From Intrusion Detection to Software Design

Sandro Etalle

Eindhoven University of Technology,
University of Twente,
and SecurityMatters BV
The Netherlands
s.etalle@tue.nl

Abstract. I believe the single most important reason why we are so helpless against cyber-attackers is that present systems are not supervisable. This opinion is developed in years spent working on network intrusion detection, both as academic and entrepreneur. I believe we need to start writing software and systems that are supervisable by design; in particular, we should do this for embedded devices. In this paper, I present a personal view on the field of intrusion detection, and conclude with some consideration on software design.

1 Preamble

Allow me to start with a personal note: it is useful to understand where my comments come from. I landed on the field of intrusion detection in 2004, after years of moving from rather theoretical to increasingly more practical research topics. We dove into the intrusion detection field with the declared intent of setting up a company afterwards. After years of trying many useless ideas, we focused on a couple of promising technologies. In 2009, my 2 PhD students Damiano Bolzoni and Emmanuele Zambon and I started SecurityMatters. As of May 2017, SecurityMatters is doing well, and there are some very demanding customers who are very happy with its network monitoring system, so in-between the failures we must have done a couple of things right. While I need to clarify that SecurityMatters appliance is now much more than a network intrusion detection system and certainly way more than an anomaly-based intrusion detection system, SecurityMatters has been a tremendous learning experience regarding intrusion detection. In what follows I would like to share with you some of the lessons learned.

2 A Journey in Intrusion Detection

Network intrusion detection is the art of detecting when something goes wrong simply by monitoring network traffic. This can be done at different places in a system. In an industrial system, you can monitor the networks of the web

applications, the back office (Windows), the SCADA[1] system and the PLC[2] (I have used an industrial control system as reference, but this is immaterial). Depending where you look, you have different observables. Regardless of *where* you do the monitoring, there are two ways to detect when something goes wrong in a system: either you recognize the wrong behaviour or you are able to recognize the correct behaviour and you alert when something deviates from it. So you either have a model of the malicious behaviour or you have a model of the legitimate behaviour. There is no third way, even though you can intermix the two approaches.

This is reflected in the notation used in the intrusion detection community [1, 2], where *knowledge based intrusion detection* (a.k.a., misuse based[3]) is the kind of intrusion detection that relies on a model of the attack, and *behaviour based intrusion detection* the one that relies on the model of the legitimate behaviour. In turn, behaviour based NIDS are usually subdivided in *anomaly based* NIDS and *specification based intrusion detection* [3], with the distinction that in anomaly based NIDS the model of the target system is built more or less automatically during a "learning phase", while in specification-based NIDS models are "manually developed specifications that capture legitimate (rather than previously seen) system behaviors" [4]. The common perception about knowledge-based vs. anomaly-based and specification-based NIDS is that

P1 Knowledge-based NIDSs work well in practical deployments, but they are "ineffective"

P2 Anomaly-based NIDS are effective (only) in benchmarks, but do not work well in practical deployments.

P3 Specification-based NIDS are effective (only) in benchmarks and for very specific small systems, but cannot be applied to practical large systems (next to being too expensive to build and maintain).

In what follows, I will touch on what I think are the reasons behind this "perception", and I will particularly focus on anomaly detection systems because our experience with them is instrumental to the goal of this paper. Where I want to get to in the end is to argue that *the true reason of the shortcomings of acceptance-based systems (P2 and P3) are more rooted in the way we design software than in the actual limitations of those approaches.*

But first, we need to agree on the parameters we refer to, when we evaluate the detection systems. Intuitively, IDSs need to be effective on real systems

[1] Supervisory control and data acquisition (SCADA). For the purpose of this paper it is a control (computer) system used e.g., in industrial control systems. Intuitively, SCADA systems control e.g., PLCs.

[2] Programmable Logic Controller (PLC). Typically small computer systems used in e.g., manufacturing to connect to sensors and actuators.

[3] The notation in the literature is unfortunately confusing: *misuse* based systems are often narrowly associated with the use of signatures; similarly, anomaly based systems are usually associated with the use of machine-learning techniques like neural networks, while their scope is much broader.

and cost-effective to operate, and *in my opinion* this translates in the following partially unusual list of desiderata:

D1 High **detection rate** (effectiveness), also w.r.t. attacks that have not been witnessed yet (e.g., 0-days).
D2 Low **false positive rate** (FPR). The FPR is one of the important factors in determining the total cost of ownership of the intrusion detection system.
D3 **Actionability**. When the IDS raises an alert, someone needs to act upon it. The more information the IDS can provide over the alert raised that can be useful to determine the reaction strategy, the better it is. While often forgotten in the benchmarks, actionability is always an important factor in the operational cost of an IDS.
D4 **Adaptability**. Most IT systems change continuously (even SCADA system, for that matter), therefore the IDS has to be able to cope with that. In our experience, adaptability is another primary factor in the total cost of ownership of an IDS, because changes can raise false alerts, that need to be acted upon.
D5 **Scalability**. One of the obvious challenges ahead is monitoring increasingly complex, heterogeneous and open systems of systems. Not all IDS technologies scale up that well.

For the sake of clarity, we need to unclutter the terminology used in the sequel, because the word "system" is overloaded and it is used to indicate both the monitoring and the monitored system. To distinguish the two uses of "system" I will use the following notation

- the "target system" (or "underlying system") is the system being monitored,
- the "system" is usually the monitoring system (the NIDS).

We can now discuss P1 ... P3, starting with knowledge-based detection.

Knowledge-Based Systems That knowledge-based NIDS systems "work" is demonstrated by the fact that basically all network intrusion detection and prevention systems commercially available are knowledge-based (typically based on signatures). There are probably millions of knowledge-based NIDS in use around the globe. In particular, knowledge-based systems score very well on actionability (they recognize the kind of attack, so they can immediately refer to the appropriate mitigation strategy), scalability (when you recognize the attacks it does not matter if you are looking at one target system or at a hundred of them, provided that the FPR is reasonably low).

However, they are ineffective because it is very easy for attackers to evade them [5,6]. Knowledge-based systems (in particular, signature-based) catch mainstream, well-established attacks, but they are always a few steps behind, and are actually helpless against skilful and targeted attackers. Knowledge-based detection is (and I believe will always be) extremely useful, because it handles efficiently the low-key attacks, but will never be the key technology that will defend us from the prepared attacker.

So let us move to behavior-based NIDS.

Behavior-Based Systems As argued before, here I will focus in particular on anomaly detection systems. In a nutshell, the art of anomaly-based intrusion detection is finding a suitable abstraction function AF such that if $AF(present\_state) \notin AF(model\_of\_target\_system)$ then the system raises an alert. In the anomaly-based systems the model of the system is built using machine learning techniques. A hard to break misunderstanding is that the machine learning in use must be general-purpose and domain agnostic, like e.g., neural networks. This is not so, and nowadays the machine learning (and the AF) used is often tailored for the specific protocol and the specific domain of the target system. We will further elaborate on that in the section about *whitebox* anomaly detection. On the other hand, since we are talking about behavior-based detection, the AF should in theory be attack-agnostic. In practice, however, this cannot be completely so, in the sense that the possibly interesting anomalies (the attacks) must not be lost in the abstraction (***). So to build a good AF you do need to have some idea of the possible attacker vectors and the kind of events you are interested in observing. If you don't know what you are looking for, you are probably not going to find it.

Getting back to the statement P2, that "anomaly-based systems do not work in practice" It is now interesting to take a look at it in the light of the experiences we had in making anomaly detection systems actually work. Let us look at D1 D5 and how anomaly detection copes with them. Allow me to keep D1 and D2 (detection rate and false positives) as last.

Actionability. By definition, anomaly-based network intrusion detection systems (ABNIDSs) do not recognize the attack (otherwise, you would have used a knowledge-based detection, with less headaches), the only thing they can rely on is their knowledge of the target system. But if you have completely lost the semantics of the target system when you applied the abstraction function AF, then you have also lost an important source of information, that can be very useful in deciding how to act upon an alerts. This is the reason why I like to distinguish two kind of ABNIDS, which I call blackbox and whitebox. - We call *blackbox* those ABNIDs that use abstraction function unrelated to the system's semantics, like n-gram analysis, neural networks, and alike. - We call *whitebox* systems those ABNIDS in which the abstraction function AF retains something of the high-level semantics of the target system. I would call whitebox IDS an IDS that would distinguish between *read* and *write* file access, and would be able to report an alert like "thesubstation Alpha is giving instructions to PLC Beta: this is an anomalous action as Alpha normally only reads data from Beta". (Aside: we started using the notation "whitebox ABNIDS" in [7], there is another reference to whitebox anomaly detection in [8], but that is about host-based detection, and is unrelated).

To start with an unprofessional statement:

Personal Opinion 1 *I believe that blackbox anomaly-based intrusion detection systems are of very limited use for security.*

This was noticed back in 2010 by Sommer and Paxson, [9], who wrote "We argue that when evaluating an anomaly detection system, understanding the target

system's semantic properties is much more valuable than identifying a concrete set of parameters for which the system happens to work best for a particular input". To this, we added some evidence in [10]. In our hands-on experience, the problem with blackbox (say n-gram-based) ABNIDS is that their actionability is zero: you get an alert and to find out what is going on you need to have a very skilled someone take a look with Wireshark. Is it interesting? What should be done about it? The information given with the ABNIDS warning was "the frequency distribution of this packet is abnormal". Whitebox detection here has a tremendous advantage: it tells you something about the semantics of the anomaly and what the target system was doing at the time of the alert, and the insight in the alert can be much more detailed, like there is a doctor breaking the glass 10 times in a day, the observed limit is 5.

In our search for usable anomaly-detection, we came to the conclusion that

Personal Opinion 2 *"Useful" anomaly-based intrusion detection is not quite about intrusion detection; it is about being able to understand what happens in the target system and being able to monitor its integrity.*

In our opinion, good anomaly detection starts by a good representation of the target system. A representation people can understand. You do not concentrate so much on the attack you need to discover (even though (***) has to be satisfied), but on explaining what happens. This brings it closer to specification based systems and to monitoring/forensics. By doing so, you'll have less difficulties (a) getting the IDS accepted at the stakeholder (it appears familiar) (b) providing actionable security when something goes wrong.

To give a concrete example: this is the recurring pattern of what typically happens in real-life deployments of a whitebox ABNIDS (based on the experience we have when deploying SilentDefense): the first thing we make is the model of the target system. This usually takes a couple of days of passive listening to the network traffic, and the application of our whitebox AF. Then we present the customer with the results. We haven't started doing anomaly detection yet, we have just learned the model. And by only producing a good model of the target system we have identified at least a dozen issues in the network that need to be solved and can be acted upon (notice that a blackbox model would not produce the same results). When this happens, we believe we are in presence of an anomaly detection system that (a) is "good", and (b) fits well the target system.

The downside of this approach is that anomaly-detection systems need to specialize to a particular domain, which is not only a particular network protocol but a particular set of applications of it. In addition, things might work well in a certain domain (e.g., Industrial Control Systems – ICS) but they might not work at all in another domain (e.g., IT). For instance, because the changes and intrinsic dynamism of a domain make a certain model obsolete too quickly. Our experience with SecurityMatters taught us that domain knowledge is crucial to success, in that for instance we "understand" very well domains such as energy distribution, oil and gas, etc. We have also learned how to approach a new domain, but each new domain requires adjustments and understanding.

Adaptability. Behaviour-based systems - regardless of whether they are anomaly-based or specification based - need by definition to be adjusted every time there is a change in the underlying system. This is a problem, even in a closed, relatively static setting like ICSs. There is a common misunderstanding that the network traffic (and the underlying settings) of Industral Control Systems does not change much in time. This is not true: there are continuous changes due to maintenance, replacement of parts, new functionalities, etc.; if a behavior-based system is connected, then it should have the ability of adjusting itself to these changes without raising a myriad of alerts. This requires providing facilities to the people who are in charge of the monitoring to distinguish the typical benign cases from the possibly malicious ones. Again, it comes down to understanding the application domain, and building some actionability into the system. This is yet another reason why - given how software is written today –

Personal Opinion 3 *There cannot be a one-size-fits-all anomaly-based network intrusion detection system that works equally well on all domains.*

Examples of "domains" are backoffice, webapplication, IoT, but also more specifically: Oil and Gas, Banking, Water companies. In short: ABNIDSs are always tailored to the target system.

This brings up the point of *Scalability.* Since ABNIDS are tailored to the target system, scalability is by definition an issue. To monitor 1000 networks, you need a thousand different models, that need to be trimmed when things change, etc. To monitor a smart city you have to monitor every single building, every single room etc.: there is no fixed recipe that fits all of them (as in the case of knowledge-based detection). The obvious conclusion is that this technology scales only up to a point, but areas like IoT, with thousands and thousands of different networks, will need a leap forward in our approach to monitoring.

Detection rate and false positives. FPs are the nightmare of researchers and practitioners alike because a high false positive rate (FPR) means that the IDS will not be looked at. Our experience in ICS confirms that it is usually possible to tune the system to find the "best" compromise between DR and FPR, though in our experience, in the case of whitebox anomaly detection this is done more by focusing on what is monitorable and disregarding what is "not monitorable", which are the parts for which it is simply impossible to make a reasonable model of the observables. In ICS, the "monitorable" part dominates, and we took advantage of that to engineer an effective NIDS; but if we look at e.g. a standard laptop, there is no way we could make a reasonable whitebox model of what happens in there. I want to address this in the next section, but before I do so it is time to touch on specification-based systems.

Specification based intrusion detection systems Here I need to say that I do not have enough first-hand experience about them to have a bold opinion, but it seems to me that they share with ABNIDS a lot of the pro's and the con's, with the added problem that producing the specification is usually very costly. I believe that one of the root problems with this technology is that - to

be effective – the specification should take into consideration the environment: the same system (say a PLC) can behave very differently when used in different contexts, and having the specification of the PLC in isolation is of little use for intrusion detection. On the other hand, providing a specification for each implementation is prohibitively expensive. Here ABNIDSs have a tremendous advantage over specification-based systems, because they learn the behaviour of the target system in the appropriate context. Additional (obvious) difficulties include dealing with changes in the systems and actionability. While I believe that specification-based NIDS form a very promising area, my personal opinion is that for the moment their applicability is limited to very specific domains, that are even more narrow that those to which we can apply anomaly-detection profitably.

Some Considerations on Intrusion Detection While we cannot say (yet) that whitebox ABNIDSs are successful in general, we have seen that they can be successful in monitoring specific systems and in particular we have experienced that *when* they are successful, the reason is usually that they manage to lift the understanding to the application level: by analysing the network trace they understand what the application is doing. That is where anomaly detection can be effective. In our specific case, achieving this required putting together a massive knowledge of the domain, and was possible because our target systems (ICS) are less confusing than e.g., standard computers. In fact, there is little hope that our method could be (economically) applied to e.g. the applications running in a modern laptop. This is because the network observables they exhibit are so complex, limited and confusing that you simply can't understand what is going on, let alone make a usable whitebox model of it (not to mention, deal with changes, which are the rule, rather than the exception).

3 Writing supervisable software

I now want to step away from the topic of intrusion detection and build on the above considerations to talk about software design. Giving for granted that software and systems will never be secure, as statistics and trends amply demonstrate, we have to focus on engineering resilient systems, and a large part of this resiliency lies in early understanding of when things go wrong. This is what an intrusion detection is supposed to do. Unfortunately, as the journey above indicates, I believe that there is little hope that intrusion detection will work on a global scale; it will always work on some sectors, some target areas, but there are large areas where they are ineffective or too expensive.

This is not surprising, if we consider that ICT systems are largely built as black boxes, and after building them we pretend that the monitoring system is able to detect when something goes awry. For some of those black boxes (the "simpler" ones) IDSs are able to do so, but when the black box is too complex inside or when there are too many of them connected together we lose control, and IDSs can only pick some meaningful indicators here and there and hope to

make the best of them. The global picture is then lost and in my opinion this is when IDSs stop being effective.

It also appears that complexity this is only going to get worse: on one hand the scale of the target systems is exploding (see IoT), on the other hand, we tend to try to make things "more secure" by making systems more unintelligible (e.g., by obfuscating and encrypting the observables), therefore making it harder to reconstruct the global picture.

To build resilient systems, I believe we need to change drastically the way we actually write software. Next to "security by design", we need something else:

Personal Opinion 4 *We should develop a discipline of writing software that is supervisable (and privacy-preserving) by design.*

I do not have (yet) a precise definition of what supervisable is. What I am advocating is a discipline more than a science, a discipline I believe we need to develop; with a lot of practical, hands-on work.

In general, I think that programs and systems should be designed to provide meaningful observables (including meaningful network observables), which should be suffcient for the instructed observer to understand:

(a) what the underlying applications are actually doing,
(b) if the system is actually doing what it is pretending to do,

and, ideally,

(c) what the system is failing to do,
(d) whether there is something wrong with the system, and how to react to it.

Privacy and data confidentiality are obviously very important concerns, and these points seem to oppose them. This is the reason why privacy is explicitly mentioned in the opinion above: supervisability and privacy/confidentiality cannot be considered as separate issues and need to be addressed together at design time. This can be done by separating the information regarding the working of the application from the information that needs to be kept confidential, and adopt different encryption strategies for them.

Personal Opinion 5 *Trying to achieve privacy by making the software not supervisable is in my opinion as wrong as trying to achieve security by obscurity.*

This is – I am afraid – a common engineering mistake: encrypting "everything" to stay on the safe side. Unfortunately, this often makes the system less supervisable, less manageable, it makes troubleshooting harder and in several cases it does not help security [11].

It is better to consider everything public, except for the confidential and the private information. In addition, I am not saying that everything should be monitored by everyone, but everything should be supervisable by something, and there should be something monitoring on it. Something trusted. Communication can be encrypted, when needed, and supervisors need to be able to decrypt the non-confidential parts to monitor the functioning of the system.

Getting back to the points above, point (a) advocates the use of observables with a clear semantics. This is a necessary condition to obtain (b), which is the key element. It states that the observables (and the communication) should be designed in such a way that it is difficult for a hypothetical attacker who has managed to subvert the target system to do anything without being noticed. I realize that in many cases this is impossible: televisions, servers etc. will always deal with gigabits binary data in which it is by definition easy for an attacker to embed his own payload. But there are other cases in which this is possible. I am thinking in particular at how we should deal with the software of smaller embedded systems and IoT devices. Point (c) goes a step further and encourages the engineering of systems with predictable behaviour and providing sufficient observables to allow one to determine whether they are actually operating correctly. As it happens, while point (b) argues for a minimization of the communication, point (c) makes a case for the opposite: that the number of observables should be sufficient to understand also when something is *not* happening. Finally, (d) touches on the idea that we should start thinking about how to do incident response right from the moment that we design the systems. It is very much "wishful thinking", but in the long run, it is probably unavoidable. It should be clear that what I called supervisable is reminiscent of but is very different from the concepts of monitorability as defined in runtime verification (e.g., [12, 13]), and the concepts of observability and diagnosability [14].

It may seem that I am advocating writing software for which it is possible to do specification-based intrusion detection. This is not quite true, for the same reason I mentioned earlier when discussing specification-based NIDS: the same artefact behaves (rightly) very differently when put in different contexts and I don't believe this variability can be captured by a specification (not a cost-effective one). I would happy to be contradicted. What I am advocating is writing software that allows to do monitoring it, possibly using a combination of techniques like those in anomaly-based detection, specification-based detection and correlation as is done in present SIM-SIEMS.

In this ideal world, software artefacts should be self-explanatory in their behaviour, and it should be straightforward to for the instructed observer to be able to understand what the system is actually doing by simply observing its network behaviour. Unfortunately, this is not the direction we are following, and despite the adoption of "standard protocols" when possible, confusion is the rule and clarity is the exception. Scalability remains an issue, which in my opinion can only be dealt with in the obvious way by breaking down a system into monitorable subsystems, etc.

I think this discipline is going to be indispensable in systems where solutions of different vendors and providers are combined together. Like it is happening in IoT. Liabilities in case of failure are probably going to play an interesting role in how systems will be shaped, and in my opinion a form of supervisability will be a necessary instrument to identify actual responsibilities and actions to be taken when things go wrong.

Acknowledgements Many, many thanks to those who have given comments to this paper, including: Luca Allodi, Elisa Costante, Marc Dacier, Guillaume Dupont, Davide Fauri, Dieter Gollmann, Alexios Lekidis, Daniel Ricardo dos Santos, Boris Skoric, Nicola Zannone.

This work has been funded by SpySpot, a project under Cyber Security programme by NWO, Dutch Organization for Scientific Research. It was also partly funded by IDEA-ICS project by NWO and U.S. Department of Homeland Security.

References

1. Debar, H., Dacier, M., Wespi, A.: A revised taxonomy for intrusion-detection systems. Annals of Telecommunications 55(7), 361–378 (2000)
2. Mitchell, R., Chen, I.R.: A survey of intrusion detection techniques for cyber-physical systems. ACM Computing Surveys (CSUR) 46(4), 55 (2014)
3. Ko, C., Ruschitzka, M., Levitt, K.: Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In: Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on. pp. 175–187. IEEE (1997)
4. Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., Zhou, S.: Specification-based anomaly detection: a new approach for detecting network intrusions. In: Proceedings of the 9th ACM conference on Computer and communications security. pp. 265–274. ACM (2002)
5. Ptacek, T.H., Newsham, T.N.: Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., DTIC Document (1998)
6. Siddharth, S.: Evading nids, revisited. Symantec Connect Community pp. 1–5 (2005)
7. Costante, E., den Hartog, J., Petković, M., Etalle, S., Pechenizkiy, M.: Hunting the unknown - white-box database leakage detection. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 243–259. Springer (2014)
8. Shu, X., Yao, D.D., Ryder, B.G.: A formal framework for program anomaly detection. In: International Workshop on Recent Advances in Intrusion Detection. pp. 270–292. Springer (2015)
9. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: Security and Privacy (SP), 2010 IEEE Symposium on. pp. 305–316. IEEE (2010)
10. Hadžiosmanović, D., Simionato, L., Bolzoni, D., Zambon, E., Etalle, S.: N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols. In: International Workshop on Recent Advances in Intrusion Detection. pp. 354–373. Springer (2012)
11. Fauri, Davide de Wijs, B., den Hartog, J., Costante, E., Etalle, S., Zambon, E.: Encryption in ics networks: a blessing or a curse? Tech. rep., Eindhoven Technical University (2017), to appear
12. Viswanathan, M., Kim, M.: Foundations for the run-time monitoring of reactive systems–fundamentals of the mac language. In: International Colloquium on Theoretical Aspects of Computing. pp. 543–556. Springer (2004)
13. Pnueli, A., Zaks, A.: Psl model checking and run-time verification via testers. FM 2006: Formal Methods pp. 573–586 (2006)
14. Bittner, B., Bozzano, M., Cimatti, A., Olive, X.: Symbolic synthesis of observability requirements for diagnosability. In: AAAI (2012)