An Architectural Design to Address the Impact of Adaptations on Intrusion Detection Systems

Ian Riley
Tandy School of Computer Science,
University of Tulsa,
Tulsa, Oklahoma, USA 74104
ian-riley@utulsa.edu

Allen Marshall
Tandy School of Computer Science,
University of Tulsa,
Tulsa, Oklahoma, USA 74104
allen-marshall@utulsa.edu

Logan Quirk
Tandy School of Computer Science,
University of Tulsa,
Tulsa, Oklahoma, USA 74104
ldq1403@utulsa.edu

Rose Gamble
Tandy School of Computer Science,
University of Tulsa,
Tulsa, Oklahoma, USA 74104
gamble@utulsa.edu

Abstract

Many self-adaptive, autonomous systems rely on component technologies to report anomalies to planning processes that can choose adaptations. What if the analysis technologies themselves need to be adapted? We consider an intrusion detection system (IDS) supported by two component technologies that assist its decision making: a neural network that finds security anomalies and an attack graph that informs the IDS about system states of interest. The IDS purpose is to send alerts regarding security anomalies. Planning processes respond to alerts by selecting mitigation strategies. Mitigations are imposed system-wide and can result in adaptations to the analysis technology, such as the IDS. Without adaptation the IDS can become stagnate in its detection quality. In this paper, we describe an architectural design for an adaptive layer that works directly with an IDS. We examine two use cases involving different mitigation strategies and their impacts on the IDS's supporting components.

Keywords: Self-adaptive systems, intrusion detection systems, neural networks, attack graphs

1. Introduction

Self-adaptive, autonomous systems are deployed to environments where systems require mechanisms that can alter system organization, configuration, or behavior to respond to changes in the environment (Cheng et al., 2009). Failure to respond to such changes results in poor performance, faults, and security vulnerabilities, which warrant the implementation and deployment of a system that is adaptable. Self-adaptive systems (SASs) are implemented using a Monitor-Analyze-Plan-Execute (MAPE) control loop (Kephart & Chess, 2003) which may include a Knowledge component in the case of a MAPE-K loop. These steps can be implemented by individual component technologies or satisfied through the interaction of system components. It is important to inquire into what would happen if some of these components, such as the anomaly detection components, needed to be adapted.

To further this inquiry, we consider an intrusion detection system (IDS) (Liao et al., 2013) that is supported by two component technologies that assist its decision-making processes. These component technologies include a neural network component that finds security anomalies and an attack graph component that informs the IDS about the system states of interest. Collectively, these three components represent analysis components of the SAS that may also need to be adapted to improve their ability to identify security anomalies. Failure to do so could lead to a state of stagnation in which current "blind-spots" in the anomaly detection process are persistent.

In the system presented in this paper, the IDS generates alerts in response to identified security anomalies. Those alerts are forwarded to planning processes that select corresponding mitigation strategies, such as the NIST SP 800-160 candidate mitigations (National Institute of Standards and Technology, 2021). The selected mitigation(s) are imposed system-wide, potentially resulting in adaptations to the anomaly detection components. Further planning qualifies the adaptations that will be applied to those components, while the components themselves are responsible for executing the adaptation.

We describe an architectural design for an adaptive layer that integrates directly with the IDS to facilitate

URI: https://hdl.handle.net/10125/103466
978-0-9981331-6-4

adaptation of the IDS as well as the neural network and attack graph components. We illustrate each of its layers, their components, and the interactions between components as well as elucidate design challenges at each step. We then examine two use cases involving different mitigation strategies and their impact on the IDS and its supporting components.

To provide a brief roadmap, Section 2 provides background research related to SASs, the IDS, the NN and AG components, and NIST CMs. Section 3 describes the architectural design. Section 4 examines the two use cases. Section 5 outlines design challenges across the various components of the architecture as well as the limitations of the design. Section 6 is the conclusion and description of future efforts.

2. Background

2.1. Self-Adaptive Systems

An SAS can be thought of as a system that adapts its functionality as its environment evolves to ensure that its operational goals are consistently met (Cheng et al., 2009) (Elhabbash et al., 2019). As such, it must be able to reason over the conditions that necessitate adaptation (Langford & Cheng, 2019) to enact the strategy that best improves its performance or maintains its compliance with respect to system controls. The typical MAPE-K loop (Kephart & Chess, 2003) implementation is used so that the system continuously monitors relevant data, analyzes the data, and plans appropriate adaptations.

Key challenges to the implementation of an SAS include 1) understanding the minimal amount of information that is required to plan an adaptation, 2) efficiently analyzing the information to produce actionable adaptations, and 3) enabling appropriate adaptation mechanisms that can adapt system functionality (Bellman et al., 2017) (Zambonelli et al., 2011). Designing adaptive components requires sufficient expertise and an understanding of sources of uncertainty that affect the SAS operation (Colman et al., 2014) (Scholze et al., 2013). Justifying the selection of an adaptation can involve making trade-offs between various criteria or making decisions based on limited information about the future state of the environment, both of which can be computationally expensive to perform. In such cases, the system may be required to employ heuristics that can be used to assess adaptations amidst various sources of uncertainty. Multiple interacting MAPE loops to support intra-loop and inter-loop coordination (Vromant et al., 2011) are often needed to address different priorities (Weyns et al., 2013). Adaptation mechanisms are most often deployed as external adaptive frameworks or middleware that can interact with components within the system that can execute adaptations at the system level. When addressing security threats, these systems must be able to adapt security functionality (Yuan et al., 2014) while maintaining other quality concerns (Le, 2015).

Researchers have investigated various approaches to verifying and selecting appropriate adaptations. ActivFORMS (Iftikhar et al., 2014) uses formal models to verify the correctness properties of adaptations. Filieri et al. (2017) use Markov chains to verify adaptations prior to system deployment. In prior work, we have investigated the use of assurance cases to assess adaptations based on their impact on security controls and functional requirements (Jahan et al., 2020). Assessments were conducted through interacting MAPE-K loops following a similar MAPE-K design pattern as those proposed by Weyns et al. (2013). We have also investigated the use of theorem provers to construct formal proofs that demonstrate compliance with system requirements (Marshall et al., 2018) (Riley et al., 2021). Adaptations were assessed using a set of heuristics to measure the risk that the adaptation would fail to maintain compliance with system requirements based on current environment and operating conditions. Other researchers have investigated approaches to constructing and enabling adaptive components. Garlan et al. (2004) have developed Rainbow, a framework that uses an architecture-based model to verify the properties of pre-defined adaptations. In prior work, we have defined a plug-in architecture that acts as a middleware to provide a target system with self-adaptation capabilities (Jahan et al., 2021).

2.2. Intrusion Detection Systems

An IDS is a system that is capable of, at a minimum, detecting an intrusion into a system that violates a core security policy. IDSs can take many forms, relying on a variety of detection methodologies (Liao et al., 2013). Using these techniques, they detect if an intrusion is occurring or has occurred and generate an alert provided to another component that manages the alert. IDSs are typically used for network security, due to the wide variety of existing and emerging network attacks.

Two of the main detection methods used by IDSs are signature-based detection and anomaly-based detection (García-Teodoro et al., 2009) (Liao et al., 2013). For a signature-based approach, the IDS is pre-loaded with knowledge of certain attacks and how they operate on the system. At runtime, the signature-based IDS checks the events that it observes against its known attack patterns to detect threats (Liao et al., 2013).

Anomaly-based detection works slightly differently. Rather than being pre-loaded with attack signatures, the IDS is pre-loaded with typical usage

data. The IDS then monitors the runtime behavior of the system to detect deviations from known patterns of typical use (Liao et al., 2013). Current research is investigating the inclusion of neural networks to identify patterns in typical usage data (Beqiri, 2009).

Ideally, an IDS should be capable of detecting any intrusion into a system or network, even if the attack is trying to disguise itself as benign activity. This detection, however, remains as a significant challenge in the design and deployment of IDS technology (Beqiri, 2009) (Capobianco et al., 2019) (García-Teodoro et al., 2009). For example, a neural network would need access to as much data as possible to understand typical system activity, but it still might be possible for an IDS supported by a neural network to misidentify well-disguised intrusions at the beginning of an attack.

2.3. Neural Networks

Artificial neural networks (NNs) have been proposed for intrusion detection in numerous contexts (Liao et al., 2013). NNs can encode complex functions using a network of neurons (nodes) that propagate their real-valued activation levels through weighted synapses (edges). Recurrent neural networks (RNNs) are a subcategory in which edge loops are allowed, and neuron activation levels may be maintained as mutable state while the network processes its input (Keller et al., 2016). The stateful nature of RNNs enables it to process a sequence of inputs over time, producing output at each step, while holding memory derived from past inputs for arbitrary lengths of time. We believe that this stateful sequence processing capability makes RNNs more promising than stateless feedforward neural networks (FNNs) for our purposes. This is because events in a network system, such as IP packets, are expected to have strong statistical and causal dependencies that will not be captured by analyzing each event in isolation.

NNs generally have many parameters that must be tailored to solve a specific problem. A popular machine learning method for training NNs is gradient descent, which historically proved difficult to perform on RNNs (Hochreiter & Schmidhuber, 1997). A type of restricted RNN architecture called long short-term memory (LSTM) was developed to mitigate these problems (Hochreiter & Schmidhuber, 1997). Multiple variants of LSTM have been developed since then, and they remain popular as a method for creating RNNs that support gradient-based training.

NNs can be trained using various machine learning paradigms. A common paradigm is supervised learning, in which the training samples provided to the learning algorithm are annotated with their expected outputs. Another option is unsupervised learning, in which the training samples do not have labels for the expected output or any other special annotations. Since unsupervised learning is not given a target behavior for each sample, often the best it can do is to learn statistical patterns in the data. While this is a significant limitation, the learned patterns can notably be used for anomaly detection, which is probably more valuable in the IDS domain than in many other domains.

A third learning paradigm is LUPI (Vapnik & Vashist, 2009), in which the training samples contain strictly more information than in supervised learning. The goal of LUPI is to aid the training process by providing more information than what will be available at runtime, which means that the learning algorithm should be designed to take advantage of statistical patterns in the extra variables but must ultimately produce an NN (or other predictive model) that does not require those extra variables as inputs. All three of the above paradigms have been used with LSTM networks (Greff et al., 2017) (Mahasseni et al., 2017) (Xu et al., 2017), although supervised learning is the most popular.

There are other novel approaches to address outstanding research challenges associated with employing NNs for network intrusion detection. LuNet (Wu and Guo, 2019) is a hierarchical deep neural network that combines convolution and recurrent sub-nets to reduce false-positive rates using supervised learning. High false-positive rates can condition users to ignore flagged anomalies and increase overhead in a system that autonomously responds to alerts. ADA (Yuan et al., 2020) employs an LSTM network to perform unsupervised learning over system logs in networks with large volumes of incoming data. ADA uses dynamic thresholds that can be adjusted to improve the NN model during anomaly detection rather than retraining the model which is time intensive. The contribution of this paper is an architectural design that can incorporate these novel efforts, as well as future efforts, into an adaptive architecture that utilizes anomaly detection alongside mitigation and adaptation selection, where adaptations can target the NN to the extent of what configurations it can support.

2.4. Attack Graphs

The term *attack graph* refers to a number of related techniques for security analysis, revolving around the use of a formal state space model in which state transitions represent the actions of an attacker or other force (Li et al., 2021) (Louthan et al., 2014) (Phillips & Swiler, 1998). Terminology related to attack graphs varies significantly across authors. To avoid confusion, we will maintain consistent terminology here even when citing other works that use different terms.

Attack graph analysis requires a state-transition model defining a state space and possible transitions

between states. One approach used by researchers is to model each state as a collection of assets and facts, where facts may describe properties of individual assets or relationships among assets (Li et al., 2021) (Louthan et al., 2014). Transitions can be defined indirectly using exploit patterns with specified preconditions and postconditions, which may contain free variables that must be bound to assets to create a concrete transition.

A state-transition model theoretically defines a complete, static graph describing all states and transitions. Since that graph is likely infinite or intractable to compute, practical analysis often involves computing a subgraph from a specific starting state, possibly using a pruning criterion such as a finite search depth (Louthan et al., 2014). We use the term *attack graph* to refer to the computed subgraph. Attack graphs can be used to analyze system-wide security properties. Examples of analysis tasks include finding the shortest path to a compromised state and estimating the potential consequences of a specific vulnerability (Li et al., 2021).

Even with pruning criteria, attack graph generation can be computationally intensive. Research has been performed on accelerating the computation through algorithmic and hardware improvements (Li et al., 2020) (Li et al., 2021). However, performance can also be improved by restricting the state-transition model. Notably, algorithmic complexity can be significantly reduced by assuming a form of monotonicity, i.e., that preconditions for an exploit never become false after becoming true (Ammann et al., 2002). While one can certainly imagine scenarios where this assumption does not hold, it is somewhat justified by the idea that an attacker is unlikely to give up privileges after obtaining them (Capobianco et al., 2019). This monotonicity idea has led to a variation on attack graphs called *attack dependency graphs* (Louthan et al., 2011). Attack dependency graphs use different types of nodes compared to a basic attack graph, but the encoded information is approximately equivalent if the state-transition model satisfies the monotonicity assumption.

2.5. NIST Candidate Mitigations

In its Special Publication 800-160, Volume 2 Revision 1, the National Institute of Standards and Technology has provided several standardized tables of concepts and terms related to cybersecurity (National Institute of Standards and Technology, 2021). Of particular interest to us is the list of candidate mitigations (CMs), which represent standard, high-level steps that can be taken to mitigate security threats. Each CM has an identifier consisting of the letters *CM* followed by four digits, as well as a human-readable name. For example, *CM1140 Use Alternate Communications* describes the mitigation of changing communication methods to avoid a security threat. Our architectural design will make use of NIST CM identifiers to specify high-level intended behaviors for adaptations.

3. Architecture

The proposed architecture, shown in Figure 1, is made up of several components organized into four layers. These four layers are the *System Layer* (green; bottom), the *Intrusion Detection and Analysis Layer* (red; lower-middle), the *Mitigation Selection Layer* (gold; upper-middle), and the *Adaptive Layer* (blue; top).

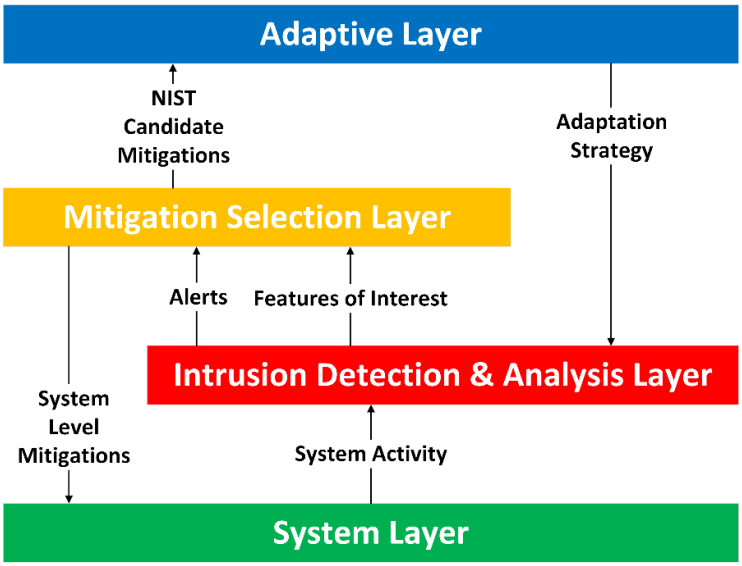

Figure 1: Architectural Diagram (Overview)

These layers work in tandem to support adaptation selection and execution across devices responsible for detecting anomalies and threats to a target system.

The System Layer represents the target system. It is an abstraction over an opaque stream of data, which can only be required to satisfy a limited set of assumptions. Namely, that the data – the system activity – can be monitored and analyzed by an IDS. It would also be beneficial if the target system could digest system mitigations, such as NIST CMs, but it is not necessary. Systems that cannot accept new mitigations, such as legacy systems, may still be adaptable with human intervention. The system mitigation would still be expected to propagate to other system components, such as the IDS, and can result in adaptations to those components. The adaptive layer would reflect an adaptation to other components (e.g., the IDS) in response to the mitigation selection. However, there may be a limited degree to which such adaptations could do more than improve the detection of anomalies.

System activity in the target system is streamed to components within the Intrusion Detection and Analysis Layer (ID&A), shown in Figure 2. This layer is responsible for monitoring the target system and detecting anomalies. The ID&A Layer consists of three major components: the *IDS* (red; middle), the *Attack Graph Component* (rust; right), and the *Neural Network Component* (purple; left). The IDS is further subdivided into three sub-components, which are the *Sensors*, the *Management Server*, and the *Database Server*. The IDS sensors are responsible for parsing the input stream of system activity. Parsed sensor data is streamed to the management server, which processes the sensor data and outputs that data to the neural network to be scored. The management server is responsible for using sensor data, scores, and relevant threat knowledge to determine if an alert should be generated.

Once generated, an alert is logged within the database server and output to the Mitigation Selection Layer. Current system state information, which is derived from the sensor data, is pushed to the attack graph. Relevant threat knowledge is provided by the database server, which maintains a history of anomalous activity, alerts, and adaptations.

The Neural Network component contains a stateful LSTM-based NN that takes individual events from the system activity as input. For each event received, the network's output layer produces a single real value representing the level of suspicion. The NN output is clamped to the range from 0 to 1.

The NN is trained using a history of system activity events, each labeled as either normal or malicious. Potential adaptations for the Neural Network component include retraining with different training data, retraining with different training algorithm parameters, and adjusting the threshold of suspicion that is considered to warrant an alert. We will focus on retraining with different training data as our primary example of NN adaptation. Note that this adaptation may involve changing not only the set of events used for training but also the set of features included in the event data points. However, adding new event features to the

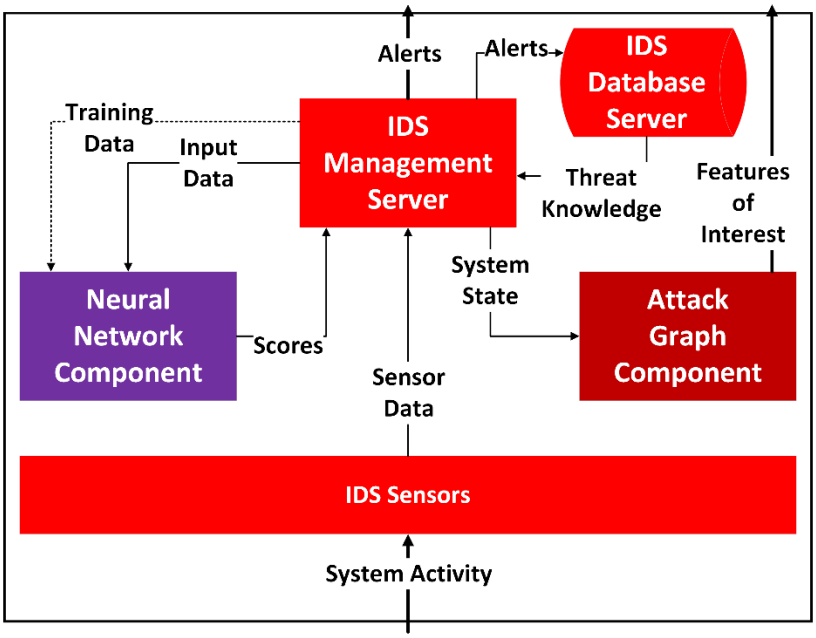

Figure 2: Intrusion Detection & Analysis Layer

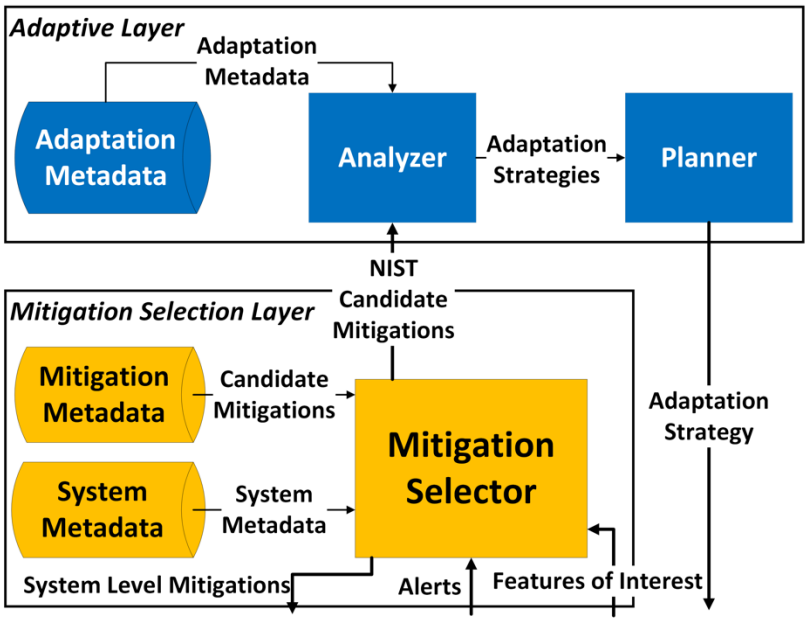

Figure 3: Mitigation & Adaptive Layers

training set means that those new features must also be provided to the NN for all subsequent incoming events.

The AG is a representation of the entire network and system state from the lens of a possible attacker. It is composed of "attack states" and "attack vectors", represented by nodes and edges respectively. Each attack state represents a permission layer that is possible for an attacker to achieve. Attack vectors are representations of attacks that permit transitions from one attack state to another. Given an attack graph and an attack state, it is possible for an analyst to determine not only any future attack states (possible escalation of permissions), but also any prior attack states (how the attacker might have been able to obtain the current state). This process provides tremendous insight for both pre-threat and post-threat analysis.AGs can be used to support a risk assessment of the current system state.

This assessment can be accomplished by establishing metadata associated with the attack states and attack vectors that include information like "Level of Impact" and "Degree of Risk". Combined with the alerts generated by the IDS, this added information would allow each alert to be assigned a "risk value" that can inform the Mitigation Selector and the Adaptive Layer. It can enable those components to make their own decisions regarding which alerts and mitigations to prioritize, or even decide if an alert warrants a mitigation. Additionally, an AG can also provide contextual information, such as open network connections, derived features of interest like packets per second, and the targets of various connected sensors. The inclusion of an AG in our architecture is a novel idea that allows the system to decide its own approach towards detecting possible vulnerabilities.

Alerts generated in the ID&A layer are received by the *Mitigation Selector* in the Mitigation Selection Layer (gold; bottom), shown in Figure 3. The mitigation selector is responsible for selecting a set of NIST CMs that correspond to each alert based on related metadata, such as *System and Mitigation Metadata*. System and mitigation metadata are needed to determine whether a mitigation is appropriate and preferred. A mitigation is appropriate if it is both related to the alert and executable by the components within its scope. Mitigations are related to an alert if they have been determined to mitigate the risk and/or harm that was the cause of the alert or if at least one of the mitigation's specified controls have been impacted by the alert's underlying cause. We define the effectiveness of a mitigation as its capability to reduce further risk or harm due to anomalous or malicious activity. Effectiveness is determined by the proportion of risk or harm that can be reduced by the mitigation and the likelihood of it doing so. A mitigation with a non-zero effectiveness with respect to an alert is appropriate for that alert.

Alternatively, each CM is assigned to a set of security controls (NIST, 2021) that relate to the requirements the affected system must satisfy. If the cause of the alert impacts the system's ability to meet its security controls, then all CMs that have been assigned to impacted controls are categorically applicable to the alert. Determining the applicability of CMs based on system controls is necessary for systems where the effectiveness of a mitigation cannot be quantified.

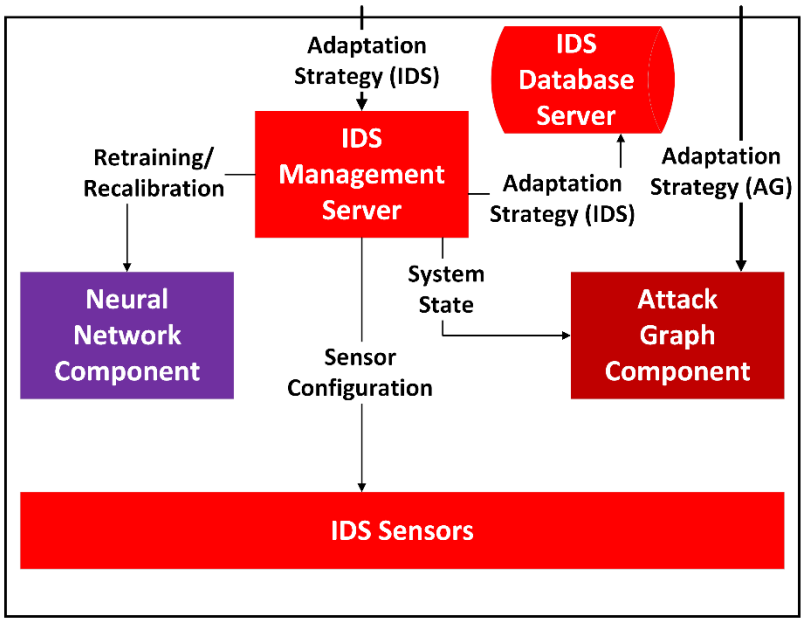

Figure 4: Adapting the ID&A Layer

Every selected CM will have a qualified set of parameters that define the affected component(s) and the intended effect. Those component(s) are within the scope of the mitigation and require some set of effectors that can sufficiently bring about the intended effect. These parameters and their possible values would need to be tailored to each subject system based on domain expertise. Parameters, their values, system architecture, system controls, the selectable CMs, mitigation effectiveness and categories, etc. can be captured within the Mitigation Selection Layer as metadata.

One of the principal challenges to selecting mitigation in response to alerts occurs when there is no one-to-one relationship between alerts and CMs. Some CMs might not be applicable to any alert and some alerts may have multiple applicable CMs. If a CM is not applicable to any alert, that mitigation can be ignored by the mitigation selector. If an alert has multiple applicable CMs, then the mitigation selector should choose the most preferred mitigation of those available. Every CM has some degree of effectiveness and operational cost, which is the resource and performance cost tied to executing the mitigation.

Operational costs can be measured according to the short- and/or long-term impact of enforcing a mitigation for some duration of time. The effectiveness and cost of a CM is both inherent and environmental. For example, *CM1134 Refresh Selected Applications or Components* has the inherent cost of requiring that a system application or component be restarted, or temporarily disabled. Its inherent effectiveness could be determined by whether temporarily disabling an application or component inhibits malicious activity. Inherent effectiveness and cost can be used to define a static profile relating CMs to alerts.

Effectiveness and cost can also be environmental. Restarting an application or component is more costly if it is being heavily utilized and less costly otherwise. The same mitigation might also be more effective when deployed during an ongoing attack or less effective if the attack has already concluded. If such system metadata is available to the mitigation selector at runtime, then the mitigation selector can dynamically select a preferred CM based on the current system state. If no such metadata is available at runtime, then a static profile would be needed based on available system data and domain expertise.

Selected CM(s) are pushed to the Adaptive Layer (blue; top), show in Figure 3, by the mitigation selector. The Adaptive Layer is composed of the *Analyzer* and the *Planner*, which are supported by *Adaptation Metadata*. The analyzer, which receives selected CM(s) from the mitigation selector, uses adaptation metadata to determine a set of adaptation strategies that are applicable to the selected CM(s). Adaptation metadata would include the set of possible adaptation strategies, which must be defined by domain expertise, as well as qualifiers that determine which adaptations can be selected for which CM(s). In this context, adaptations can be thought of as changes to the organization or configuration of components within the ID&A Layer, such as the IDS, attack graph component, or neural network component, or variations of how each CM might be executed by one of those components. For example, *CM1130 Validate Data Quality* can be applied to improve the quality of training data employed by a

NN. There are several approaches to validating the quality, integrity, or consistency of a training set, such as removing rows or features, relabeling rows in the case of supervised learning, etc. Deciding which approach to use can be handled within the Adaptive Layer rather than the Mitigation Selection Layer.

After deciding on a set of possible adaptation strategies, the adaptations are forwarded to the Planner, which is responsible for deciding a single adaptation strategy that covers the selected CM(s). To do so, the planner would need to define a utility metric that can be used to rank a set of adaptations based on selected CM(s) and available knowledge of the affected system(s) or component(s). The highest ranked adaptation for each CM can then be combined into a single adaptation strategy that is then output to the affected components within the ID&A Layer.

The IDS and attack graph component receive an adaptation strategy from the planner depending on the scope of the selected CM(s) as shown in Figure 4. If the IDS or the NN is affected by the adaptation strategy, then the adaptation strategy will be received by the IDS management server and logged within the database server. If the AG is affected by the adaptation strategy, then the attack graph component will receive the adaptation strategy. The IDS receives adaptation strategies on behalf of the neural network component as it is responsible for maintaining and validating the neural network's inputs, which includes the NN training data. If an adaptation strategy applies to the NN, then the IDS must apply the adaptation to the training set, forward the modified training set to the neural network component, and then signal retraining, or formulate a query to recalibrate the NN depending on its implementation and whether it is housed within the IDS or provided as-a-service via the cloud. Once an adaptation has been logged in the database server, the management server sends updated system state information to the attack graph component, if the system state would be affected by the adaptation strategy.

4. Responding to Security Mitigation

We outline two possible use cases among many to illustrate the application of our proposed architecture.

- The NN training set includes mislabeled data that undermine the quality of its classifications
- Seemingly benign behavior hides an escalating attack that results in an observable system compromise

The first use case illustrates a situation where mislabeled data was accidentally or maliciously included within the training set that was used to train the NN. The mislabeled training data results in anomalous activity receiving lower scores, or scores that indicate that the activity is seemingly more "normal" than it should appear. The second use case illustrates a typical scenario where an attacker has begun an escalation of permissions that results in increasing harm to the system. The attacker's activity seems benign at first, even as permissions are being acquired, until a major system component is compromised.

Consider a scenario in which the NN has been trained over data that has been accidentally or maliciously mislabeled. At runtime, the IDS becomes aware of data that should be classified as anomalous either due to human involvement or recently detected anomalous behavior. At which point, the IDS, which is responsible for validating the NN training data, determines that anomalous data exists within the training set and is labeled as "normal" activity. The IDS generates an alert for "Invalid Training Data", which is output to the Mitigation Selection Layer. The mitigation selector receives the alert, assesses possible NIST CMs against available system and mitigation metadata and selects *CM1130 Validate Data Quality*, which is output to the Adaptive Layer.

The analyzer assesses the selected mitigation against available adaptation metadata. It considers possible adaptation strategies that are then forwarded to the planner. The planner selects and qualifies an adaptation strategy to relabel all mislabeled data. The adaptation strategy is then pushed to the ID&A Layer, where the IDS applies the adaptation strategy to the training data to relabel all mislabeled data. The new training data is forwarded to the NN, which is then signaled for retraining. The adaptation is logged to the database server. No further actions are necessary.

For the second use case, consider a scenario in which an attacker has been discreetly performing an attack on the system, escalating their permissions over time. In this example, the NN is unable to detect the attacker's actions as malicious because the training data it possesses is insufficient. Eventually, the attacker compromises a major system component to gain further permissions. When this occurs, the IDS generates an alert indicating "System Compromise".

After logging the alert onto the IDS Database Server, the IDS consults the AG for possible future and past attack states and pushes the alert to the Mitigation Selection Layer. The Mitigation Selection Layer then pulls features of interest from the AG, including prior attack states, and a presumably high-risk value. For this example, the Mitigation Selector might select *CM1134 Refresh Selected Applications or Components* to refresh and restart the compromised components and *CM1155 Validate Data Output* to revalidate and add data to the NN training set. These two CMs are pushed to the Adaptive Layer where the compromised system

component is defined as the target for CM1134, and it defines a new subset of features to be added to the training data for CM1155. These strategies are pushed to the IDS management server, which then executes them, restarting the compromised component as well as retraining and adding data to the NN. These adaptations are logged, and the new System State is pushed to the AG for its adaptation through recompilation.

5. Limitations of the Design

The proposed architectural design includes several challenges related to the implementation of its components. One of its core components is the IDS, which must be adaptable and must maintain a NN. This in turn means that the IDS implementation must be able to parse and execute adaptations, which might include changes to sensors, the NN, access policies, threat knowledge, etc. In addition, managing and adapting a NN would require that the IDS be capable of storing, curating, and validating the NN training data. The IDS would need to employ mechanisms that ensure the integrity of the NN data.

The NN as proposed would conduct supervised learning, which is widely used. However, unsupervised learning might be more suitable in cases where there is little relevant threat knowledge and anomalous activity is only known in contrast to normal activity. For many systems, activity that is considered normal can change over the lifecycle of that system. NNs cannot typically be trained quickly but retraining may still be a necessary adaptation for the NN so that it can continue to acclimate to the target system over time. Lastly, it may be necessary to dynamically adjust sensitivity thresholds so that alerts are not generated too frequently.

In the proposed design, the AG has the unique ability to identify states associated with an identified intrusion. It may be necessary to include further metadata in the description of the AG so that it can also identify relevant features that could be used by the IDS and NN to improve their ability to detect future intrusions. In this way, the IDS might be able to not only recognize typical data but also typical attacks. For an SAS, it may also be necessary to reconfigure the AG to model new sets of attack states and vectors, which can be computationally expensive to do.

6. Discussion and Conclusion

In this paper, we discuss an architectural design that can address the impact of adaptations on analysis components that inform an SAS MAPE loop. We focus on analysis components that include an adaptable IDS that is supported by both a NN and an AG. In this design, the IDS is responsible for generating alerts when anomalies occur, the NN is responsible for identifying anomalies, and the AG is responsible for analyzing attacks associated with anomalies. Once an alert has been generated, a CM is selected that is then enforced system wide. Mitigations can result in adaptations that are defined within an adaptive layer associated with the analysis components. The resulting adaptation can therefore affect the IDS and its supporting components. We examine two use cases that illustrate the application of the adaptive layer within the architecture. It is our intent to employ this architecture to develop monitoring technology that is better equipped to identify security anomalies in a world of ever-increasing complexity.

For future work, we intend to investigate various NN implementations to determine which may prove to be most helpful in identifying anomalies depending on what training data can be gathered. We also seek to investigate what processes can be implemented to refine the selection of mitigations and adaptations based on effectiveness, cost, and a risk assessment of the current anomaly. Lastly, we will investigate novels designs or mechanisms to improve the latency considerations of retraining a NN or reconfiguring an AG.

Acknowledgement: This material is based upon work supported by the US Army ERDC under Contract No. W912HZ21C0062.

7. References

Ammann, P., Wijesekera, D., & Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. 9th ACM Conf. on Computer and Communications Security.

Bellman, K., Landauer, C., Nelson, P., Bencomo, N., Gotz, S, Lewis, P., & Esterle, L. (2017). Self-modeling and self-awareness. In: Kounev S., Kephart J., Milenkoski A., Zhu X. (eds) Self-Aware Computing Systems, Springer, Cham.

Beqiri, E. (2009). Neural networks for intrusion detection systems. In Jahankhani, H., Hessami, A. G., & Hsu, F. (Eds.), Communications in computer and information science: Global security, safety, and sustainability, 45:156-165. Springer.

Capobianco, F., George, R., Huang, K., Jaeger, T., Krishnamurthy, S., Qian, Z., Payer, M., & Yu, P. (2019). Employing attack graphs for intrusion detection. Proc. of the New Security Paradigms Workshop, 16-30.

Cheng, B.H.C., Lemos, R.., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, Whittle, J. (2009). Software engineering for self-adaptive systems: A research roadmap. In B.H.C. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, editors, Software Engineering for Self-Adaptive Systems, LNCS 5525, 1–26. Springer.

Colman, A., Hussein, M., Han, J., & Kapuruge, M. (2014). Context aware and adaptive systems. In: Brézillon, P., Gonzalez, A. J. (Eds.), Context in Computing, Springer, 63-82.

Elhabbash, A., Maria, S., Rami, B., & Peter, T. (2019). Self-awareness in software engineering: A systematic literature review. ACM Trans. on Autonomous and Adaptive Systems, 14(2), 5.

Filieri, A., Maggio, M., Angelopoulos, K., D'ippolito, N., Gerostathopoulos, I., Hempel, A. B., Hoffmann, H., Jamshidi, P., Kalyvianaki, E., Klein, C., Krikava, F., Misailovic, S., Papadopoulos, A. V., Ray, S., Sharifloo, A. M., Shevtsov, S., Ujma, M., & Vogel, T. (2017). Control strategies for self-adaptive software systems, In ACM Trans. on Autonomous and Adaptive Systems.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, 28(1-2), 18-28.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer.

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A search space odyssey. IEEE Transactions on Neural Networks and Learning Systems, 28(10), 2222-2232.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735-1780.

Iftikhar, M. U., & Weyns, D. (2014). ActivFORMS: Active formal models for self-adaptation. In Software Engineering for Adaptive and Self-Managing Systems.

Jahan, S., Riley, I., Walter, C., Gamble, R., Pasco, M., McKinley, P.K., & Cheng, B.H.C. (2020). MAPE-K/MAPE-SAC: An interaction framework for adaptive systems with security assurance cases. Future Generation Computer Systems, 109, 197-209.

Jahan, S., Riley, I., Sabino, A., & Gamble, R. (2021). Towards a plug-in architecture to enable self-adaptation through middleware. IEEE Int'l Conf. on Autonomic Computing and Self-Organizing Systems Companion, 214-219

Keller, J. M., Liu, D., & Fogel, D. B. (2016). *Fundamentals of computational intelligence: Neural networks, fuzzy systems, and evolutionary computation*. IEEE Press.

Kephart, J.O., & Chess, D.M. (2003). The vision of autonomic computing. Computer, 1, 41–50.

Langford, M.A., & Cheng, B.H.C. (2019). Enhancing learning-enabled software systems to address environmental uncertainty. IEEE Int'l Conf. on Autonomic Computing, 115-124.

Le, D.T. (2015). Quality trade-offs in self-protecting system. 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, 152-156.

Li, M., Hawrylak, P., & Hale, J. (2020). Implementing an attack graph generator in CUDA. IEEE Int'l Parallel and Distributed Processing Symp. Workshops, 730-738.

Li, M., Hawrylak, P., & Hale, J. (2021). Strategies for practical hybrid attack graph generation and analysis. Digital Threats: Research and Practice.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. J. of Network and Computer Applications, 36(1), 16-24.

Louthan, G., Haney, M., Hardwicke, P., Hawrylak, P., & Hale, J. (2014). Hybrid extensions for stateful attack graphs. Proceedings of the 9th Annual Cyber and Information Security Research Conference, 101-104.

Louthan, G., Hardwicke, P., Hawrylak, P., & Hale, J. (2011). Toward hybrid attack dependency graphs. 7th Annual Workshop on Cyber Security and Information Intelligence Research.

Mahasseni, B., Lam, M., & Todorovic, S. (2017). Unsupervised video summarization with adversarial LSTM networks. IEEE Conference on Computer Vision and Pattern Recognition, 2982-2991.

Marshall, A., Jahan, S., & Gamble, R. (2018). Assessing the risk of an adaptation using prior compliance Verification. 51st Hawaii Int'l Conf. on Sys. Sciences.

NIST. (2021). *Developing cyber-resilient systems: A systems security engineering approach* (NIST SP 800-160, Vol. 2 Rev. 1). US Dept. of Commerce.

Phillips, C., & Swiler, L. P. (1998). A graph-based system for network-vulnerability analysis. Proceedings of the 1998 Workshop on New Security Paradigms, 71-79.

Riley, I., Jahan, S., Marshall, A., Walter, C., & Gamble, R. (2021). Evaluating verification awareness as a method for assessing adaptation risk. Future Generation Computer Systems, 119, 110-135.

Scholze, S., Barata, J., & Kotte, O. (2013). Context awareness for self-adaptive and highly available production systems. Technological Innovation for the Internet of Things, Springer.

Vapnik, V., & Vashist, A. (2009). Neural Networks, 22(5-6), 544-557.

Vromant, P., Weyns, D., Malek, S., & Andersson, J. (2011). On interacting control loops in self-adaptive systems. Proceedings of Soft. Eng. for Adaptive and Self-Managing Systems.

Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., & Göschka, K. M. (2013). On patterns for decentralized control in self-adaptive systems. LNCS, 76–107.

Wu, P. & Guo, H. (2019). LuNet: A deep neural network for network intrusion detection. IEEE Symposium Series on Computational Intelligence, 617-624.

Xu, H., Gao, Y., Yu, F., & Darrell, T. (2017). End-to-end learning of driving models from large-scale video datasets. IEEE Conference on Computer Vision and Pattern Recognition, 3530-3538.

Yuan, E., Esfahani, N., & Malek, S. (2014). A systematic survey of self-protecting software systems. ACM Trans. on Autonomous and Adaptive Systems, 8(4), 1-41.

Yuan, Y., Adhatarao, S.S., Lin, M., Yuan, Y., Liu, Z., & Fu, X. (2020). ADA: Adaptive deep log anomaly detector. IEEE Conf. on Computer Communication, 2449-2458.

Zambonelli, F., Bicocchi, N., Cabri, G., Leonardi, L., & Puviani, M. (2011). On self-adaptation, self-expression, and self-awareness in autonomic service component ensembles. 5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops, 108–113.