ICSvertase: A Framework for Purpose-based Design and Classification of ICS Honeypots

Stash Kempinski
Eindhoven University of Technology
Secura
Eindhoven, The Netherlands
s.p.kempinski@tue.nl
stash.kempinski@secura.com

Shuaib Ichaarine
Eindhoven University of Technology
Eindhoven, The Netherlands
shuaib_ichaarine13@hotmail.com

Savio Sciancalepore
Eindhoven University of Technology
Eindhoven, The Netherlands
s.sciancalepore@tue.nl

Emmanuele Zambon
Eindhoven University of Technology
Eindhoven, The Netherlands
e.zambon.n.mazzocato@tue.nl

ABSTRACT

As attacks on Industrial Control Systems (ICS) are increasing, the design and deployment of ICS honeypots is gaining momentum as a way to prevent, detect, and research them. However, ICS honeypot creators hardly explicitly consider what adversary behavior they want to capture, potentially creating honeypots that may not completely fulfill their intended purpose. At the same time, ICS honeypots are classified using the traditional interaction level scheme which is unsuitable for ICS due to its unique properties. In turn, these issues make it hard for potential users to systematically determine the suitability of an ICS honeypot for their use case. To tackle these problems, in this paper we introduce *ICSvertase*, a novel framework allowing for structural reasoning about ICS honeypots. *ICSvertase* integrates several existing components from the ATT&CK for ICS and Engage frameworks provided by MITRE and extends them with novel elements. *ICSvertase* provides a novel approach to helping companies and users in several real-world use cases, such as choosing the most suitable existing ICS honeypot, designing new ICS honeypots, and classifying existing ones in a more fine-grained way. To show *ICSvertase*'s benefits, we provide examples for these real-world use cases and compare them to their traditional counterparts.

CCS CONCEPTS

• **General and reference → Design**; • **Computer systems organization →** *Special purpose systems*; *Embedded systems*; • **Networks → Network security**.

KEYWORDS

ICS Honeypot Selection, Classification Scheme, Cyber-Attack, Deceiving Technology, Active Defense

ARES 2023, August 29-September 1, 2023, Benevento, Italy

ACM Reference Format:
Stash Kempinski, Shuaib Ichaarine, Savio Sciancalepore, and Emmanuele Zambon. 2023. ICSvertase: A Framework for Purpose-based Design and Classification of ICS Honeypots. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29-September 1, 2023, Benevento, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3600160.3605020

1 INTRODUCTION

Using intentionally vulnerable systems, i.e., *honeypots*, in traditional IT deployments is a well-known method for gathering Cyber Threat Intelligence (CTI) or detecting adversaries in a network [24]. Honeypots are traditionally deployed to serve one of two main functionalities. The former is to support attack analysis. So-called research honeypots [9] are usually intentionally Internet-facing, provide (possibly weakened) IT services, and contain elaborate logging functionalities to capture adversarial behavior. The latter is to support intrusion detection. So-called production honeypots [9], usually consist of decoy hosts that provide alerting functionalities when meaningful interactions occur. Given that no legitimate interactions with these decoys should take place, they likely indicate an adversary being active on the network.

With attacks on Industrial Control Systems (ICS) becoming more prevalent [17], research on ICS honeypots is gaining momentum and progressing rapidly. It started with simple systems displaying static web pages of ICS devices [5] and evolved to more complex ones incorporating (simulated) physical processes to increase their credibility [3]. However, ICS honeypot creators hardly (in a structural fashion) consider what adversary behavior they want to capture. In some cases, this results in honeypots being created without (completely) fulfilling their intended purpose (see Sec. 5.1), or being overly complex for their purpose.

The reason that ICS honeypot creators do not (structurally) consider adversary behavior can be attributed to a lack of methods for structural reasoning about ICS honeypot requirements in general. When trying to design and/or classify ICS honeypots, this problem manifests itself in three ways. First, when creating an ICS honeypot, one cannot systematically determine the minimum required features to achieve the purpose of the honeypot. Second, the primary honeypot classification method is the interaction level scheme, which classifies honeypots based on complexity rather than features. This classification scheme also does not consider the physical and heterogeneous nature of ICS environments, which make traditional honeypot deployment techniques, such as emulation, non-trivial. To make things worse, the definition of the specific interaction levels differ significantly in literature. For instance, Guarnizo *et al.* define high-interaction honeypots as systems that do "not emulate any services, functionalities, or base operating systems" [10], whereas Antonioli *et al.* define them as "real services running on real Operating Systems (...) or simulate the services and the relevant parts of an Operating System" [1]. Third, to the best of our knowledge, there is no way to systematically determine the suitability of an existing (ICS) honeypot for a given use case. Consequently, interested parties cannot structurally establish what features an existing honeypot possesses and if these features match those required to achieve their intended purpose.

In this paper, we present *ICSvertase*, a framework named after the enzyme "invertase" which bees use to convert the complex sugar in nectar into the more simple ones that form honey. *ICSvertase* allows users to systematically address the aforementioned problems by (i) determining the minimum required features of an ICS honeypot for a specific use case; (ii) classifying (existing) ICS honeypots using a structured, fine-grained, and purpose-based approach; and, (iii) assisting and easing the task of choosing between existing ICS honeypots for a specific use case. In other words, our framework systematically answers the following key questions when creating and/or using an ICS honeypot: (i) what adversary behavior should a honeypot capture; (ii) how can a honeypot capture this adversary behavior; (iii) why would one want to capture such behavior; (iv) how to convince an adversary to perform such behavior; and (v) which existing honeypot is most suitable for a given deployment.

To do so, *ICSvertase* incorporates existing components of MITRE ATT&CK® for ICS and MITRE Engage™[18, 19], extends them to be more suitable in ICS and honeypot context, and introduces new concepts, such as a set of functional requirements for ICS honeypots and methods to determine them. To the best of our knowledge, no papers before used the components cited above in the ICS honeypot research domain. Note that *ICSvertase* specifically considers only ICS honeypots, due to their remarkable differences with IT honeypots, such as the need for modeling physical processes and the non-triviality of creating honeypots for ICS-specific services.

The paper is structured as follows. Sec. 2 reviews related work; Sec. 3 describes in detail MITRE's used components and the differences between IT and ICS honeypots; Sec. 4 presents *ICSvertase*, its building blocks and use cases; Sec. 5 provides examples of how to use *ICSvertase* and shows its improvement on the interaction level classification scheme and, finally, Sec. 6 tightens the conclusions.

2 RELATED WORK

Our work is not the first to propose solutions for the problems presented in the introduction. For instance, whenever a research paper introduces a new ICS honeypot, it uses a scheme to compare itself to existing literature [1, 4, 15, 25–27]. Such comparisons mostly follow a scheme that determines if the compared honeypots implement or possess a set of features in a "yes" or "no" (or comparable) fashion, possibly including an additional "partial" option. However, the features used for comparison differ vastly per paper. Moreover, they are seemingly chosen in such a way that highlights the novelty of the presented honeypot. They range from specific technical details, e.g, how complete an implementation is with regards to a specific asset, to general characteristics, e.g., if the project is still maintained or flexibility in configuration. As an example, consider the honeypots in HoneyPLC [15], HoneyVP [27], and the one proposed by Antonioli *et al.* [1]. HoneyPLC [15] includes a comparison based on technical features, e.g., TCP/IP stack simulation and ability to capture ladder logic (a commonly used programming language in industrial assets). HoneyVP [27] uses comparable features, but (among others) excludes the capturing of ladder logic (HoneyPLC's novelty) and includes R&D- and hardware-related costs (HoneyVP's novelty). Antonioli *et al.* [1] take a completely different comparison approach, using the interaction level of honeypots, if they are actively maintained, and their networking capabilities. *ICSvertase* proposes an independent method, which compares ICS honeypots primarily based on their purpose, rather than their features. We made this choice as different purposes naturally lead to a varying set of requirements and, in turn, features.

ICSvertase also introduces a new classification scheme, even though such schemes and taxonomies for (ICS) honeypots already exist [8, 9]. These contributions describe honeypots more generally, but also more broadly, than the previously discussed comparison schemes, which make them less suitable for fine-grained or feature-specific comparisons. However, their broader nature allows them to provide a more complete overview of feature categories. The differences in both types of schemes can clearly be seen in the taxonomy provided by Fan *et al.* [8], which includes both deployment considerations and methodologies that allow a honeypot to recognize, capture, or prevent attacks. This taxonomy identifies and groups feature categories, such as how to profile attacks (e.g., obtaining the tools an adversary uses), but does not describe specific methods to do so (e.g., capturing ladder logic). As the taxonomy provided by Fan *et al.* is intended for IT honeypots, they only mention ICS as a honeypot theme, while not providing related challenges and consideration. Franco *et al.* [9] provide a classification scheme that does include ICS-specific considerations, which they use for specifically surveying cyber-physical related honeypots (such as Internet of Things (IoT) and ICS). As a result, the relevant differences between the latter taxonomy and the one by Fan *et al.* lie in the consideration of ICS-unique properties, such as the inclusion of cyber-physical processes (e.g., through simulation). Note that Franco *et al.* also introduce a purpose classification, for which they use "research" and "production" as defined in the introduction of this paper. Both these classification schemes are broader than the scheme presented in this paper, as they consider more properties than just the purpose of an (ICS) honeypot. However, as *ICSvertase* proposes a purpose-based classification scheme for ICS honeypots, it is vastly more in-depth. In other words, *ICSvertase* can be seen as an intermediate approach between the existing classification and feature comparison schemes: it is detailed enough to make feature-specific comparisons, while being complete enough to classify ICS honeypots. Finally, we remark that *ICSvertase* goes further than only proposing a new classification scheme for ICS honeypots, it also proposes a way to design them based on this scheme. To sum up, *ICSvertase* not only allows its users to classify existing ICS honeypots, but also to design new ones based on their desired purpose. To the best of our knowledge, *ICSvertase* is the first framework that provides this two-way functionality for ICS honeypots.

3 BACKGROUND

As the differences between IT and ICS honeypots are significant enough to require their own research areas, we provide more insights into these differences in Sec. 3.1. Furthermore, our work uses components from two knowledge bases provided by MITRE: ATT&CK for ICS and Engage. We describe these components in Sec. 3.2 and Sec. 3.3, respectively.

3.1 IT vs ICS Honeypots

There are multiple key differences between IT- and ICS environments that introduce unique challenges when designing and deploying ICS honeypots. Cyber-Physical Systems (CPSs) (digital processes interacting with physical environments) are at the core of ICS environments, and their assets are more diverse than those in IT environments. A prime example of such assets are Programmable Logic Controllers (PLCs), multipurpose devices whose primary functionality is to interact with the real world based on their inputs (such as sensor readings) in a deterministic fashion. While IT assets run on commonly used Operating Systems (OSs) (e.g, Linux), the time-constrained nature of ICS assets usually require them to run on (predominantly) vendor-specific Real Time Operating Systems (RTOSs), i.e., specialized OSs that can guarantee time constraints. Similar considerations apply also for CPU architectures. IT systems use a generalized set of architectures (x86, ARM, etc), easily identifiable if desired (e.g., by looking up the CPU model), whereas for ICS assets this information is rarely publicly available.

As the services running on the mentioned assets are built specifically for their respective OS and CPU architecture, deploying a honeypot for such assets would require the same combination to ensure that said services run on the honeypot's hardware. Thus, we have mainly three options: (i) emulating the OS and/or CPU architecture, (ii) virtualizing the services intended to act as the honeypot, or (iii) using a real asset. The generalized usage of OSs and CPU architectures in the IT domain make emulation in most cases redundant as virtualization-capable components are widely (and relatively cheaply) available. The same holds for using real assets, as they can easily be repurposed by simply installing the desired OS and services. However, in the ICS domain none of the options are trivial. First, the emulation of embedded systems has been well researched by Zaddach *et al.* [28], who created an emulation method for embedded firmware and thoroughly discussed the challenges of real-time emulation. The heterogeneity of ICS assets and identified challenges make emulation either time-consuming or even non-feasible for ICS honeypots. Second, virtualization requires knowing the specific CPU architecture of the ICS asset and having a virtualization-capable CPU. None of the options are realistic, due to the unavailability of information and the hard feasibility of acquiring such a CPU (if they even exist). Last, ICS components can be significantly more costly than IT components. This is not necessarily an issue when creating a single honeypot incorporating a specific component. However, any changes involving said component, such as implementing an equivalent service from a different vendor, would likely imply the purchase of a new component.

A common work-around is to simulate the services of the respective asset. This approach is also common in IT honeypots, which only partly implement or imitate a service to take away some degree of freedom from the adversary (e.g, not giving full access to an asset over SSH). However, IT services predominantly use standardized protocols, whereas ICS often use proprietary ones, with no publicly available documentation. Thus, when creating an ICS honeypot, it might be required to first reverse-engineer the protocol(s) that the related services use. Note that there are exceptions, as both generalized ICS- and proprietary IT protocols exist.

Lastly, the implementation of a (seemingly) live environment can result in more convincing honeypots in both domains. The specific needs depend on the environment and services imitated by a honeypot; however, they usually consist of either (fake) human interactions or interactions between the respective processes. For example, a database that is being regularly modified or a tank holding varying amounts of water during the day both indicate that they are being actively used. In IT environments, these interactions and corresponding outcomes are processed as fast as possible (unless explicitly designed not to do so or if human actions are required); hence, for a IT honeypot to be convincing, its interactions should be processed at the same speed. However, for an ICS honeypot to provide the same level of realism, the physics involved in its environment also need to be considered. This extra consideration is necessary, as skilled adversaries would notice if a compromised system provides unrealistic interactions, e.g., a water tank emptying completely in milliseconds. As a result, ICS honeypots should implement some sort of (simulated) physical process, as it can play an essential part for its convincibility. All these considerations contribute to making the ICS honeypot domain unique.

3.2 ATT&CK for ICS

The ATT&CK for ICS knowledge base provides an overview of categorized adversary behavior and where this behavior can be detected [18]. This categorization consists of two sets: *tactics* and *techniques*. *Tactics* describe what an adversary wants to achieve at a certain step in their attack. *Techniques* describe broadly how an adversary can perform a certain *tactic*. For example, *initial access* (*tactic*) can be obtained through *external remote services* (*technique*).

To identify where this behavior could be detected, MITRE also provides two sets structured in the same way: *data sources* and *data components*. *Data sources* describe concrete information sources that, when configured and monitored correctly, can show indications of adversary behavior. *Data components* describe the specific part of a *data source* providing such indications. For example, an adversary performing *exploitation of remote services* can be seen in the *network traffic* (*data source*), when looking at the *network traffic content* (*data component*), if known payloads are used.

3.3 Engage

The Engage knowledge base provides an overview of interaction methods usable by defenders to reveal, influence, and learn about adversary behavior [19]. Engage's categorization consists of three sets: *goals*, *approaches*, and *activities*. *Goals* describe the overall intentions, i.e., to reveal, influence, or learn about the adversary. *Approaches* describe the intended outcome of these intentions. Lastly, *activities* describe the specific methods to steer an adversary towards this outcome. For example, if a defender's intention is to *affect* (*goal*) adversaries to *direct* (*approach*) towards a honeypot, they could use *introduced vulnerabilities* (*activity*) to attract them.

Next to these sets, Engage provides a mapping between the *techniques* from the non-ICS ATT&CK framework and its *activities*, which can be used to determine what *activities* are able to capture, and thus interact with, certain adversary behavior. To create this mapping, Engage uses a set of *adversary vulnerabilities* that describe what an adversary is susceptible to when showing certain behavior, and mapped them to their relevant *activities*. For example, an *adversary vulnerability* is that when adversaries collect data, they are susceptible to collecting fake data. In turn, this *adversary vulnerability* can be used to create the following mapping: when an adversary uses *automated collection*, the data that they collect can be influenced through *information manipulation*.

The mapping process consists of checking per *technique* which *adversary vulnerabilities* are applicable, then check for each related *activity* if it is relevant for the *technique* in question. This process creates a set of {*technique*, *adversary vulnerability*, *activity*} pairs that are then reduced to unique {*technique*, *activity*} pairs. We leverage this same approach for mapping the *techniques* of ATT&CK for ICS to Engage's *activities* in Sec. 4.3.

4 ICSVERTASE

ICSvertase provides a structural way for designing and classifying ICS honeypots based on their purpose. To this aim, *ICSvertase* uses the following components (and their respective mappings) from MITRE's knowledge bases: Engage's *approaches* and *activities* and ATT&CK for ICS' *techniques* and *data components*. Specifically, *ICSvertase* uses the *approaches* and *techniques* to answer the "what" and "why" of an ICS honeypot's purpose, while it uses the *activities* and *data components* to answer the "hows" of an ICS honeypot [1].

Next to using MITRE's components, we introduce two new components to answer the questions posed in the introduction. They represent the functional characteristics and other considerations of an ICS honeypot, which we name *functional features* and *non-functional considerations*. Together with the *data components* they form the possible design requirements of an ICS honeypot. To systematically determine its specific set of design requirements we also introduce two new mappings. The first, named *Engage Adjusted for ICS*, bidirectionally maps Engage's *activities* to ATT&CK for ICS' *techniques*. We created this mapping specifically for *ICSvertase* as, at the time of writing, MITRE does not provide one themselves. The second new mapping, named *feature requirements*, provides the functionality to determine the *functional features* of an ICS honeypot using the previously mentioned components. Together, these new and existing components and mappings make up the building blocks of *ICSvertase*. Fig. 1 shows a schematic representation of how these building blocks are used in each of its use cases.

[1]Note that, as these knowledge bases are updated periodically, *ICSvertase* uses the first version of Engage and version 12 of ATT&CK for ICS.

The rest of this section describes each building block of *ICSvertase*. Specifically, Sec. 4.1 introduces the *functional features*, Sec.4.2 describes the *non-functional considerations*, Sec. 4.3 details how we adjusted Engage for our work, Sec. 4.4 provides the mapping between techniques and activities, and finally, Sec. 4.5 describes in more detail how all the building blocks work together.

4.1 Functional Features

ICSvertase defines a set of four technical feature categories that must be considered when designing (and implementing) an ICS honeypot. We extracted these features from both ICS- and IT honeypots, and their relevant literature, which either implicitly consider or explicitly discuss them. These features were then filtered based on their relevance to adversary interactions, e.g., alerting and data visualisation are excluded. Note that *ICSvertase* does not consider such features explicitly, but they are abstracted to *approaches* (in these two specific cases to *expose*) during the design process. We define the following functionality-related features, named *functional features*: *size*, *ICS component*, *physical process*, and *logging*. Each *functional feature* consists of multiple options that describe its implementation details, e.g., the need for single or multiple network entities (see the *size* feature). Note that some options are divided once more in sub-options when they require more granularity, in this case we refer to them as primary and secondary options respectively. In the following, we use the notation *feature:{options}* to denote the combination of a given feature and (set of) option(s), where the denoted options are always from the most granular set. Fig. 2 shows the *functional features* and their options.

4.1.1 Size. This feature describes the honeypot's network-accessible assets. It represents the amount of assets the honeypot (seemingly) consists of from a network perspective (not on the number of physical machines, as multiple virtual hosts can be running on a single physical machine). *ICSvertase* distinguishes two self-explanatory options: *single*, and *multiple*. Note that we intentionally avoid the term "honeynet", as this is also the name of the honeypot created by Cisco [21]. This feature is most relevant for more complex honeypots, e.g., the ones intending to capture lateral movement. For example, the honeypot created by Antonioli *et al.* consists of a VPN endpoint, gateway, and two PLCs [1].

4.1.2 ICS component. This feature describes how a honeypot can integrate ICS-specific component(s). To provide a fine-grained way to reason about possible implementations, *ICSvertase* distinguishes two primary options: *real device* and *imitation*.

The *real device* primary option has no secondary options and indicates that a real device is used as (part of) the honeypot. An example of a honeypot using a real device is HoneyVP [27], which combines a virtual component and a real PLC that handles requests that the virtual component is unable to process.

The *imitation* primary option includes five secondary options, describing features of a real device that can be imitated to some extent: *protocol*, *runtime*, *OS*, *bootloader*, and *system*. Depending on a honeypot's purpose, multiple options can be implemented simultaneously and at different levels of realism. For instance, one can use a basic web interface (*protocol*) with a more elaborate service implementation (*runtime*) to increase the honeypot's believability.

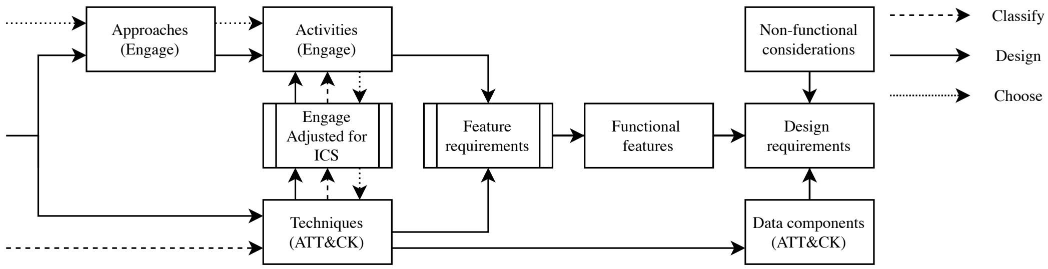

Figure 1: High-level architecture and steps for each of *ICSvertase*'s use cases.

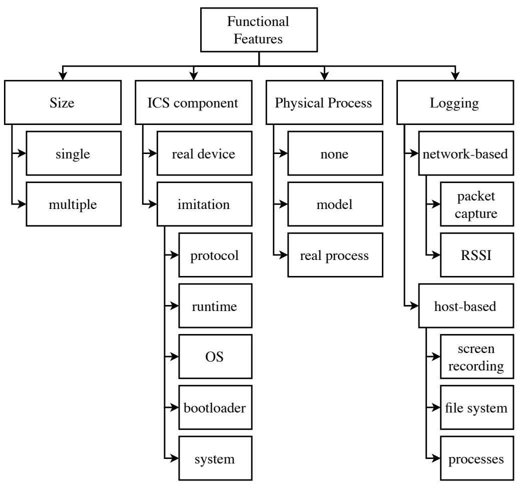

Figure 2: The defined functional features, including their primary and secondary options.

The *protocol* secondary option describes a honeypot adhering to a protocol specification in a static manner. This option consists at most of a finite state machine to trick an adversary into thinking that there is some form of dynamic interaction possible. An example of such a honeypot is S7commTrace [26], which implements Siemens' S7Comm protocol with static responses, i.e., valid requests to change the process status or the PLC configuration are not reflected in further communication.

The *runtime* secondary option describes a dynamic process whose execution can be influenced by, or observed through, adversarial interaction. In the context of ICS this means that, for instance, an adversary can create new program blocks or make adjustments to existing programs or parameters that are handled in a stateful way by the honeypot. Note that this option is not limited to influencing control logic, if the imitated ICS component is for example an Human Machine Interface (HMI), the runtime can be a VNC server (a remote desktop service commonly used in industrial settings) that shows the state of a physical process. An example of such a honeypot is Mimepot [3], which implements a water distribution system simulation whose operational parameters can be influenced via its Modbus/TCP server.

The *OS* secondary option describes the implementation of an OS capable of process execution. Note that this option usually includes the need for a complete file system to convince adversaries that they are interacting with a real OS, rather than an imitation. In practice, this option always requires the use of some sort of firmware emulation or virtualization. To the best of our knowledge, there are no ICS honeypots implementing this option.

The *bootloader* secondary option describes the possibility to update and execute a new firmware image on the imitated device. A honeypot implementing this option would, e.g., be able to capture the malicious firmware uploaded to the serial-to-Ethernet converters in the 2015 Ukrainian power grid attack, which prevented operators to remotely enable the electrical substations disabled during the attack [14]. To the best of our knowledge, there are no ICS honeypots implementing this option.

The *system* secondary option describes the imitation of a real device's hardware properties, such as memory layout/size and CPU architecture. Other properties include the (number of) I/O ports and serial communication interfaces, and being able to physically disable its programming mode. Indeed, a complete system imitation would be indistinguishable from a real ICS asset from a network perspective. Note that this option goes beyond acting as a real device by giving desirable responses to fingerprinting queries (which would be categorized under the *protocol* option). To the best of our knowledge, there are no ICS honeypots implementing this option.

4.1.3 Physical Process. This feature describes the awareness of the physical process(es) underlying the (imitated) ICS environment. *ICSvertase* distinguishes three options: *none*, *model*, and *real process*. The *none* option is added for completeness and describes a honeypot that does not implement any form of physical process. A honeypot without physical process awareness would either provide static answers to read requests, not respond in a meaningful way to command messages, or it would not adhere to physics when determining variable changes.

The *model* option describes the use of a model to simulate a physical process. This option does not differentiate between simple or complex models, but its implementation should add a meaningful layer of believability to the honeypot. An example of such a honeypot is Mimepot [3], which uses a mathematical model to determine the state of a simulated water distribution system.

Finally, the *real process* option describes the implementation of a real physical process. Note that this also implies the usage of a real ICS asset. To the best of our knowledge, the only example of a honeypot implementing this option is the one created by Hilt *et al.* [11], containing a mixture of simulated and real processes.

4.1.4 Logging. This feature describes how a honeypot records adversary activities. Although this feature does not affect the believability of a honeypot, it is essential to determine to what extent adversary behavior can be captured. *ICSvertase* distinguishes two primary options for this feature: *network-based*, and *host-based* logging. The *network-based* logging option indicates capturing adversarial actions observable from the network hosting the honeypot. It includes two secondary options: *packet capture* and *RSSI*. The *packet capture* secondary option is straightforward and often seen in honeypots. The *RSSI* option (referring to the logging of received signal strength of wireless messages) is, to the best of our knowledge, not present in any ICS honeypot currently available, but only in IT [23]. However, given that several known ICS attacks are using wireless protocols [12], this option can provide valuable insights into the physical location of an adversary.

The *host-based* logging option consists of logging activities not derivable from the network communication between the honeypot and the adversary. It contains three secondary options: *screen recording*, *file system*, and *processes*. The *screen recording* secondary option defines the recording of an ICS device's visual input/output. This option is only relevant to honeypots capable of such output, e.g., HMIs. As adversary interaction with these devices can provide valuable insights into their behavior, its logging must be considered. An example of such a honeypot is the one of Hilt *et al.* [11], which starts recording the workstation's screen when detecting significant visual changes.

The *file system* secondary option defines the logging of actions related to file creation, deletion, and modification. Capturing such actions and their outcomes gives insights into both the adversary objectives when interacting with files (e.g., deleting crucial files) and the interaction steps (e.g., changing parameters in configuration files). An example of such a honeypot is HoneyPLC [15], which captures control logic uploaded via the S7comm protocol.

The *processes* secondary option defines the capability of a honeypot to log actions related to process start, stop, and interactions with the OS. Logging such actions allows honeypots to identify activities such as adversaries' reconnaissance, e.g., starting system-native network discovery processes, or detecting how malware interacts with a system, by hooking OS API functions. To the best of our knowledge, there are no ICS honeypots implementing this option.

4.2 Non-Functional Considerations

Next to the functional features described in the previous section, there are several considerations regarding ICS honeypots that impact their design but do not directly relate to their functionality. These considerations remain from the extraction process (see previous section) and we call these *non-functional considerations*. Non-functional considerations are explicitly described here as they can vastly influence the effectiveness of a deployed honeypot. Note that these considerations are not identified through any of *ICSvertase*'s mappings, but must be considered by the users themselves.

4.2.1 External Persuasion. Methods other than just the honeypot itself can be used to lure adversaries or to increase the believability of a honeypot. These methods can take many forms. For instance, by setting up a website and creating social media profiles of employees, it is possible to trick adversaries into thinking a honeypot is real and belonging to an actual organization, as done by Hilt *et al.* [11]. Other methods in this category include posting information regarding the honeypot on hacker forums and other mediums commonly used by attackers to look for vulnerable devices, as done e.g., by Sasaki *et al.* in [22].

4.2.2 Hosting Location. The hosting location of an ICS honeypot can vastly influence its credibility. Hence, a convincing location is crucial for its purpose both from an external (Internet) and internal network perspective. For instance, whereas it is normal for IT services to be hosted at Cloud providers, experienced attackers would realize that it is unlikely for an ICS. The same holds for all publicly-identifiable IP addresses, such as those of universities. This consideration is important for both Internet-facing honeypots and for honeypots whose purpose is to detect adversaries on the internal network. For example, the deeper honeypots are placed inside a network, the less likely it is for adversaries to stumble upon them, potentially impacting their effectiveness.

4.2.3 Deployment Period. The (planned) deployment duration of a honeypot must also be considered. We include this consideration as it is usually not guaranteed that any adversarial interaction is captured, even if the honeypot is deployed for a long time. This is even more true for ICS honeypots as "the lifecycle of a sophisticated ICS attack is often measured in years" [16]. Depending on other features and considerations, e.g., if its deployment location has a monthly fee, this consideration can significantly impact the feasibility of a honeypot project.

4.3 Engage Adjusted for ICS

To tailor Engage to our needs, we created a mapping from ATT&CK for ICS' *techniques* to Engage's *activities*. This mapping follows nearly the same methodology used by MITRE for the mapping between ATT&CK and Engage (see Sec. 3). Compared to the reference methodology, we removed the email-related *adversary vulnerability*, due to its (low) relevance for ICS honeypots. Note that we consider malicious email attachments to be malware, which is considered through other elements of Engage. The mapping can be found on the reference *ICSvertase* GitHub page at [2].

Next to the adjustment in the mapping methodology, we made two minor adjustments to the Engage matrix itself to better fit it to the purpose of *ICSvertase*. First, we removed *Email Manipulation* from the set of *activities* for the aforementioned reason. Second, we added the *Network Analysis* and *Network Monitoring activities* to the other's original *approaches*, as both *activities* apply to the *Collect* and *Detect approaches*, depending on the details of their implementation. For instance, *Network Monitoring* relates, among others, to identifying anomalous traffic patterns. The identification of such patterns can be used both as threat intelligence (*Collect*) and to identify adversaries within a network (*Detect*). The adjusted matrix is shown in Fig. 3.

To increase the suitability of Engage in the context of ICS honeypots, we extend the definitions of the *Reassure* and *Motivate approaches*. *Reassure* is extended to convincing adversaries that they are interacting with an actual physical process, while *Motivate* is extended to convincing adversaries that they are interacting with a legitimate system. A practical example is the difference between the classification of Mimepot and HoneyPLC. HoneyPLC does not support physical interaction but imitates a commercially available device (Siemens PLC), *motivate*-ing an adversary to try Siemens-specific vulnerabilities. On the other hand, Mimepot does do simulation, but over a "brandless" Modbus/TCP device, *reassure*-ing an adversary that they are interacting with a real physical process. Note that, grammatically, the names of these *approaches* are interchangeable; hence, it is important to consider them using their Engage definitions.

Lastly, we explicitly decided not to add any new ICS-specific *activities* to the adjusted matrix as the existing *activities* can be interpreted in such a way that they fully consider the physical part of ICS. For instance, the *activity Information Manipulation* "is used to support the engagement narrative and directly impact adversary activities" [19]. This *activity* can be used, possibly jointly with *Pocket Litter* and *Peripheral Management*, to indicate implementation of physical processes in ICS honeypots, e.g., through simulation.

Expose		**Affect**			**Ellict**	
Collect	**Detect**	**Prevent**	**Direct**	**Disrupt**	**Reassure**	**Motivate**
API Monitoring	Introduced Vulnerabilities	Baseline	Attack Vector Migration	Isolation	Application Diversity	Application Diversity
Network Monitoring	Lures	Hardware Manipulation	~~*Email Manipulation*~~	Lures	Artifact Diversity	Artifact Diversity
Software Manipulation	Malware Detonation	Isolation	Introduced Vulnerabilities	Network Manipulation	Burn-In	Information Manipulation
System Activity Monitoring	Network Analysis	Network Manipulation	Lures	Software Manipulation	~~*Email Manipulation*~~	Malware Detonation
<u>*Network Analysis*</u>	<u>*Network Monitoring*</u>	Security Controls	Malware Detonation		Information Manipulation	Network Diversity
			Network Manipulation		Network Diversity	Personas
			Peripheral Management		Peripheral Management	
			Security Controls		Pocket Litter	
			Software Manipulation			

Figure 3: The adjusted Engage matrix for ICS. Changed *activities* are in italics, those underlined are added to the respective *approaches* and those crossed out are removed.

4.4 Feature Requirements

This section describes the mapping of *techniques* and *activities* to the *functional features* used by *ICSvertase* to systematically determine the feature requirements of an ICS honeypot, together with the decisions and assumptions made during its creation.

Our mapping contains a set of minimum required features to capture a given *technique* while, at the same time, not allowing an adversary to realize that they are interacting with a honeypot. Given the nature of *techniques*, it is hard to reason about all the possible capture methods; hence, we map a lower bound. For instance, a honeypot capturing *automated collection* does not necessarily need logging capabilities if it aims to detect collection attempts; however, it might require *file system* logging if it aims to detect what is being collected. Therefore, if required for its purpose, a honeypot might need to implement more (advanced) features than only those provided by this mapping. However, implementing less (advanced) features would indicate a honeypot unable to completely fulfill its purpose. Hence, through this mapping, users are made aware of the lower bound, ensuring that the honeypot can fulfill its purpose.

The mapping relies on two *logging*-based assumptions. First, *logging* options are only mapped to the *Collect* and *Detect activities*, as only these *activities* relate to capturing adversary behavior. Second, when an *activity* is mapped to the *ICS component protocol* option, we assume that the relevant logging is implemented in the imitating scripts, due to the triviality of such an operation. From *runtime* onward, we assume this logging is not necessarily present due to its implementation not being trivial (e.g., when using device-specific firmware), and thus relevant *activities* are mapped. For example, *packet capture* is not necessary with *protocol* implementation scripts as these will read packet contents anyways and can log the relevant data. An exception to this assumption occurs when the honeypot consists of multiple network-connected components, or the *technique* in question requires by definition communication between assets (and the honeypot possibly being only one of those assets).

Furthermore, we made two *physical process*-related assumptions. First, we mapped *physical process* options only to the *Reassure* and *Motivate activities*, as only these *activities* relate to convincing adversaries that they are in a legitimate environment. Second, we mapped to *physical process* options only *techniques* of the following *tactics*: *Collection*, *Inhibit Response Function* and *Impair Process Control*, as only they are relevant to process control.

Finally, we mapped no *techniques* from the *Impact tactic* as these *techniques* describe outcomes of adversary actions rather than being actions themselves. In other words, if the purpose of a honeypot is to capture *Impact techniques*, the honeypot can achieve such objective by capturing the *techniques* that lead to the impact.

For the sake of presentation, Tab. 1 only shows a snippet of the resulting matrix, the complete matrix can be found on the *ICSvertase* GitHub page [2]. Note that, in the matrix, we omitted the *none* secondary option from the *physical process* options as it is implicit for *activities* not requiring this feature. The matrix reads as follows: each row contains a *technique* and its relevant *activities*, and these *activities* are placed in the column that defines their minimum implementation requirement.

4.5 Use Cases

In this section we describe how users can employ *ICSvertase* to address each of its motivating use cases. We provide an example for each use case in Sec. 5.

4.5.1 Designing an ICS Honeypot. As shown in Fig. 1 via straight arrows, this use case consists of four steps and starts by identifying the purpose of the honeypot, through two parallel tasks. The first task requires users to identify the *techniques* that make up the adversary behavior they want to capture. The second task requires users to identify the *approaches* that make up what they want to do with the captured behavior and how/if they would like to convince the adversary to perform this behavior. Then, *ICSvertase* uses the identified *techniques* and *Engage Adjusted for ICS* to form a preliminary set of *activities*. The user should match these with the identified *approaches* and filter those they want to use in their honeypot. Third, *ICSvertase* uses the identified *techniques* and *activity* set to form the minimum feature set, through the feature requirement matrix. This is done by identifying the columns referenced by the *activities* for each *technique*. Last, *ICSvertase* provides the honeypot (minimum) design requirements, by combining: (i) relevant *data components* of

Table 1: Snippet from the feature requirements matrix.

Technique	Size		CPS integration				
	Single	Multiple	Protocols	Runtime	OS	Bootloader	System
Detect Operating Mode	*All*		API Monitoring, Information Manipulation, Lures	Software Manipulation, System Activity Monitoring			Hardware Manipulation, Security Controls
Device Restart/Shutdown	*All*		API Monitoring	Introduced Vulnerabilities, Security Controls, Software Manipulation, System Activity Monitoring	Malware Detonation		

the identified *techniques*, whose mapping is provided by MITRE; (ii) identified feature set; and (iii) non-functional considerations.

4.5.2 Classifying an Existing ICS Honeypot. As shown in Fig. 1 via dashed arrows, this use case consists of three steps and starts with the user identifying the *techniques* a honeypot captures. Optionally and/or as a coherence check, users can confirm the set of identified *techniques* by checking if the relevant *data components* are implemented. Then, *ICSvertase* uses the identified *techniques* and the *Engage Adjusted for ICS* mapping to form a set of *activities* possibly used by the honeypot in question. The user should filter this set of *activities* to those actually being used, through the *activities*' definitions. Last, the user should use this filtered set of *activities* to determine the honeypot *approaches*, which form the ICS honeypot classification. An possible future benefit of this use case is that, if both existing and future ICS honeypots would be classified using this scheme, users could more quickly determine the suitability of an ICS honeypot by means of the next use case.

4.5.3 Choosing Between Existing ICS Honeypots. As shown in Fig. 1 via dotted arrows, this use case consists of three steps. It relies on *ICSvertase*'s classification to be performed first, and for users to have decided on a purpose for their honeypot. First, the user should decide which *approaches* match the specific use case, filtering existing ICS honeypots based on the *approaches* they support. Then, the user should filter the *activities* of the matched ICS honeypots to further narrow down their suitability. Last, the user should compare the *techniques* of the remaining ICS honeypots to determine which of them serves their purpose best.

Note that it might be hard to find a honeypot that perfectly matches this purpose. However, each step systematically identifies honeypot(s) closer to the user's intended purpose by making them explicitly consider what features they are (and are not) looking for, constituting the most suitable starting point(s) for further development or customization.

5 ICSVERTASE USE CASE EXAMPLES

In this section we showcase *ICSvertase* to address the use-cases described in Sec. 4.5.

5.1 Designing a New Honeypot

To show the advantages of our framework for honeypot design, we reason on the design of the honeypot CryPLH [5] using *ICSvertase*. The authors of CryPLH state that their goal is to "develop a high-interaction honeypot which appears identical to the real device from an attacker's point of view" and its purpose is that it "needs to be able to log all the actions an attacker takes, while trying to exploit the PLC". This goal and purpose matches the *motivate* and *collect approaches*, respectively. CryPLH's authors use a Siemens Simatic S7-300 PLC as reference device, and mimic it as much as possible using four different methods. First, CryPLH scrapes the ethernet-accessible protocols (HTTP(S), SNMP, S7comm) provided by this PLC and uses the collected data to craft static replays that are sent at (adversary) request; all of the honeypot responses are static, except for a small set of SNMP messages which include some form of state awareness. Second, CryPLH uses a HTTPS certificate identical to the self-signed certificate of the PLC. Third, where necessary and possible on the simulated system, CryPLH is configured to match the PLC's properties, such as its non-standard MTU size. Lastly, when receiving any applicable authentication-request commands, CryPLH returns an "incorrect password"-error. In other words, using MITRE's terminology, the authors of CryPLH create a honeypot that appears identical to the real device by deceiving adversaries that use *remote system information discovery* through its *information manipulation* and *application diversity activities*. Furthermore, they implement a logging system that captures the honeypot's network traffic (*network analysis*) and is deployed as an *internet accessible device*. As no interaction is possible beyond unauthenticated information requests and stateless network messages, no exploitation *techniques* (e.g., *exploit public-facing application*) are identified, as adversaries cannot execute these successfully.

By using *ICSvertase*, we notice a design gap when reasoning about the required *techniques* and *activities*. Namely, during *ICSvertase*'s first parallel task, we explicitly identify *techniques* related to the various ways of exploiting a PLC. Tab. 2 (second row) shows a small but (for this example) sufficient set of *techniques* matching CryPLH's intended purpose. In turn, during step two of the design process, we identify the *activities* from the *collect approach* needed to serve CryPLH's intended purpose. Tab. 2 compares the *techniques* and *activities* required to support the cited objective to those implemented. We notice that the difference is significant: 4 *activities* and at least 3 *techniques* are missing in [5]. Note that we omit *non-functional considerations* from Tab. 2 as CryPLH's authors were knowingly limited in their options, and such limitations are likely the case for most researchers.

The missing *techniques* and *activities* show that CryPLH at least partially fails in "being able to log all actions an attacker takes while trying to exploit the PLC", as the authors primarily focused on making it appear "identical to the real device from an attacker's point of view". Indeed, while for some of *ICSvertase*'s newly identified *techniques* and *activities* it can be argued that their absence does not necessarily mean that CryPLH does not fulfill its purpose, their complete absence does. Namely, neither of the two *techniques* capturable by CryPLH relate to exploiting a PLC.

Table 2: Comparison between CryPLH's original implementation in [5] and *ICSvertase* suggested implementation.

Impl.	Step 1 Approaches	Techniques	Step 2 Activities	Step 3 Functional Features	Data Components
CryPLH	Collect Motivate	Internet Accessible Device Remote System Information Discovery	Network Analysis Information Manipulation Application Diversity	*Size*:single *ICS component*:protocol *Physical process*:none *Logging*:packet capture	Software Network Traffic
ICSvertase	Collect Motivate	Internet Accessible Device Remote System Information Discovery Brute Force I/O Execution Through API Modify Controller Tasking ...	Network Analysis Information Manipulation Application Diversity API Monitoring System Activity Monitoring Artifact Diversity Malware Detonation	*Size*:single *ICS component*:OS *Physical process*:none *Logging*:{file system, processes, packet capture}	Software Network Traffic Application Log Logon Session Network Traffic OS API Execution ...

5.2 Classification of Existing Honeypots

ICSvertase can also be used to classify existing ICS honeypots based on their purpose, better differentiating existing solutions than the traditional interaction level scheme. Tab. 3 shows the classification obtained using *ICSvertase* aside to the self-assessed interaction level of the reported solutions, taken from the related papers. The outcomes of each step taken to perform each classification can be found in Tab. 4. We also considered, but not included, the following honeypots due to a lack of confirmable or mappable information: Pliatsios *et al.*[20], Krasznay *et al.*[13], Dodson *et al.*[7], and Dipot[6].

An example highlighting the problem with classifying ICS honeypots purely based on their interaction level can be observed when comparing the honeypot of Antonioli *et al.* [1] and S7CommTrace [26] (Tab. 3). The authors self-classified their honeypots as being *high* interaction. Based on this, one could think the two solutions to be comparable. However, *ICSvertase* shows a clear difference, not only in the *approaches* they serve, but also in the supported *activities* (one overlap) and *techniques* (no overlap) (Tab. 4).

Another example highlighting inconsistencies with interaction-level-based classification emerges when comparing HoneyPLC [15] and CryPLH [5]. The respective authors (self) determined different interaction levels for their honeypots (*medium* for the former and *high* for the latter). Based on this, one could think the two solutions to be more different than, e.g., the two considered in the previous example. However, applying *ICSvertase* for classification we see they support the same *approaches*: they both *collect* information and *motivate* the adversary to target the honeypot by providing some sort of realism. Note that the interaction level's complexity indication is not lost in *ICSvertase*'s classification. It also shows the difference in their complexity when observing their mapped *activities*, which is part of the classification process (see Tab. 4).

Classifying honeypots by means of *ICSvertase* allows us to compare ICS honeypots by considering both their intended purpose and to a certain extent their complexity, but in a more natural (and objective) way than the interaction-level approach.

5.3 Choosing an Existing Honeypot

Consider a CTI organization that would like to *collect* PLC malware samples by creating a convincing honeypot that *motivates* adversaries in deploying their malicious code. To prevent potential double work, the organization would like to investigate existing honeypots to see if they are sufficient for the organization's purpose or need to be extended.

Table 3: ICS honeypots' author-classified interaction level vs ICSvertase's classification.

Honeypot	Interaction level	Approaches
HoneyPLC[15]	Medium	Collect, motivate
SIPHON[a] [10]	High	Collect, detect, prevent, reassure
Antonioli *et al.*[1]	High	Collect, reassure, motivate
HosTaGe [25]	Low	Collect, detect
Mimepot [3]	-	Detect, direct, reassure
LOGistICS [4]	Medium	Detect, direct, motivate
HoneyVP [27]	High	Detect, motivate
CryPLH [5]	High	Collect, motivate
S7CommTrace [26]	High	Collect

[a]SIPHON's example implementation is used here for its classification.

The organization uses the two *approaches* previously identified as a starting point for its investigation. Using the classification provided by *ICSvertase*, the organization identifies three viable options: HoneyPLC [15], CryPLH [5], and Antonioli *et al.* [1].

Using *ICSvertase*'s existing mapping, the organization can narrow down the options further by looking at their mapped *techniques* and *activities*, which can be found in Tab. 4. *Activities* matching the organization's purpose are *API monitoring*, *software manipulation*, *system activity monitoring*, *application diversity*, and *information manipulation*. Moreover, suitable *techniques* to capture malware samples are: *modify program* and *download program*.

These *activities* and *techniques* match the most with those supported by HoneyPLC. Although this match is not perfect, HoneyPLC can serve as starting point for the CTI organization.

We highlight that, according to the reference paper in [15] and to the traditional classification approach, HoneyPLC is a *medium* interaction honeypot. Contrasting the common sense that would suggest that *high* is better than *medium*, the available *high* interaction honeypots do not fit the requirements as tightly as the *medium* interaction honeypot. Thus, *ICSvertase* not only provides a systematic way of addressing honeypot selection, but might also help reducing costs (assuming a high interaction honeypot is more expensive than a medium interaction honeypot to acquire).

6 CONCLUSION

In this paper, we presented *ICSvertase*, a framework for purpose-based design and classification of ICS honeypots. Our framework addresses the lack of methods to structurally reason about ICS honeypots. It uses components and mappings from MITRE's ATT&CK for ICS and Engage knowledge bases in combination with newly

Table 4: ICS honeypots classification step outcomes.

Honeypot	Identified techniques	Activities	Approaches
HoneyPLC	Modify Program, Program Download, Internet Accessible Device, Graphical User Interface, Point & Tag Identification, I/O Image	API Monitoring, Software Manipulation, Application Diversity	Collect, Motivate
SIPHON	Valid Accounts, Internet Accessible Device, Graphical User Interface	Lures, Security Controls, Network Analysis, Network Monitoring, Peripheral Management, System Activity Monitoring	Collect, Detect, Prevent, Reassure
Antonioli *et al.*	Adversary-in-the-Middle, Block Command Message, Block Reporting Message, Brute Force I/O, Command-Line Interface, External Remote Services, Monitor Process State, Network Connection Enumeration, Network Sniffing, Point & Tag Identification, Remote System Discovery	Information Manipulation, Network Analysis, Network Monitoring, Security Controls, Network Diversity, Lures, Application Diversity, System Activity Monitoring	Collect, Reassure, Motivate
HosTaGe	Lateral Tool Transfer, Remote Services, Remote System Discovery, Remote System Information Discovery	API Monitoring, Network Analysis, Lures	Collect, Detect
Mimepot	Adversary-in-the-Middle, Block Command Message, Block Reporting Message, Spoof Reporting Message, Unauthorized Command Message, Remote System Discovery, Network Connection Enumeration	Attack Vector Migration, Information Manipulation, Network Monitoring, API Monitoring, Peripheral Management, Lures	Detect, Direct, Reassure
LOGistICS	Dervice Restart/Shutdown, I/O Image, Internet Accessible Device, Monitor Process State, Service Stop, Unauthorized Command Message	Introduced Vulnerabilities, Security Controls, Network Analysis, Application Diversity	Detect, Direct, Motivate
HoneyVP	Automated Collection, Brute Force I/O, Detect Operating Mode, Execution through API, I/O Image, Internet Accessible Device, Modify Alarm Settings, Modify Controller Tasking, Modify Parameter, Modify Program, Monitor Process State, Program Download, Service Stop, System Firmware, Unauthorized Command Message	Network Analysis, Application Diversity, Artifact Diversity	Detect, Motivate
CryPLH	Internet Accessible Device, Remote System Information Discovery	Network Analysis, Information Manipulation, Application Diversity	Collect, Motivate
S7commTrace	Execution through API, Internet Accessible Device	API Monitoring, Network Monitoring	Collect

introduced ones, such as the extension of Engage to ICS. Using these building blocks, *ICSvertase* provides a novel approach to address several design and classification use cases. To demonstrate *ICSvertase*'s capabilities, we have used it to derive the design requirements of an existing honeypot and compared these to its original design, showing crucial gaps in the original design that could have been identified using *ICSvertase*. In addition, we have shown that *ICSvertase* can be used as a purpose-based classification scheme for ICS honeypots, replacing and improving the traditional interaction level-based approach. We showed that this new classification still allows to differentiate honeypots based on their complexity, but using a more informative feature set. By focusing on the purpose(s), *ICSvertase* also eases the selection of existing honeypots.

We plan to use *ICSvertase* in our ongoing efforts of creating new ICS honeypots, as well as extending the classification of existing ICS honeypots using our scheme to commercial honeypots, not evaluated here due to the lack of public information. We also envisage the possible extension of *ICSvertase* to IT honeypots.

REFERENCES

[1] D. Antonioli, A. Agrawal, and N. Ole Tippenhauer. 2016. Towards High-Interaction Virtual ICS Honeypots-in-a-Box. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy.*

[2] Anonymized Authors. 2023. *GitHub Page of the ICSvertase Project.* https://anonymous.4open.science/r/ICSvertase-C1E4/

[3] G. Bernieri, M. Conti, and F. Pascucci. 2019. MimePot: a Model-based Honeypot for Industrial Control Networks. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC).*

[4] S. Bistarelli, E. Bosimini, and F. Santini. 2021. A Medium-Interaction Emulation and Monitoring System for Operational Technology. In *Proceedings of the 16th International Conference on Availability, Reliability and Security.*

[5] D. Buza, F. Juhász, G. Miru, et al. 2014. CryPLH: Protecting Smart Energy Systems from Targeted Attacks with a PLC Honeypot. In *Smart Grid Security.*

[6] J. Cao, W. Li, J. Li, et al. 2018. DiPot: A Distributed Industrial Honeypot System. In *Smart Computing and Communication.*

[7] M. Dodson, A. Beresford, and M. Vingaard. 2020. Using Global Honeypot Networks to Detect Targeted ICS Attacks. In *2020 12th International Conference on Cyber Conflict (CyCon).*

[8] W. Fan, Z. Du, D. Fernández, et al. 2018. Enabling an Anatomic View to Investigate Honeypot Systems: A Survey. *IEEE Systems Journal* (2018).

[9] J. Franco, A. Aris, B. Canberk, et al. 2021. A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. *IEEE Communications Surveys Tutorials* (2021).

[10] J. Guarnizo, A. Tambe, S. Sankar Bhunia, et al. 2017. SIPHON: Towards Scalable High-Interaction Physical Honeypots. *CoRR* (2017).

[11] S. Hilt, F. Maggi, C. Perine, et al. 2020. Caught in the Act: Running a Realistic Factory Honeypot to Capture Real Threats. *Trend Micro Research* (2020).

[12] S. Kempinski. 2021. *OTCAD: Operational Technology Cyber Attack Database.* Secura. Retrieved 2023-01-22 from https://www.secura.com/whitepapers/otcad

[13] C. Krasznay and G. Gyebnár. 2021. Possibilities and Limitations of Cyber Threat Intelligence in Energy Systems. In *2021 13th International Conference on Cyber Conflict (CyCon).*

[14] R. Lee, M. Assante, and T. Conway. 2016. Analysis of the Cyber Attack on the Ukrainian Power Grid. *SANS* (2016).

[15] E. López-Morales, C. Rubio-Medrano, A. Doupé, et al. 2020. HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security.*

[16] S. Miller, N. Brubaker, D. Kapellmann Zafra, et al. 2019. *TRITON Actor TTP Profile, Custom Attack Tools, Detections, and ATT&CK Mapping.* Retrieved 2023-01-04 from https://www.mandiant.com/resources/blog/triton-actor-ttp-profile-custom-attack-tools-detections

[17] T. Miller, A. Staves, S. Maesschalck, et al. 2021. Looking back to look forward: Lessons learnt from cyber-attacks on Industrial Control Systems. *International Journal of Critical Infrastructure Protection* (2021).

[18] MITRE. 2022. *ATT&CK for ICS.* Retrieved 2023-01-22 from https://attack.mitre.org/matrices/ics/

[19] MITRE. 2022. *Engage.* Retrieved 2023-01-22 from https://engage.mitre.org/

[20] D. Pliatsios, P. Sarigiannidis, T. Liatifis, et al. 2019. A Novel and Interactive Industrial Control System Honeypot for Critical Smart Grid Infrastructure. In *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD).*

[21] V. Pothamsetty and M. Franz. 2005. *SCADA HoneyNet Project: Building Honeypots for Industrial Networks.* Retrieved 2023-01-23 from https://scadahoneynet.sourceforge.net/

[22] T. Sasaki, A. Fujita, C. H. Gañán, M. van Eeten, K. Yoshioka, and T. Matsumoto. 2022. Exposed Infrastructures: Discovery, Attacks and Remediation of Insecure ICS Remote Management Devices. In *2022 IEEE Symposium on Security and Privacy (SP).* 2379–2396. https://doi.org/10.1109/SP46214.2022.9833730

[23] R. Siles. 2007. HoneySpot : The Wireless Honeypot Monitoring the Attacker ' s Activities in Wireless Networks A design and architectural overview.

[24] L. Spitzner. 2001. *The Value of Honeypots, Part One: Definitions and Values of Honeypots.* Retrieved 2023-01-22 from http://www.symantec.com/connect/articles/value-honeypots-part-one-definitions-and-values-honeypots

[25] E. Vasilomanolakis, S. Srinivasa, C. Cordero, et al. 2016. Multi-stage attack detection and signature generation with ICS honeypots. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium.*

[26] F. Xiao, E. Chen, and Q. Xu. 2018. S7commTrace: A High Interactive Honeypot for Industrial Control System Based on S7 Protocol. In *Information and Communications Security.*

[27] J. You, S. Lv, Y. Sun, et al. 2021. HoneyVP: A Cost-Effective Hybrid Honeypot Architecture for Industrial Control Systems. In *ICC 2021 - IEEE International Conference on Communications.*

[28] J. Zaddach, L. Bruno, D. Balzarotti, et al. 2014. Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares *(Network and Distributed System Security (NDSS) Symposium).*