributed Systems: Principles and Paradigms. Pearson Prentice Hall (2007)
4. Zimmermann, O.: Microservices tenets: Agile approach to service development and deployment. Computer Science - Research and Development pp. 1–10 (2016)
5. Yuan, E., Esfahani, N., Malek, S.: A systematic survey of self-protecting software systems. ACM Trans. Auton. Adapt. Syst. 8(4), 17:1–17:41 (Jan 2014)
6. Fetzer, C.: Building critical applications using microservices. IEEE Security & Privacy 14(6), 86–89 (Nov 2016)
7. Sun, Y., Nanda, S., Jaeger, T.: Security-as-a-service for microservices-based cloud applications. In: CloudCom. pp. 50–57. IEEE (2015)
8. Otterstad, C., Yarygina, T.: Low-level exploitation mitigation by diverse microservices. In: ESOCC. pp. 49–56. Springer, Cham (2017)
9. Yarygina, T., Bagge, A.H.: Overcoming security challenges in microservice architectures. In: Service-Oriented System Engineering (SOSE'18). IEEE (Mar 2018)
10. Stakhanova, N., Basu, S., Wong, J.: A taxonomy of intrusion response systems. International Journal of Information and Computer Security 1(1-2), 169–184 (2007)
11. Osborne, M., Rubinstein, A.: A Course in Game Theory. MIT Press (1994)
12. Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., Wu, Q.: A survey of game theory as applied to network security. In: HICSS. pp. 1–10. IEEE (2010)
13. Zonouz, S.A., Khurana, H., Sanders, W.H., Yardley, T.M.: RRE: A game-theoretic intrusion response and recovery engine. IEEE TPDS 25(2), 395–406 (2014)
14. Noureddine, M.A., Fawaz, A., Sanders, W.H., Başar, T.: A game-theoretic approach to respond to attacker lateral movement. In: Zhu, Q., Alpcan, T., Panaousis, E., Tambe, M., Casey, W. (eds.) GameSec. pp. 294–313. Springer, Cham (2016)
15. Snort official web-site, `www.snort.org`, [Accessed Feb. 23, 2018]
16. Folino, G., Sabatino, P.: Ensemble based collaborative and distributed intrusion detection systems. J. Netw. Comput. Appl. 66(C), 1–16 (May 2016)
17. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE TDSC 1(1), 11–33 (2004)
18. Jackson, T., Salamat, B., Homescu, A., Manivannan, K., Wagner, G., Gal, A., Brunthaler, S., Wimmer, C., Franz, M.: Compiler-Generated Software Diversity, pp. 77–98. Springer New York, New York, NY (2011)
19. Simmons, C.B., Shiva, S.G., Bedi, H.S., Shandilya, V.: ADAPT: a game inspired attack-defense and performance metric taxonomy. In: IFIP International Information Security Conference. pp. 344–365. Springer (2013)