2020 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
CYBER TECHNICAL SESSION
AUGUST 11-13, 2020 - NOVI, MICHIGAN

BEST PRACTICES FOR GROUND VEHICLE INTRUSION DETECTION SYSTEMS

Elena I. Novikova[1], Vu Le[1], Michael Weber[1], Cory Andersen[1], Samuel N. Hamilton[2]

[1]BAE Systems, Arlington, VA
[2]Plucky Innovation, Oakton, VA

ABSTRACT

Addressing the well-established need for accurate cyber situational awareness on military vehicles and weapons platforms, we developed a well-tested, robust Intrusion Detection System – Fox Shield^{TM} – currently rated TRL-8. The system is described and the lessons learned during its development are discussed. The basic principles of our anomaly detectors are outlined, and the details of our innovative warning-aggregating Fuser are presented. Many attack detection examples are presented, using a publicly available CANbus dataset.

Citation: E.I. Novikova, V. Le, M. Weber, C. Andersen, S.N. Hamilton, "Best Practices For Ground Vehicle Intrusion Detection Systems", In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 13-15, 2020.

1. Introduction

The importance of timely, accurate cyber situational awareness on military vehicles and weapons platforms has been well established [1]. Achieving that awareness in those environments with traditional enterprise network Intrusion Detection System (IDS) approaches is infeasible from a data processing and storage perspective, and suboptimal from an attack surface coverage perspective. Traditional IDSs digest voluminous logs from disparate sensors watching many instances of a few types of data streams (e.g., a single HTTPS stream can be examined from the IP, TCP, SSL, HTTP, and application layer perspectives), being more tolerant to false alarms, watching for known attack signatures, unknown attacks, and signs of remote C2 and data exfiltration over long periods of time [2]. Embedded vehicle IDSs must examine few instances of many types of data streams simultaneously (e.g., vehicle mission subsystems control and status, environment sensors, vehicle engine and related system control and status, comms data, external threat and SA sensors) from many perspectives, with almost no tolerance for false alarms, watching for unexpected types of attack against vehicle mission success in real time, and do it without overwhelming embedded processing systems with a fraction of the capacity of enterprise SOC installations. It is a difficult challenge that requires platform owners, systems integrators, and cyber software builders to work together. Towards that end, we share some lessons learned from implementing our IDS system designed for ground vehicle environments (Fox

Shield^{TM}), and show its performance against a publicly available CAN Bus dataset.

In next section we will review best practices literature for design, implementation, and installation of IDSs. Sections 3 to 6 describe in details how we built the embedded vehicle IDS Fox Shield^{TM}, using best practices as main points of discussion. Section 7 illustrates the concepts with a case study.

2. Related Work

Since embedded-vehicle IDS designs and implementations have more constraints than those for common IP network IDSs, the related work section first surveys best practices for building IDSs in broader domains and then focuses on best practices we learned during building an effective vehicle IDS in a resource-constrained environment.

The most mentioned best practice in literature is related to tuning IDS sensors. Tuning sensors during implementation and deployment is considered as one of the most important tasks in IDS development [3-5]. Grading and selecting the alarms [6] to display or present to operators is essential in highlighting important messages. Without alarm filtering [7], IDSs can overwhelm operators with alarms, potentially affecting them negatively. Alarms that come from different sensors should be correlated to ensure that the messages presented to the operators are consistent and complete without excessive duplication [8-10]. In addition to coherent alarms, IDSs also use logging mechanisms to detect potential attacks via pattern matching and other data mining techniques [11]. The correlation among logs and alarms is necessary to ensure potential attacks are captured as much as possible and to reduce false alarms in many cases [12,13], which enables the use of digital forensics.

Before tuning IDS sensors, it is imperative to study normal operations data [14]. A baseline is obtained by profiling the system during normal operations, in benign environments, and is used later for identification of any deviation from that norm. Deviations trigger warnings or alarms, depending on the characteristics of the abnormality. However, not all deviations are potential intrusions; some could come from a glitch or a sudden surge in the number of messages, for instance. The alarms generated in these cases are considered false alarms [15]. The number of false alarms must be controlled to ensure the trustworthiness of the IDS [16]. One strategy to reduce false alarms is fine-tuning the IDS with more data that cover as many different situations possible.

These best practices described above are popular and widely used in IDS implementation. Other practices, even though not as popular, are also mentioned in IDS-related literature. One of them is the careful choice of the location where the IDS sensors should be placed to obtain the best outcome, as described in [17,18]. Since the amount of traffic increases every year on most networks, it is necessary to install multiple IDSs to increase the probability of detection of network attacks [19], or to use multi-layered IDSs [20]. This might be true for CAN bus systems as well, with more data flowing with most versions upgrades of software and hardware. When the traffic becomes too intensive, the network becomes too complex, and the threat landscape becomes too vast, one of the important best practices is to assess the inventory and information to harden system cyber security [21].

Given the strict environment where vehicle IDS is built, additional best practices must be taken into account. We call them “generic specialization,” “platform-unique interface code isolation,” and “avoid confusion by fusion.” These new best practices will be discussed at length in next sections.

3. Guiding principles used during Developing Fox Shield^{TM}.

In light of best practices outlined above, we approached the task of developing our IDS with reliability, modularity, efficiency, adaptability, transferability, and ease of tuning as our guiding principles. In addition, we worked under two overarching mandates: “do no harm” and “emit almost zero false alarms.”

These requirements are all high priority, however, the most important one, even though it is frequently overlooked, is the need to not make matters worse. An IDS has to be safe to use – it should not lead to a harmful increase in computational resource consumption, it should not crash, and it should not get stuck in infinite loops or have other logic errors.

A *reliable* IDS, in addition to being safe, is capable of detecting anomalies and raising alarms in a repeatable, predicable manner.

A *modular* IDS is written using an approach that allows each functional component to be designed, coded, debugged, and tuned as separately as possible from other components.

An *efficient* IDS is frugal with memory and CPU cycles while meeting its functional goals. Algorithms can typically trade RAM for CPU cycles and vice-versa; choosing the correct algorithm first and implementing it efficiently second keeps the IDS small and fast.

An *adaptable* IDS easily accommodates changing conditions, such as requiring only limited or no retraining in field conditions without the need for major model overhauls.

A *transferable* IDS can be used on other platforms of the same type with no or minimal tuning, as well as on platforms of completely different types with modification only to the data ingestion and management interfaces. The logic core of a transferable IDS does not need code modifications to run on different processors or buses.

An *easily tuned* IDS provides access to algorithm parameters in external files that are loaded at the startup of the system (so-called “configuration” or “model” files). There also should be clear documentation as to how to tune the sensors and interpret the results. Custom external tools to aid understanding of input data and each sensor’s response to it also contribute to easy tuning. Automatic tuning is a nice goal, but its outputs will inevitably need tweaking or explaining, in which case the custom external tools are again essential.

A *trustable* IDS emits almost zero false alarms. Together with the reliability characteristic, trust goes to the essence of an IDS. An IDS that too frequently falses will quickly become a nuisance ignored by the platform operators it aims to inform. The IDS must also be decisive; it must not rapidly issue and cancel related alarms.

We should not overlook the importance of logging information during the work of the IDS; it is needed for forensic analysis, as well as for reporting. The level of detail logged is set in the model files to support the expected decision explanation or report granularity.

The application of best practices during development is intended to result in the IDS that satisfies all of the mandates and requirements.

4. Information flow through Fox Shield^{TM}

The generalized design of our IDS is shown in Figure 1. The IDS consists primarily of the *Front End* and the *Anomaly Engine*. There are auxiliary services, such as a Logger, Scheduler, and Performance Manager, as well as a set of model files – all necessary for operation of the system. Together, the Front End and Anomaly Engine (AE) implement the *generic specialization* best practice. Each detector in the AE specializes in a specific type of anomaly, allowing it to do one thing well, but operates on a generic data representation produced by the Front End, allowing it to be re-used wherever that one thing is useful. The *modular* and *transferable* principles drive this structure.

We will discuss the main components in the sections that follow.

4.1. Front End and data representation

Vehicles are different, but mission vulnerabilities are often shared. Therefore, it makes sense to separate code that must change between vehicle types from code that can stay the same, as this limits the scope of change required to transfer the system to a new vehicle type. Investing time into an

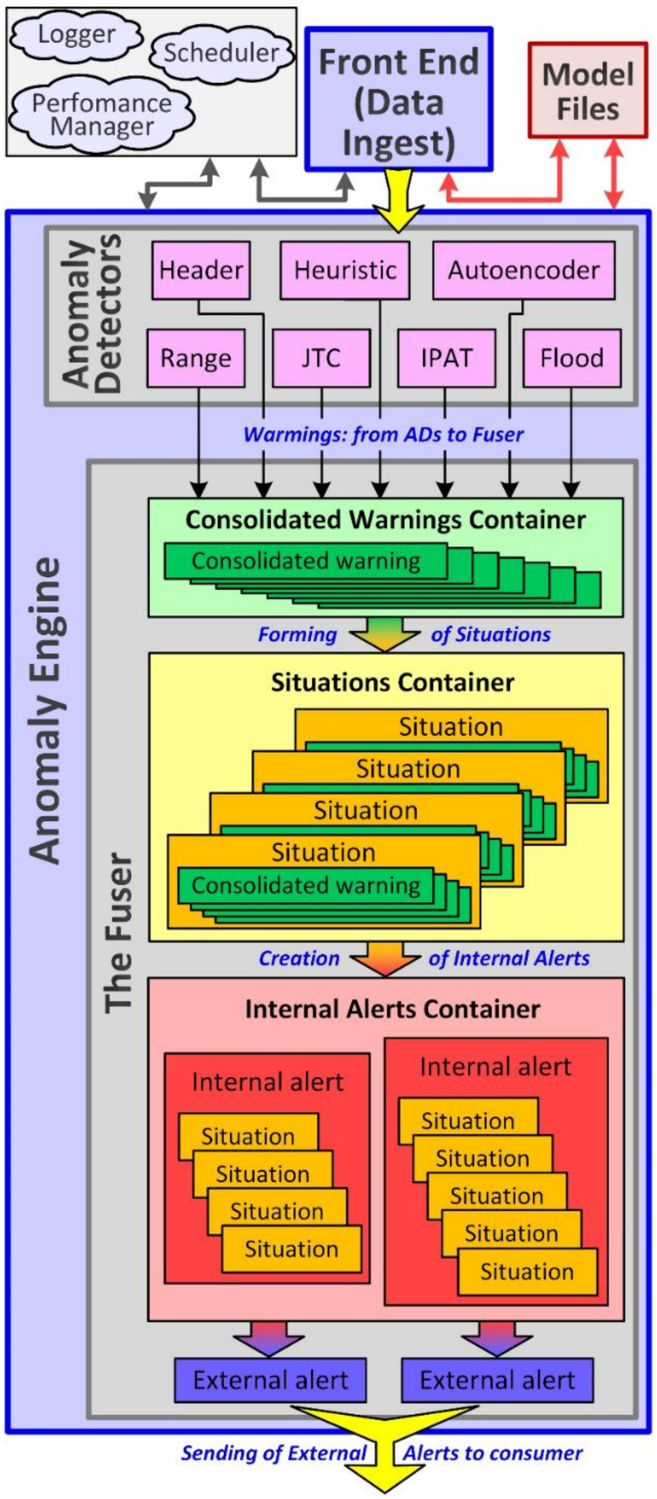

Figure 1: Major components of Fox Shield^{TM}, and the flow of information through them.

encapsulated generic approach enhances the portability and re-usability of the brains of the IDS – the Anomaly Engine – saving cost and reducing risk. However, platform-specific code is unavoidable. In Fox Shield^{TM} we learned several valuable lessons about data ingestion and representation and established these best practices.

Abstract away bytes: One might be tempted to use raw bytes as the ultimate generic data structure. Unfortunately, coding at the byte-level is error-prone, requires careful consideration of machine/compiler details, and results in platform- or datasource-specific sensors. In Fox Shield^{TM}, the Front End transforms ingested bytes into *features* represented with standard data types (e.g., `int` or `float`) for consumption by the Analysis Engine. Platform-specific details such as bus data capture format, data byte positions, and endianness are contained in only the Front End, which uses a generic format to describe those details for each platform. Each feature definition is given a unique ID for later reference. A feature can be a signal value extracted from the data stream, such as a temperature, or it can be a piece of meta-information, such as packet arrival time or the entropy of its contents.

Isolate platform-specific code through interfaces: Fox Shield^{TM} data ingestion is decoupled from analysis through a three-link chain. Data consumers expect a series of timestamped values called features that are uniquely identified by a number. In the first link, bytes are read and transformed into a platform-specific packet structure and added to a queue. This function is typically performed by a data capture mechanism outside the scope of the IDS itself. The second link, embodied in the Front End, reads packets off the queue, transforms them into features, and stores them in a multi-buffered holding pattern that permits downstream usage and upstream ingestion simultaneously. In the final link, data consumers process the buffered features. This feature-oriented ingestion allows consumers to be reused in other platforms provided that their logic isn’t platform-specific and that the new platform's data can be represented by feature streams. Furthermore, this three-part system may be multithreaded to increase throughput if resources allow.

Separate feature values and metadata: A feature may have a great deal of associated information that does not change for every packet or data unit. Bundling this metadata with ingested values is inefficient and unnecessary. Instead, a unique instance of metadata about the feature may be kept separate from value instances and accessed through the feature's ID.

4.2. Anomaly Engine – the heart of IDS

Upon exiting the Front End, the stream of data, in the form of series of features, enters the Anomaly Engine (AE). The heart of our IDS, the AE consists of seven specialized anomaly detectors (ADs) and the Fuser. All ADs not only consume the data in the same format, but also produce uniformly formatted warnings.

We implemented seven ADs; their parameters are specified in several model files. These files hold an extract of the most important information from available Interface Control Documents (ICDs), as well as the results of the model-learning phase of detector training on benign data. Placement of as many tuning parameters as possible in the model file instead of only in the code is driven by the *adaptable*, *transferable*, and *easily tuned* principles. While developing these detectors, we made a few mistakes and realized shortcomings of some approaches. The ADs and lessons learned are discussed in detail in section 5.

The fuser consumes the warnings produced by the ADs and issues alerts when appropriate. It provides multi-layered, non-trivial logic, comparison and analysis of received warning contents, and guards against false alarms and unnecessary alert update jitter. The fuser accomplishes its tasks using pre-trained models that outline relationships between various anomaly detectors, and special rules with multiple parameters – all of which are available in the model files. The Fuser is discussed in detail in section 6.

5. Anomaly Detectors

We will describe all seven of our anomaly detectors here, and emphasize some of the lessons we learned while designing and implementing those.

5.1. Header Anomaly Detector

The Header Detector inspects each packet header via a 'packet header' feature, and by its nature each Header AD is particular to one kind of packet. Initially, Header AD checked only the validity of sources and destinations (i.e., validity of Parameter Group Numbers, PGNs on a CAN bus), but we later realized that it would be beneficial to check the priority and other header fields of each packet, so those checks were added. We also learned that ICDs are not necessarily up-to-date with respect to the systems they document (for example, actual packet priorities), and it is essential to check the data observed during benign runs against most the reliable documentation sources. When in doubt, the data itself rules.

5.2. IPAT Anomaly Detector

The Inter Packet Arrival Time (IPAT) detector checks the timing of sequential packets of the same ID (e.g., same source and PGN on a CAN bus). Packets with different IDs are analyzed in separate streams. The average inter-packet time is measured in a sliding time window, and compared to the one stated in the model files. E.G., a message sent at 10Hz might have an expected inter-arrival time of 100ms. The average inter packet arrival time (time-delta) is allowed to be shorter than the canonical one by a coefficient stated in the model file, and any greater discrepancy results in an internal IPAT warning. Violations to the longer side are not reported. Violations to the shorter side are not reported immediately; there is some extra filtering present which assures the violation is legitimately concerning. Parameters of the extra filtering can be tuned from the model file.

The need for different treatment of the violations on the shorter and longer side, as well as the need for extra filtering for the shorter-side violations, became clear when we studied our vast database of benign recordings. We learned on more than one occasion that following the practice of profiling normal network behavior *carefully* is a key process for IDS successful deployment. As you see in Figure 2, the violations of the prescribed time-deltas to the longer side happen numerous times; it can be attributed to various benign situations that are beyond the scope of this writing. The violations on the shorter side (in benign conditions) may be attributed to rare, but possible, network traffic conditions as well as the details of the mechanism of time-stamping. These details can be different for different configurations and installations, but these violations are often transient, and thus, an extra requirement for persistency of the reportable short time-deltas provides a guard against false alarms. Steps necessary to reduce the false alarm rate – another essential best practice in developing IDS – should be taken at every possible junction, including in the functioning of every warning-issuing AD. Reducing irrelevant warnings reduces the load on the Fuser, which is tasked with

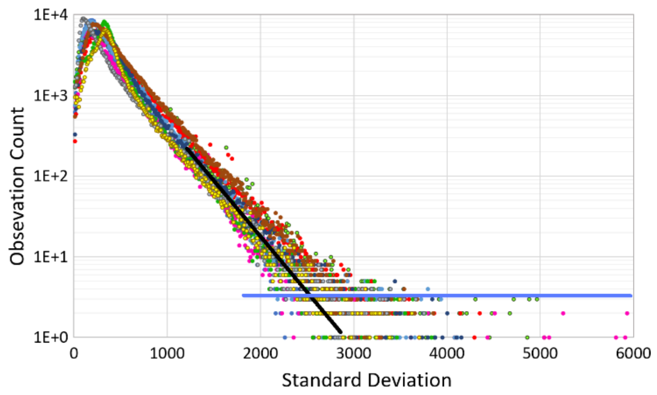

Figure 3: Heuristic: Observed standard deviations of the signal values (for one signal type) in sliding time window, in benign data for several vehicles. Also are shown approximations of regular data and noise (two straight lines).

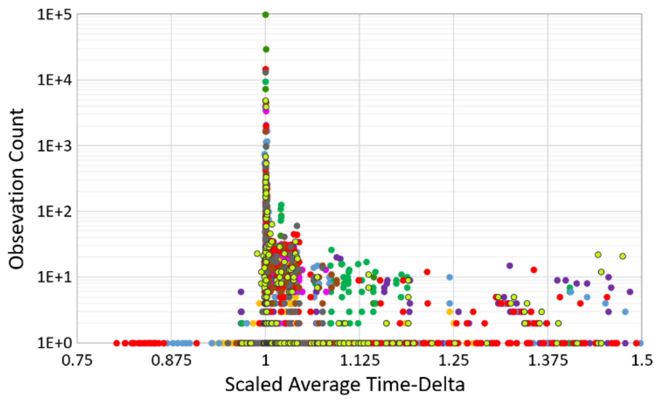

Figure 2: IPAT: Observed average time-deltas in sliding time window, in benign data for several vehicles.

digesting warnings and issuing alerts to the external consumer.

5.3. Heuristic Anomaly Detector

The Heuristic AD is responsible for guarding some statistical properties of signals. Just like IPAT, it works in a sliding time window, but unlike IPAT, it works with individual signal values, not packets. For each window position, for each signal, two statistics are computed: the standard deviation of the values of the signal, and the average inter-packet jump in the signal values. These statistics are compared to the maxima (per signal) allowed (set in model files), and if a violation occurs, a warning is sent to the Fuser.

As with all our anomaly detectors, we follow the best practice of tuning the Heuristic sensor independently, using well-understood benign data. The distribution of the standard deviation measured for one of the signals, for several vehicles, for all positions of the sliding time window, is shown in Figure 3. The Heuristic AD guards against excessive jitter in data values, since it may be indicative of several types of attacks; the determination of the maximum allowed standard deviation is made from benign data analysis. The distribution shows an expected gradual drop in observations as the standard deviation increases. Eventually, the distribution flattens, indicating entry into the “noise” region. Multiple trials aimed at optimizing the Heuristic AD’s warning policy

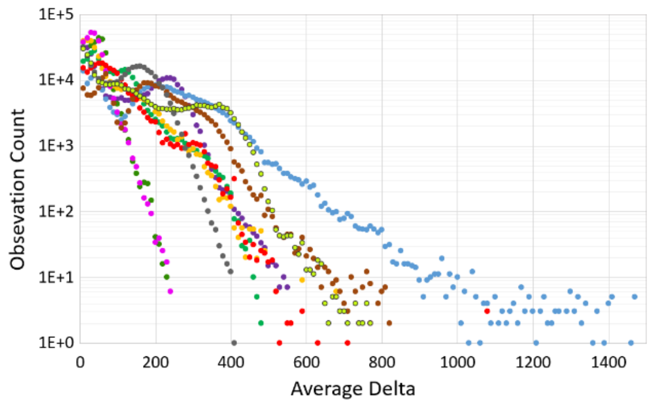

Figure 4: Heuristic: Observed average delta in consecutive values of a signal in sliding time window, in benign data for several vehicles.

resulted in adopting a threshold that lies approximately at the intersection of the downward part of the distribution (summarized by black line on the graph) and the noisy flat tail (blue line of the graph). Obviously, that threshold allows a higher probability for false warnings. In case of the Heuristic AD, most of the solution to this problem is delegated to the Fuser, where the lower reliability of Heuristic warnings is taken into account.

We came across a property of the Heuristic AD while studying the benign data that emphasized the need for adaptability. Apparently, the second statistic used by Heuristic AD may depend on a specific vehicle, as Figure 5 illustrates. The ideal reporting threshold position for this signal changes from about 200 for the tightest distributions shown, to about 1000 for the distribution with the heavy tail of high values, shown by green dots. Anticipating cases like this, a quick tune-up in the field may be required. Alternatively, this flavor of Heuristic warning can be disabled in the model file.

5.4. Anomaly Detector “Jump to Constant”

The Jump to Constant (JTC) AD is capable of detecting instantaneous jumps in signal values followed by a “plateau” – an unusual behavior in a valid signal, but something that may be indicative of a cyber-attack or a malfunction in a physical sensor. JTC requires tuning similar to that of the Heuristic AD, with additional parameters needed to detect the “const-ness” of the signal and the definition of a “significant” jump.

It is worth noting that our JTC AD detected a malfunction of a physical sensor during live testing. The alarm was raised by our IDS within a minute of the first malfunction, while the analysis of the hardware logs (not related to the IDS) showed the same malfunctions hours later.

5.5. Range Anomaly Detector

The Range AD assures that signal values fit into the proper range. From the first glance it should be the easiest AD to code, but we learned that in practice it is not so. Often the signals violate the ranges stated in the ICD, and it usually happens when the physical sensor is malfunctioning, or during glitches of the analog-to-digital converters, or some other such circumstances. For example, the ICD stated that the value of a particular 8-bit signal should not exceed 200, yet we observed values of 254 and 255. These values in some cases were flags for the cases of “no signal” or “not ready.” We found that cases like these are not always well documented, and hence, care must be taken when coding and training a Range AD. One possibility is to always allow a signal to assume the maximum value possible for its size in bits (and maybe even a value one less than that maximum), but guard against all other values exceeding the maximum stated in the ICD.

With these uncertainties in interpretation of what allowed range is, the Range sensor may also be a source of excessive warnings that are not indicative of a cyber attack. Just like in the case of Heuristic, the Fuser model is adjusted to treat warnings from the Range AD with extra “suspicion.”

5.6. Flood Anomaly Detector

The Flood AD detects excessive overall bus load. It may guard several busses, if more than one is present in the system, checking each bus separately. Warnings are issued when the observed load exceeds the threshold set in the model file. The threshold should be somewhat below the knee of the bus's utilization/throughput curve so that the IDS host has time to respond to the overload condition before it crushes the system completely.

5.7. Set of Autoencoder Anomaly Detectors

Finally, we employ a set of Autoencoder anomaly detectors. An autoencoder is a neural net trained to reproduce its input as its output. The input in our case is a trio of instantaneous values from three well-chosen signals. We call each such trio of signals a "group." The process of choosing groups is based on relationships between signals – only signals related to each other in some manner, however complex, end up forming a group.

A signal may belong to several groups, depending on how it interacts with other signals. To study this interaction, we interpret the normalized instantaneous values of three signals (the potential candidates to form a group) as coordinates in 3-D space, and thus, each instance in time is represented by a point in this 3-D space. By "interaction" between different signals we mean the patterns of the manifolds and sub-volumes formed by the cloud of these points in the full 3-D space.

As seen in Figure 5, the "cloud" of points from a well performing group is showing a clear pattern; the volume of the cloud occupied by these points is a small fraction of the full 3-D space bounded by the min and max of the three signals. This formation of manifolds and sub-planes happens every time when the signals represent some physical variables that are related to each other, or some "state variables" that are somehow connected.

For each IDS implementation (in this content, for each deployment of the Anomaly Engine to a different environment) we identify as many well-formed 3-signal groups as possible. Each group results in one Autoencoder anomaly detector, and we use all well-formed groups, resulting in several Autoencoders running simultaneously (if possible, in multithreaded manner). Each group guards its three signals to fit inside the 3-D point cloud learned from the benign data. If, for any given moment in time, one or more signals in the group assume a value that "pulls" the corresponding data point away from the expected pattern established for this group – this may be an indication of a violation of the established benign behavior. The Autoencoder's neural net recognizes the violation and produces a warning, with a significance value proportional to the degree of the violation, to the Fuser.

We learned a valuable lesson when developing our Autoencoder detector, which can be formulated as the following best practice that should be always followed: "Pay attention to what your preliminary analysis of the data is trying to tell you." In our case, the "talking" was done by the distribution of the error of reconstruction. This error is defined as the distance between the points P_{before} and P_{after}. P_{before} is a point 3-D space, like one of the dots in Figure 5, with coordinates equal to the normalized values of the three signals at a moment in time, before this trio entered the neural net. P_{after} is the point reconstructed by the neural net from the values of P_{before}. By the very definition of the autoencoder, this distance (in agreed upon measure) has to be minimal, although it is never zero. This distance, i.e., the error of the reconstruction, is averaged over all of the pointes in

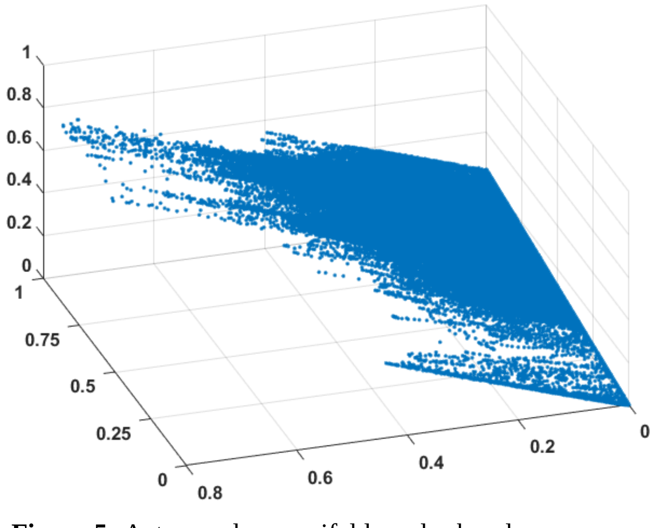

Figure 5: Autoencoder: manifolds and sub-volumes formed by one of the groups of SynCAN data [22].

each training epoch. Training stops when the average error stops shrinking from epoch to epoch.

In our attempts to find the best configuration for the 3-signal autoencoder, we decided to characterize every configuration with not just the *average* of the reconstruction errors during training, but with a *distribution* of the errors. After training of each configuration was complete, we considered a histogram reflecting the distribution of the reconstruction error, as seen in Figure 6, where the reconstruction error for all points used during training is depicted.

It is clear from the blue curve in Figure 6 that the peak at the left side of the histogram represents some special case that has to be investigated. If we were not to use the full histogram, and were to be satisfied by the (very small) average reconstruction error, we would have stopped here, and would have never seen that the small *average* error is almost fully attributable to the fact that about 90% of data fall into this peak. Further investigation showed that the peak is formed by the points that belong to one very limited volume in the 3-D space of potential instantaneous values of the three signals. This means that, during training, the neural net is overconcentrated in reflecting this small volume of the 3-D space in its weights, i.e. overtraining itself for the points in this specific portion of space, and "forgetting" to pay attention to all other data. We argue that it is wise to force the neural net to train more uniformly, to cover equally all possible relations between three signals, and not "get stuck" on one small sub-volume due to popularity. This idea resulted in changing how we selected training data points; instead of using all available points in the training data, we implemented a "bin-and-cut" approach: the 3-D space is diced into small cuboids, and only a limited number of data points from each cuboid is allowed into training data set, cutting away the rest of repetitive data points. Autoencoders trained using the bin-and-cut approach adapt to recognize patterns formed in all sub-volumes and manifolds of the 3-D space. This behavior is reflected in the new distribution of errors (red curve) shown in Figure 6: the overall histogram is shifted to the left with respect to the blue curve, signaling significant improvement in recognition of all patterns present in the relationships between the three signals in the group.

Autoencoder AD is very reliable, but just like another reliable detector – IPAT, – it can use some extra filtering, to assure that the violations it detects are not transient.

6. Fuser

Warnings from all ADs are consumed by the Fuser, which acts as an integrator and filter. A diagram of information flow inside the Fuser consumes the better portion of Figure 1. In the following sections, we will examine the Fuser's construction and operation in detail, emphasizing the best practices leveraged to develop a better, safer, more robust IDS generating as few false alarms as possible.

The Fuser is a tiered system working with four main classes of object: (1) consolidated warnings, (2) situations, (3) internal alerts, and (4) external alerts. In the spirit of *modularity*, we designed, coded, and debugged these classes separately, tuning the behaviors of each one independently, and then tuning the interactions between them. Also, to practice *efficiency*, we paid close attention to the memory consumption and execution time for

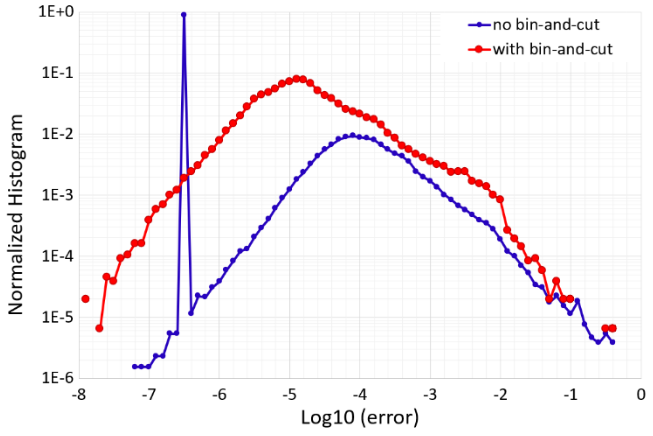

Figure 6: Autoencoder: distribution of the reconstruction error for all training data, for one of the autoencoder of 3-signal groups.

each component algorithm, since the ability to work in real time and to fit onto an execution platform with limited resources is crucial for a vehicle IDS.

6.1. Consolidated Warnings

A *reliable* vehicle IDS must carefully track alert priority, timing, and "need-to-issue." To that end, each anomaly detector assigns a significance metric to each warning it issues; this is the beginning of the anomaly intensity assessment for potential alerts. The locality principle suggests that each AD is best equipped to determine that significance, as its logic performs the initial detection, and it has access to more information than would make sense to pass on to another module to do the same job. The fuser receives these warnings and proceeds with its first task: consolidation of incoming warnings, including tracking their significance and timing.

All incoming warnings for the "same" anomaly are collapsed into one Consolidated Warning (CW). "Same" means the warnings were issued by the same AD for the same source and target with respect to the same feature ID. Observation time and severity may change, so the CW tracks the first and last times this "same" warning has been received along with the highest significance seen. In this way, a continuous stream of warnings about an ongoing anomaly from a single AD is condensed to a single object describing how long something has been going on and how significant the originating AD believes it to be. The fuser keeps all CWs in a single CW container that reflects the current status of the continuously changing stream of "feeder" warnings coming from all anomaly detectors.

When a new kind of "feeder" warning arrives at the Fuser and there is not already a CW that is the "same," a new CW is created and assigned a "time of life" constant. Per the *easily tuned* principle, these constants are specified for each AD in the model files. Every CW must be updated by a matching "feeder" warning at least as often as its "time of life," otherwise it is marked for deletion and elimination from all "downstream" structures it participates in, such as situations and alerts.

6.2. Situations

The concept of situations was first introduced by Debar and Wespi [23]. A situation is a collection of CWs that have one, two, or three properties in common, where the properties are: source, target, and the issuing AD. A situation with all three properties matching may contain only one CW, since by definition all incoming warnings that have matching source, target, and issuing AD will be consolidated into a single CW. Situations of the other six types may contain multiple CWs whose properties match in a specific way. For example, CWs whose source and target fields match but whose issuing AD do not match define a type, and CWs whose target and AD fields match but whose sources do not match define another type. Seven different types of situations are possible given the three properties tested. Important attributes of a situation include its severity, which is calculated from the significances of its member CWs, and its rank, which is an ordering of the importance or meaningfulness for the seven possible types. This ordering is configured in the model files.

Fox Shield™ uses situations to characterize what's happening on the vehicle with maximum accuracy, based on all information coming from all ADs, considering all sources, targets, and types of anomaly. This characterization goes long way towards satisfying the best practice of profiling normal network behavior (during tuning) as a key process for IDS deployment. The advantage of situations is that the tuning process produces a detailed picture of how warnings of different types interact, and how those interactions should be considered during the fusion process. In keeping with the *adaptable*, *transferable*, and *easily tuned* principles, these consideration tunings are stored in the model files.

When the fuser updates the set of CWs, each of them is checked for membership in existing situations, which are updated based the properties of their member CWs. Next, the fuser scans the updated set of CWs to determine if new situations have arisen due to the current relationships among the CWs. The creation of new situations is stopped when there is no warning left that doesn't belong to at least one situation. Thus, the order of creation of the situations of different types is important, and remains as one of the tuning parameters in the model file.

After all possible situations are created, the next step in their evolution happens: all situations, both old ones and the newly created ones, absorb warnings from the current set of CWs. Each CW can belong to any number of situations. A CW is absorbed into a situation if its source/target/AD tuple properly matches corresponding properties of the situation, given its type. As a result of this absorption, there will be situations that have, for example, all warnings coming from a particular AD, or situations that have matching source and target, but different issuing ADs, and so on.

Some situations are deemed "alertable." Alertability of a situation is fully defined by its severity and by the types of anomaly detectors that issued its warnings. A situation with only one AD involved is alertable when its severity exceeds the self-alert threshold for the issuing AD. If there are two warning-issuing ADs involved in a given situation, it is called alertable if its severity exceeds a threshold set for this AD pair in a symmetrical square matrix with the size equal to the number of ADs in the system. If there are more than two ADs involved in the situation, the situation's severity has to exceed the highest threshold for all available pairs of ADs. Such an approach assures that only CWs from compatible detectors are forming alertable situations. All these thresholds are set in the model file.

With this system, Fox Shield™ achieves fine control over the relative importance of different ADs, their reliability (how much we can trust their warnings), and interactions between different ADs, down to outright prohibition of alertable situation that happens to involve certain pairs of anomaly detectors. For example, a situation that involves Header and JTC warnings will never be deemed alertable, because of the high value of the threshold set for this pairing. Even if these two detectors are reporting at the same time, chances are they are reporting about unrelated events. Similarly, JTC and IPAT are incompatible. The two noisiest ADs, Heuristic and Range, have a relatively high threshold for alertability as a pair, thus shifting the responsibility of alert initiation to other, more reliable situations (the self-alert threshold for each of these two ADs is even higher than their pair-threshold).

In the course of evolution of situations, a change of situation's alertability may happen. A previously non-alertable situation may become alertable if escalation of significance of some or all of its warnings leads to a sufficient increase of situation's severity. The loss of alertability is also possible.

The last property to describe here is the end-of-life of situations and warnings. As we already mentioned, a CW expires when it isn't updated for a time exceeding its time-of-life. When a CW expires, it is removed from all situations it belongs to, which may lead to a change of situation's type or an empty situation. Empty situations immediately expire. After situation updates, duplicate situations are removed.

6.3. Internal Alerts

After all updated CWs are absorbed into situations, the situations updated, and the situation list is cleared of expired items and duplicates, the forming and updating of internal alerts begins. An Internal alert consists of one or more alertable situations deemed to be part of the same alertable cause. Internal alerts can be viewed as the union of all CWs present in its member situations.

Internal alert update processing starts by checking whether every current alertable situation belongs to at least one internal alert; we will call an alertable situation that is not yet a member of an internal alert an "orphan." If any orphan situations exist, aggregation by similarity occurs: situations that are

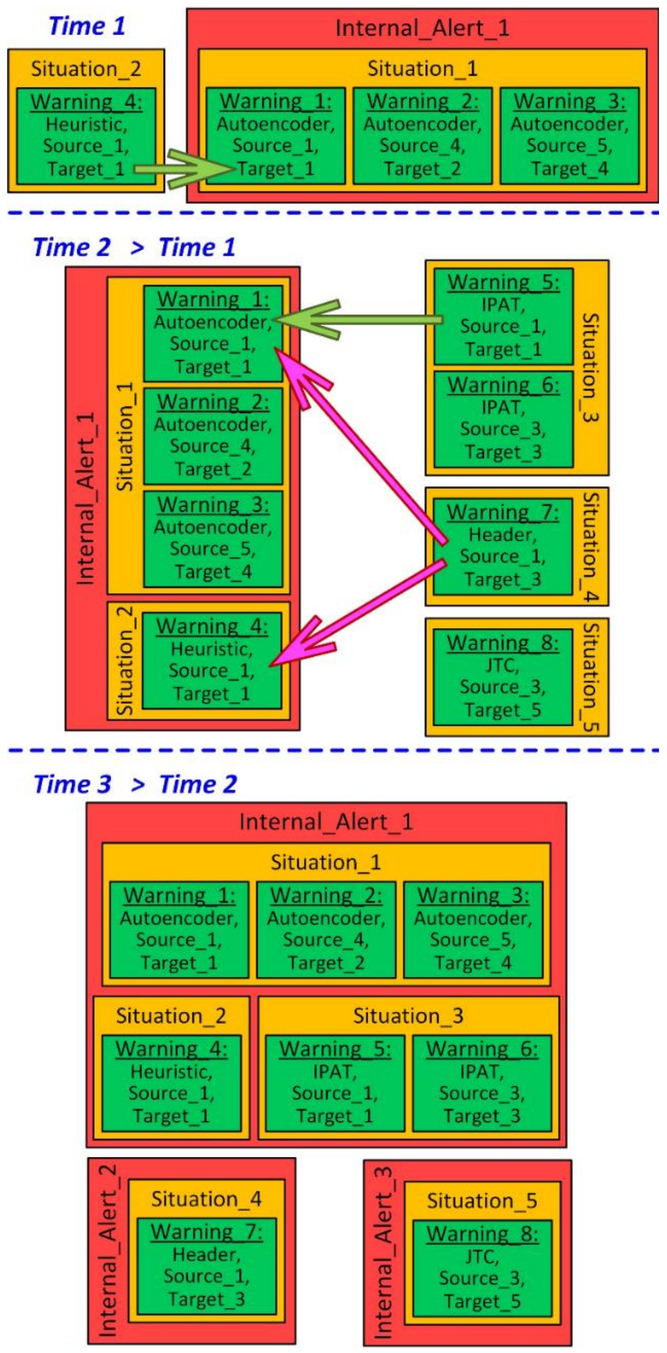
Time 1
Internal_Alert_1
Situation_2
Situation_1
Warning_4: Heuristic, Source_1, Target_1
Warning_1: Autoencoder, Source_1, Target_1
Warning_2: Autoencoder, Source_4, Target_2
Warning_3: Autoencoder, Source_5, Target_4
Time 2 > Time 1
Warning_5: IPAT, Source_1, Target_1
Warning_6: IPAT, Source_3, Target_3
Situation_3
Warning_7: Header, Source_1, Target_3
Situation_4
Warning_8: JTC, Source_3, Target_5
Situation_5
Time 3 > Time 2
Internal_Alert_2
Internal_Alert_3

Figure 7: Example of evolution of internal alerts, in terms of absorbing new situations. The status of internal alert(s) and situations are shown for three moments in time.

already members of existing internal alerts evaluate their similarity to each of the orphan situations to determine if the orphan should be pulled in to an existing alert. The evaluations are done in order of situation rank, from high to low; this represents one of the parameters for Fuser tuning. Higher-ranked situations have a better chance of pulling the new situation into its host internal alert, similar to a sports team draft picking process. In our current tuning, the situation type containing warnings matched by both source and target have the highest rank, followed by situations having warnings with matching target and issuing AD. Orphan situations not pulled in to any existing internal alert form new internal alerts.

We will describe here the aggregation rules, which are an important part of internal alert formation and evolution, using Figure 7 as an example. Three moments in time are shown: At “time 1,” situation #2 is pulled in to internal alert #1. At “time 2” (occurring after “time 1”), situations #3, #4, and #5 are evaluated, but only situation #3 is pulled, indicated by the green arrow. As a result, at “time 3” (occurring after “time 2”), there are three active internal alerts.

When an evaluation is made, a situation tries to pull the orphan in to join an internal alert. Pulling is successful if the situations are similar enough, which is the case if there is at least one pair of CWs (one CW in the pulling situation, and another CW in the situation being pulled) that are similar enough. Similarity is measured by the degree of match between sources and targets multiplied by the entry in the “proximity matrix” for the pair of ADs issuing the warnings. For the pull to succeed, the degree of similarity has to exceed a threshold specified in the model file.

The proximity matrix is an *asymmetric* square matrix specified in the model file; it is the main tuning tool for the Fuser. It determines the priority of each AD with respect to other ADs. The proximity matrix governs what ADs are considered to be “primary” (warnings with these ADs will be successful “pullers”), and what ADs are considered to be “secondary” (warnings coming from these ADs will be pulled in by others, but will rarely or never successfully initiate a pull). The proximity matrix also reflects our opinion of what pairs of detectors are likely to trigger during the same attack, and what pairs of detectors, even if triggered at the same time, were probably triggered by different attacks. For example, Flood warnings are very likely to pull situations with IPAT warnings in them – these two detectors are triggered by the same type of disturbance and should typically inform a single alert. IPAT is also allowed to easily pull Flood. Autoencoder warnings are allowed to pull Heuristic warnings, but not vice-versa. Heuristic warnings are not an exclusive signature of attacks that cause Autoencoder to trigger; Heuristic warnings may as well be caused by an attack that triggers JTC or even IPAT.

The presence of a new alertable situation that was not pulled into any of the existing internal alerts typically indicates that it is incompatible with the existing internal alerts, and should reside in a new internal alert.

Returning to Figure 7, we will explain why each of the pulling attempts succeeded or failed.

At “time 1,” the existing internal alert #1 contains only situation #1, which evaluates pulling situation #2. For the pull to succeed, one of the CWs inside the alert (i.e., inside situation #1) has to be able to pull one of the CWs inside the situation being pulled (i.e., situation #2), which in this case contains only one Heuristic CW. First, the source and target similarity is checked. CW #1 (puller) has the same source/target combination as CW #4 (pullee), producing a score of two (one each for target and source match). This score is multiplied by the proximity matrix value for the Autoencoder/Heuristic puller/pullee combination. This value is set high, because we know that anomalies triggering autoencoder have a reasonable probability of triggering Heuristic as well (but the opposite is not true, as mentioned above).

When aggregation at “time 1” is complete, internal alert #1 has two situations and four CWs, as shown in panel “time 2.” At “time 2,” there are also three new situations present. Situation #3 can be easily pulled into the alert by the virtue of CW #1 pulling CW #5. Their source and target both match, and the proximity matrix value for Autoencoder/IPAT is high. Notice also that this is the only way situation #3 can be pulled into the alert: CW #6 (the second warning of the situation #3) has no matching sources or targets with any of the warnings already present in the alert, and CW #5 cannot be pulled by any warning other than #1. CWs #2 and #3 do not match the sources or targets of CW #5 (so the score to be multiplied by the proximity matrix entry is zero). CW #4 has a matching source/target combination, but the proximity matrix entry for the Heuristic/IPAT combination is zero: Heuristic warnings are not allowed to pull other warnings.

Turning to situation #4 at “time 2,” CW #7 has a non-zero source/target score with CW #1 and #4. For warnings #1 and #7, the proximity matrix prohibits Autoencoder and Header from pulling each other because they are likely triggered by unrelated disturbances (which may be happening at the same time, but have a different nature). For warnings #4 and #7, the pull cannot succeed because the proximity matrix values prevent Heuristic warnings from pulling any other warnings (since Heuristic warnings can be seen in many different types of disturbances). Thus, neither situation #1 nor situation #2 can pull situation #4 into internal alert #1.

Considering situation #5: the proximity matrix has a high score for the JTC detector as a pullee for the Autoencoder as puller. Unfortunately, there is no source/target match between CW #8, the only entry in the situation #5, and any of the Autoencoder warnings already present in the internal alert #1. The only other CW in the alert, #4, is from Heuristic, which as discussed above always fails to pull. Thus, situation #5 cannot be pulled in to internal alert #1, and aggregation assessments for time #2 are complete.

This brings us to “time 3,” shown in the last panel of Figure 7. The fuser created an internal alert for the higher-ranked situation, and after another round of aggregation evaluation, another internal alert for the last situation since they could not be aggregated due to the proximity matrix and source/target scores. These separate internal alerts indicate possible simultaneous disturbances, separate and different from the one that resulted in internal alert #1.

An important characteristic of an internal alert is its severity, computed from the severity of its member situations. For later external consumption, this number is then reduced to a discrete characteristic: “low,” “medium,” or “high,” using thresholds set in model file.

Another property of the internal alert is its type, which is a characterization of a possible attack that caused this alert. This characterization is done via mapping of the set of ADs that caused the alert to an attack name, such as “Flood,” “Impersonation,” “Status manipulation,” “Fuzzing,” “Substitution,” or “Shadow.”

Internal alerts evolve when their component situations and warnings evolve and when new situations are pulled into existing alerts. As old warnings “age out” and are removed, the situations change: their type or severity may change, or they may die out altogether. All these changes cause changes in the internal alert. When new situations are pulled into existing alerts, the severity of the alerts may change. All these changes mean that internal alerts are fluid entities that reflect the most up-to-date information happening “on the wire,” the information that is initially sent to the Fuser by anomaly detectors and consolidated using multiple highly configurable mechanisms inside the Fuser.

6.4. External Alerts

Internal alerts lead to the issuance of External alerts, the only notification available to the consumer of our IDS, besides logs. External alerts are generated by an internal alert under certain conditions. First, the corresponding internal alert has to have at least one alertable situation. Even though an internal alert cannot be created out of non-alertable situation, the evolution of the member warnings may lead to the loss of the alertability property. Second, a certain time has to pass after creation of the internal alert before it can trigger an external alert: this delay allows for the initial evolution to happen, i.e., multiple situations (and warnings) will collect themselves in this alert, resulting in a better evaluation of the status of the vehicle. The delay time depends on the alert’s severity and is set in the model file.

Evolution of internal alerts may result in updates or outright cancellation of corresponding external alerts. Depending on the current status, the external change may happen immediately, i.e. together with the change in the internal alerts, or after some delay (in case of a change in severity, for example). Cancellation of an external alert, in most cases, will be delayed with respect to cancellation of the internal alert. All of the delays between creation, change, and cancellation of internal alerts, and the corresponding actions in the external alerts, are needed to allow for the internal alert to settle down and minimize change and on-and-off jitter in the external alerts.

6.5. Summing up the role of Fuser

As we see in the previous sections, the Fuser is a layered system with multiple tuning possibilities: various thresholds and relationship matrices, collected in the model file, allow for turning the multiple warnings sent by many anomaly detectors into a concise, jitter-free, low false-alarm-rate set of external alerts. As best practices of IDS development require, each of the multiple phases of the transformation from warnings to external alerts can be tuned separately. Interactions between phases can also be handled one-at-a-time. The principles of reliability, modularity, efficiency, adaptability, transferability, and ease of tuning are all observed. As a result, the final system is capable of correctly reflecting the status of the systems it protects within the scope of its detectors, even if several different attacks are happening simultaneously.

7. Case Study

Comparison of the behavior of different IDSs, or even different configurations for a single IDS, requires a stable input dataset. While we developed our IDS against large volumes of internal data, we also tested it using a published dataset, SynCAN [22]. After training on the SynCAN benign data set, we tested our IDS using their "attack" files. We will concentrate here on the simultaneous detection of the attacks by our various ADs, as well as on the merging ability of our unique Fuser; the response of our Autoencoder detectors to other SynCAN attacks will be detailed in a future publication.

7.1. Examples of "plateau" attacks

Figure 8 shows the response of the Fox Shield™ IDS to three consecutive SynCAN "plateau" attack signals. During Attack 1, signal ID4-Sig1 is flattened (the blue curve in Figure 8a), while the signal ID5-Sig1 is left at its original values. During Attack 2, the signal ID1-Sig1 (orange curve) is pulled down significantly and flattened. During Attack 3, signal ID5-Sig1 is flattened (green curve), while the other signals remain untouched.

In our study of SynCAN data, These three signals form the Autoencoder group #8, while ID1-Sig1 is also a member of Autoencoder groups #1 and #2. The response of all three Autoencoder groups is shown in Figure 8b; the height of the colored bars is proportional to the significance of the warnings. Attack 2 is the easiest to detect, and warnings from all three groups containing the attacked signal (groups #1, #2, and #8) arrive at full force. Attacks 1 and 3 are not so easily detectable, as the flattened portions of the attacked signals remain well inside their corresponding data ranges. However, Autoencoder group #8 issued very clear warnings in both cases.

Autoencoder detector, in addition to its neural net, also has a low-pass filter, with some additional dampening properties, which is applied to the raw warnings before they are issued to the Fuser; this is done to eliminate transients and to reduce false alarms. This results in the short time delay of the start and end of the autoencoder warnings activity, compared to the time of start and end of the attacks, as can be noticed in Figure 8b.

The JTC AD successfully detected all three attacks (Figure 8c). JTC has a delay reaction to the attack as it waits to see that the value of the signal stays constant after the jump.

The Heuristic AD detects unusual statistical properties of a signal using a sliding time window. It easily catches the abrupt change in value during the beginning and the end of the Attack 2, but not during Attack 3, or at the end of Attack 1, when the disturbance is not strong enough (Figure 8d). It also gives a few false-positive warnings after Attack 1.

The work of the Fuser is shown in Figure 8e. The warnings related to the Attack 1 are coming from Heuristic, Autoencoder, and JTC sensors. Heuristic warning shows up first, but it is not highly respected by the Fuser, since it is known to have false-positives. Thus, the internal alert is not formed for Attack 1 until Heuristic warnings are confirmed by the Autoencoder warnings. Soon

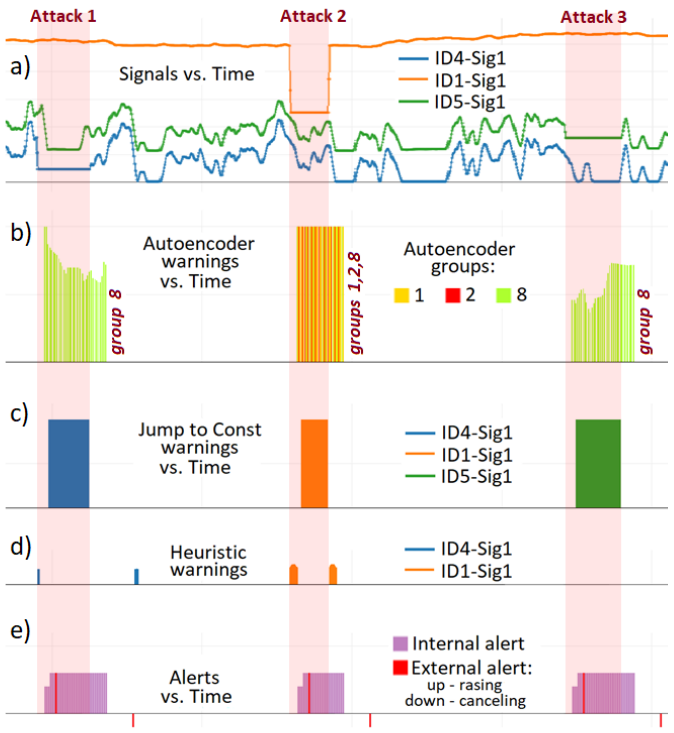

Figure 8: Detecting "plateau" attacks in SynCAN data.

after, JTC joins the chorus of warnings, and the internal alert intensity is upgraded (see the step up in the purple bars). Soon after the upgrade, the external alert is issued to the host system (the red bar above the line). The small time delay between internal and external alerts allows for settlement of the severity and for completeness of the evolution of the internal alert, allowing it to collect all its situations and warnings. As a result, the jitter of the alerts going out of the system is reduced: only one external alert is issued, and it fully describes the attack taking place. After the warnings that gave rise to the internal alert expire, the internal alert is cancelled, and, after appropriate delay, the external alert is cancelled as well (the red bar below the line). Attacks 2 and 3 are handled similarly.

7.2. Examples of "flood" attacks

Flood attacks are characterized by a large number of extra packets for one or more packet types. Detection of three flood attacks is shown in Figure 9, with the panel a giving the rate of three types of packets as a function of time. Our IPAT anomaly detector, working in a sliding time window (hence reacting with a short delay), easily finds these attacks (see Figure 9b), and issues appropriate warnings, with significance proportional to the degree of violation of the expected time deltas. Recall that each packet may have several signals in it; any violation of the packet rate most probably means violations of signal shapes for every signal contained in that packet. The response of the Heuristic detector (Figure 9c) is shown for some of the signals contained in packets ID1, ID2, and ID9; and it confirms the violation of the expected statistics for these signals. Autoencoder, as another signal-based detector, issues warnings for many groups (Figure 9d), since each packet under attack has several signals involved, and some signals participate in more than one group. The Fuser (Figure 9e) has no problem summarizing the received warnings, creating internal alerts, and issuing (and canceling) external alerts.

For in-depth understanding on how our ADs detect the flood attacks of SynCAN dataset, consider Figure 10, where an attack on packet ID7 is shown in detail. The number of ID7 packets per second and the values of the signal ID7-Sig1 are shown as a function of time in panels a) and b). It

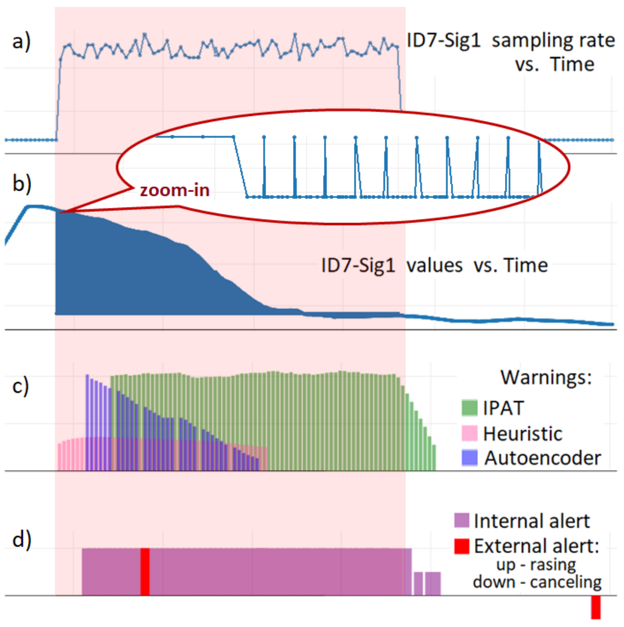

Figure 10: Anatomy of a flood attack and its detection.

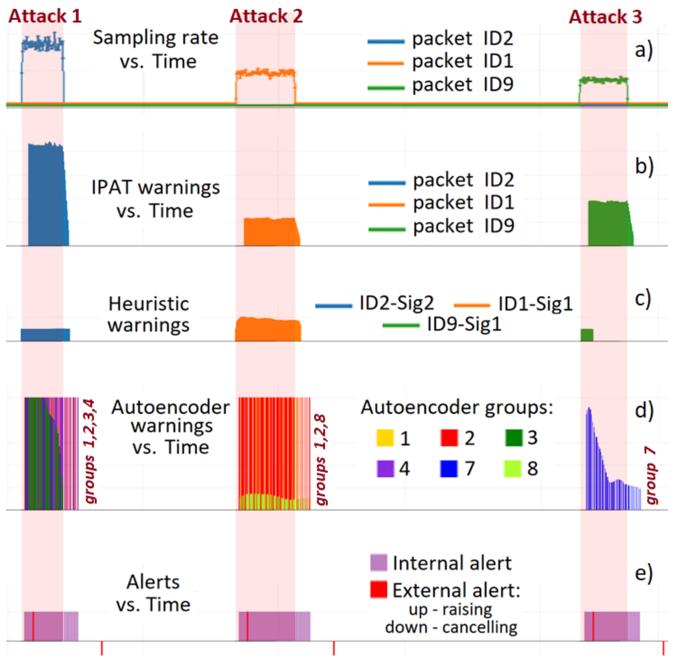

Figure 9: Detecting "flood" attacks in SynCAN data.

is clear that the attack consists of inserting extra packets with valid, zeroed signal values into the ID7 packet stream (the lack of Range warnings indicates zero is a valid value for that signal). While IPAT AD focuses on the timing of packets, both Heuristic and Autoencoder examine the values of signals. The zeroed values produce disturbances in the patterns expected by both of these ADs; this results in warnings, issued in support of warnings from IPAT – see Figure 10c. The significance of Autoencoder warnings diminish as time progresses because the actual, original values approach the artificial values inserted into the signal over time, as seen in panel b). Coherent work of all ADs involved allows the Fuser (Figure 10d) not only to issue a proper timely alert, but also classify it correctly.

8. Conclusions

We demonstrated how a collection of best practices and principles allowed us to create a reliable, low-false-alarm IDS currently at TRL-8. Best practices include “generic specialization,” “platform-unique interface code isolation,” and “avoid confusion by fusion.” Guiding principles include reliability, modularity, efficiency, adaptability, transferability, and ease of tuning. The performance of that IDS was also demonstrated on a publicly available dataset.

REFERENCES

[1] P. Hayden, D. Woolrich and K. Sobolewski. Providing Cyber Situational Awareness on Defense Platform Networks. 2018. Journal of Cyber Security and Information Systems.

[2] J.A. Mohammed, and B.G. Kok. Intrusion Detection Systems: Principles and Perspectives. 2018. Journal of Multidisciplinary Engineering Science Studies. ISSN: 2458-925X.

[3] B. Greenwood. Tuning an IDS/IPS From The Ground UP. 2007. SANS Institute.

[4] A. Ely. IDS Best Practices. 2010. https://www.networkcomputing.com/networking/ids-best-practice

[5] Z. Yu, J. Tsai, and T. Weigert. An Automatically Tuning Intrusion Detection System. 2007. IEEE Transactions on Cybernetics 37(2):373-84. https://doi.org/10.1109/TSMCB.2006.885306

[6] L. LaPadula. Intrusion detection system requirements: a capabilities description in terms of the network monitoring and assessment module of CSAP21. 2001. MITRE Technical Report.

[7] T. Sommestad, U. Franke. A test of intrusion alert filtering based on network information. 2015. Security and Communication Networks/Volumn 8, Issue 13. https://doi.org/10.1002/sec.1173.

[8] M.M. Siraj. Survey and Comparative Analysis of Alert Correlation Systems in Information Security. 2007. The 3rd Brunei International Conference on Engineering and Technology 2008.

[9] S. Salah, G.M. Fernandez, and J.E. Diaz-Verdejo. A model-based survey of alert correlation techniques. 2013. Computer Networks 57:1289-1317. https://doi.org/10.1016/j.commet.2012.10.022.

[10] S. A. Mirheidari, S. Arshad, and R. Jalili. Alert Correlation Algorithms: A survey and taxonomy. 2013. In Cyberspace Safety and Security. https://arxiv.org/ftp/arxiv/papers/1811/1811.00921.pdf.

[11] A. Muscat. A Log Analysis based Intrusion Detection System for the creation of a Specification Based Intrusion Prevention System. 2003. Proceedings of the University of Malta Annual Computer Science Research Workshop, 2003.

[12] D. Gorton. Extending Intrusion Detection with Alert Correlation and Intrusion Tolerance. 2003.

MPhil Thesis, Chalmers University of Technology, Department of Computer Engineering, Goteborg, Sweden.

[13] R. Yusof, S.R. Selamat, and S. Sahib. Intrusion alert correlation technique analysis for heterogeneous log. 2008. International Journal of Computer Science and Network Security, 8:132–138, September 2008.

[14] L.H. Yeo, X. Che, and S. Lakkaraju. Understanding Modern Intrusion Detection Systems: A Survey. 2017. arXiv:1708.07174

[15] H.S. Milan, K. Singh: Reducing false alarms in intrusion detection systems – a survey. 2018. Int. Res. J. Eng. Technol. (IRJET) 05(02), 9–12

[16] Q. Qassim, A. Patel, and A. Mohd-Zin. Strategy to Reduce False Alarms in Intrusion Detection and Prevention Systems. 2014. International Arab Journal of Information Technology, Vol. 11, No. 5, September 2014

[17] L. Babatope, B. Lawal, and I. Ayobami. Strategic Sensor Placement for Intrusion Detection in Network-Based IDS. 2014. I.J. Intelligent Systems and Applications, 2014, 02, 61-68.

[18] S. Noel, and S. Jajodia. Optimal IDS Sensor Placement and Alert Prioritization Using Attack Graphs. 2018. Journal of Network and Systems Management 16(3):259-275. DOI: 10.1007/s10922-008-9109-x

[19] T. Ha, S. Yoon, A. C. Risdianto, J. Kim and H. Lim. Suspicious Flow Forwarding for Multiple Intrusion Detection Systems on Software-Defined Networks. 2016. in IEEE Network, vol. 30, no. 6, pp. 22-27, November-December 2016, doi: 10.1109/MNET.2016.1600106NM.

[20] C. Wang, R. Huang, W. Zhang and J. Sun. Multilayer Intrusion Detection System Based On Semi-supervised Clustering. 2019. 16th International Computer Conference on Wavelet Active Media Technology and Information Processing, Chengdu, China, 2019, pp. 355-360.

[21] D. Zahn. ICS Cybersecurity: You Cannot Secure What You Cannot See. 2017. PAS Global, LLC 2017.

[22] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data", arXiv preprint arXiv:1906.02492, 2019

[23] H. Debar, and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In Proceedings of the International Symposium on Recent Advances in Intrusion Detection. 2001. 85--103