ICTOOLS: Fingerprinting ICS Tools to Characterize Honeypot Interactions

Stash Kempinski, Victor Asanache, Lucas Lahaije, Kristopher Schlett, Savio Sciancalepore,
Emmanuele Zambon, Nicola Zannone
Eindhoven University of Technology

***Abstract*—Honeypots are widely used to collect Cyber Threat Intelligence (CTI), particularly in Industrial Control System (ICS) environments where understanding malicious activity is critical. Existing analysis strategies, however, may misclassify honeypot interactions, leading to misdirected analytical efforts and misleading conclusions. In this paper, we present ICTOOLS, a framework for identifying ICS-capable tool fingerprints within honeypot network traffic logs, which is released as an open-source project. ICTOOLS supports multiple log formats and currently implements matchers for the Modbus/TCP and EtherNet/IP protocols through Zeek and PCAP logs, respectively. Using heterogeneous honeypot datasets, we show that ICTOOLS reliably distinguishes tools with overlapping functionality and provides valuable context for interactions that traditional function- or actor-based analyses cannot fully explain. By complementing existing approaches, it enables more targeted CTI extraction and reduces noise from irrelevant interactions.**

1. Introduction

Honeypots, i.e., deliberately vulnerable and exposed systems, are a long-standing method for collecting CTI. Their use dates back to the early 1990s [1] and continues today in both research and commercial domains [2]. In the context of ICSs, where cyber attacks are increasingly prevalent [3], ICS-specific honeypots are deployed to capture interactions that may reveal threats to critical infrastructure. However, not all recorded interactions are equally relevant for CTI. Many originate from unsophisticated scanning or generic Internet traffic, which is typically untargeted and thus yields limited actionable CTI. Consequently, characterizing interactions is a crucial step in enabling analysts to focus their efforts on those most likely to provide meaningful intelligence about adversarial goals, techniques, or capabilities [4].

From a network perspective, honeypot data typically comprise information about the actor (e.g., IP addresses) and the exchanged Protocol Data Units (PDUs). Analysts often characterize interactions by querying external CTI sources [5]–[7], or interpret the protocol function calls observed in the PDUs [4], [8]. However, these strategies are prone to misclassification, as external CTI sources do not always reflect an actor's specific behavior toward a given honeypot deployment, and protocol function calls do not always capture the purpose for which they were used. For instance, functions normally used to read process data may be repurposed for reconnaissance [9], while functions typically considered benign may be exploited to carry out attacks [10].

In this work, we propose an alternative but complementary approach: *tool-assisted behavioral interpretation*, which associates interactions with the tools used to perform them. Tools are defined as any instrument contributing to the exploitation of system vulnerabilities [11]. Here, we focus on network-capable software or scripts targeting ICS protocols. As tools are designed for specific purposes, identifying them enables CTI analysts to assess the capability and purpose behind an interaction by combining tool-related information with contextual cues. For example, the use of a tool may be interpreted differently depending on a honeypot's deployment environment: a tool used for Internet-wide scanning may be expected against an Internet-facing honeypot, whereas its use within an internal network could indicate suspicious activity. Similarly, distinguishing between tools with similar functionality supports behavior analysis: a tool designed to scan a single target is more indicative of targeted reconnaissance than one designed to scan the entire Internet. By combining tool-related information with the deployment context, analysts can focus on the interactions most likely to yield valuable insights.

To operationalize this approach, we introduce ICTOOLS (pronounced "I See Tools"), a framework for identifying the use of ICS-capable tools in honeypot interactions. ICTOOLS matches tool-specific fingerprints against honeypot logs and reports whether known tools were used during each interaction. ICTOOLS is designed to be log-format agnostic and to support a broad range of ICS protocols. Currently, ICTOOLS supports Modbus/TCP and EtherNet/IP (ENIP) ("Industrial Protocol") via Zeek and PCAP logs, respectively. In this work, we focus on Modbus/TCP using Zeek logs (discussing ENIP only when the context differs significantly) and analyze two Modbus and one ENIP honeypot datasets (Section 5).

We collect and fingerprint a representative set of Modbus- and ENIP-capable tools, classify them by capabilities and purposes, and integrate them into ICTOOLS. This classification enables us to assess the behavior of observed interactions and interpret interactions with unidentified tools by comparing them to known ones. Applying ICTOOLS to three datasets, we show that tool attribution complements existing CTI analysis approaches, reduces ambiguity in interaction assessment, and provides additional insights.

The paper is organized as follows. We discuss related work in Section 2. We present ICTOOLS in Section 3 and our

procedure for collecting and fingerprinting tools in Section 4. We present the data analysis in Section 5 and provide our considerations in Section 6. Section 7 concludes the paper.

2. Related Work

Studies that analyze data from ICS honeypot deployments primarily focus on comparing deployment strategies or gathering CTI at both the actor and protocol level. Deployment strategy studies examine how different honeypot capabilities (the "interaction level") and environments influence interactions. Srinivasa *et al.* [12] show that higher-interaction honeypots attract more traffic, while others emphasize the role of geographic IP location [13]–[15] and deployment environment (e.g., cloud or corporate network) [16], demonstrating that both factors significantly affect interaction volume and thus must align with the research objectives, as mismatched capabilities can yield incomplete or misleading insights.

Studies aimed at gathering CTI primarily focus on identifying the actors behind interactions or characterizing these interactions at the protocol level. Actor-focused threat intelligence studies typically rely on IP address analysis [4]–[7], [12], [13], commonly deriving attributes such as geolocation, Autonomous System (AS) (e.g., linking traffic to an AS frequently associated with bulletproof hosting [4]) and classifications from external CTI sources such as GreyNoise [17], which labels IPs as *Benign*, *Unknown*, or *Malicious* [5]–[7]. Most studies implicitly assume that these classifications generalize to honeypot interactions; however, as shown in Section 5.6.1, this assumption does not always hold. For example, an IP classified as malicious may perform only device indexing, which is not relevant for CTI.

CTI studies that analyze interactions at the protocol level are not prone to this type of misclassification. They examine individual protocol functions on a per-packet basis (e.g., authorization attempts or data reads) [4], [8], [18], or classify interactions based on the functions used (e.g., a brute-force attack) [12], [13]. This improves the understanding of interaction behavior, although some inference remains necessary. For example, López-Morales *et al.* [8] analyze the function calls received by their S7comm honeypot: a "Programmable Logic Controller (PLC) Stop" command is clearly malicious, whereas the intent behind "Read SZL", which requests device status information, is ambiguous. It may originate from an Internet scanner indexing the device or a tool verifying compatibility before a model-specific exploit. Moreover, as we show in Section 4.2.1, protocol functions can be used beyond their intended purpose (e.g., using a data read function to identify system information), so even protocol-level function analysis risks drawing incorrect conclusions about a honeypot interaction.

Some studies combine these strategies for finer-grained analysis. For example, Lupia *et al.* [5] assess whether protocol-level interactions aimed at device identification are malicious using external CTI classifications and analyze protocol functions across interactions. However, this approach is susceptible to both mislabeling issues described above.

Research gaps and challenges. Existing CTI studies attempt to infer the intent behind ICS honeypot interactions, ranging from labeling benign or malicious [12] to inferring interaction purpose (e.g., device identification or data exfiltration). However, both approaches are prone to systematic misclassification. IP-reputation-based labeling can classify even benign scanning as malicious, inflating the perceived volume of malicious traffic, while function-based labeling can misjudge purpose, since functions may be issued merely to verify protocol support rather than to attack [9], and seemingly benign functions (e.g., device identification) may instead precede targeted exploitation [10]. These biases can divert effort from real threats or cause valuable CTI to be missed.

These challenges motivate a different perspective on interaction analysis. Rather than focusing on protocol-level function codes, we classify interactions by the tools that generate them. This approach builds on prior findings that most ICS honeypot traffic is produced by scripted tools [5], [13]. Since tool purpose is typically easier to interpret than the low-level operations implementing it, tool-based classification can provide a clearer characterization of honeypot activity.

3. ICTOOLS

To address these gaps, we present ICTOOLS, a framework to identify ICS tool fingerprints in honeypot network traffic.

3.1. Requirements

The design of our framework is guided by four requirements (Table 1). Honeypot logs are generated in various formats: capturing raw packet data is the most straightforward approach but is not always desirable or feasible, e.g., when space-efficient logging is preferred or past deployments used alternative formats that preclude reconstructing raw packets. Our solution should therefore be format-agnostic (**R. 1**).

To determine which tools interacted with a honeypot, log entries corresponding to the same tool execution must be grouped together. Since honeypot logs record communications between devices chronologically, messages between a single device pair may appear in non-consecutive entries, and the same devices may also communicate multiple times in unrelated contexts. Identifying tool usage therefore requires grouping log entries that represent communication between two devices within the same context, i.e., a tool execution (**R. 2**); we refer to such a group of logs as an *interaction*.

Interactions ideally consist of a single tool execution. However, multiple tool executions may be grouped into a single interaction, e.g., when an actor runs several tools in seemingly the same context. In addition, because protocols support only a fixed set of functions and many tools exist, different tools may share portions of their fingerprints. For instance, device identification often relies on only a single (or few) function(s), making such tools hard to distinguish. Moreover, one tool's fingerprint can be a subset of another, leading to erroneous identification of multiple tools within the same interaction (see Section 5.3). To avoid misinterpretations, our solution must provide explainable outcomes (**R. 3**), explicitly indi-

Table 1. ICTOOLS REQUIREMENTS.

ID	Requirement
R. 1	Be able to support any honeypot log format.
R. 2	Group log entries originating from single tool execution.
R. 3	Provide explainable matching outcomes.
R. 4	Be extendable to any ICS tool and protocol.

cating which parts of an interaction led to the identification of each tool so that overlaps can be recognized and understood.

New tools are continuously developed and published, and the large variety of standard and proprietary ICS protocols makes it infeasible to support all protocols immediately. Our solution must therefore be extendable, allowing users to add support for new ICS tools and protocols (**R. 4**).

3.2. Design

ICTOOLS processes logs in four steps: *(i)* log parsing, *(ii)* interaction building, *(iii)* fingerprint matching, and *(iv)* coverage calculation. Figure 1 shows these steps and their relation.

3.2.1. Log Parsing. ICTOOLS processes input logs through dedicated parsers, each tailored to a specific protocol-log format combination. Each parser identifies the log structure, extracts protocol-relevant fields (e.g., PDU values), and converts them into a *unified packet* data structure. This structure contains core fields (timestamp, source and destination IP addresses, source and destination ports, and a request/response indicator) and can be extended with protocol-specific fields required for fingerprinting. Although log formats may encode the same protocol information differently, each protocol's representation (field mapping and semantics) is implemented only once, enabling ICTOOLS to support diverse log formats (R. 1) and adapt to any ICS protocol (R. 4) provided that the required fingerprinting data is available.

3.2.2. Interaction Building. After extracting all unified packets from the logs, ICTOOLS groups those likely belonging to the same tool execution into *interactions* (R. 2). This grouping requires identifying the actor and target, and determining when a tool execution has completed. ICTOOLS identifies the actor and target through the source and destination IP address pair of a packet, under the assumption that this pair remains constant during a single tool execution and that the target IP uniquely identifies a single honeypot. Unlike Lupia et al. [5], this pairing excludes port numbers because many tools dynamically change ports during execution (e.g., opening a new TCP session for each request). This allows ICTOOLS to consistently group executions across sessions and protocols. ICTOOLS considers an interaction complete when the interval between consecutive packets exceeds a predefined inactivity threshold δ. This accommodates connectionless protocols and tools that open multiple connections during a single execution, for which there are no reliable contextual clues to determine whether packets belong to the same execution. Selecting an appropriate δ is non-trivial: too small a value may split a single execution, while too large a value may group multiple executions together, a case ICTOOLS addresses through its

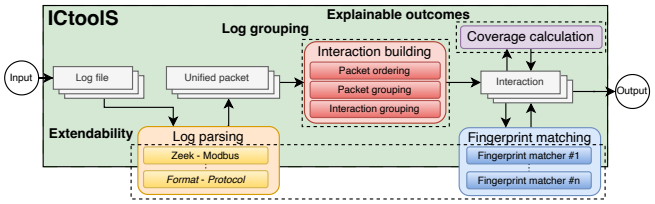


Figure 1. Flow chart of ICTOOLS, including the related requirements.

coverage calculation (Section 3.2.4). The interaction building process consists of three sequential steps: *(i)* sort all unified packets chronologically, *(ii)* group them by actor-target IP pair, and *(iii)* split each group into interactions based on δ.

3.2.3. Fingerprint Matching. ICTOOLS employs a packet-based approach for matching tool fingerprints, leveraging packet-level characteristics for accurate identification. Distinguishing between tools that produce similar network traffic may require characteristics such as specific PDU values or per-request TCP connection behavior (see Section 4). Each tool fingerprint includes a *fingerprint matcher*. The matchers analyze each interaction and tag the packets matching the corresponding tool fingerprint. Currently, matchers rely on the following features for fingerprint comparison (further discussed in Section 4.3.2): sequences of PDU values and their patterns (e.g., constant, sequential, or random), the number of requests (single, fixed, or repetitive), and branching behavior that may alter request sequences. Each matcher operates independently, enabling new fingerprints to be added without affecting existing ones (R. 4). Note that reliable results depend on complete packet logging. Missing data, due to mid-stream capture, early termination, or packet loss, may cause false matches or missed detection. Fortunately, this risk is minimal in most honeypot deployments, where traffic is limited and logging can be tightly controlled.

3.2.4. Coverage Calculation. To support informed decision-making and prevent misinterpretation (R. 3), ICTOOLS provides detailed visibility into interaction coverage. Coverage reflects how an interaction matches known tool fingerprints and helps analysts identify cases requiring further investigation. For example, incomplete coverage indicates that at least one packet in the interaction does not match any known tool, suggesting the presence of unknown or additional behavior. Coverage is computed after all matchers have processed an interaction, ensuring complete information is available. Each interaction is annotated with *(i)* whether more than one tool is identified (*multiple tool match*); *(ii)* whether at least one packet is matched by more than one fingerprint, which may point to tools with (partly) overlapping fingerprints (*overlap*); and *(iii)* whether at least one packet does not match any fingerprint (*incomplete coverage*). These three indicators produce seven possible coverage states (see Table 2).

3.3. Implementation

ICTOOLS consists of modular components corresponding to the steps described above and is *released as an open-*

Table 2. COVERAGE CALCULATION OUTCOMES.

Outcome	Description	Tool Matches	Overlap in Matched Packets	All Packets Covered
SC	Single tool match; Complete coverage.	One	N/A	Yes
SI	Single tool match; Incomplete coverage.	One	N/A	No
MNC	Multiple tool match; No overlap; Complete coverage.	Multiple	No	Yes
MNI	Multiple tool match; No overlap; Incomplete coverage.	Multiple	No	No
MOC	Multiple tool match; Overlap; Complete coverage.	Multiple	Yes	Yes
MOI	Multiple tool match; Overlap; Incomplete coverage.	Multiple	Yes	No
$\emptyset$	no match	None	N/A	N/A

source Python project [19]. Interactions are represented as objects containing actor and target IPs, unified packets, timestamps, matching results, and coverage results. ICTOOLS outputs a single JSON file containing all extracted interactions, each annotated with matching and coverage results. Currently, ICTOOLS implements a Zeek log parser for Modbus logs and a PCAP parser for ENIP logs. Since Zeek's default Modbus analyzer omits essential PDU fields, two plug-ins must be enabled when processing raw captures: CISA ICSNPP-Modbus [20] and Zeek UMAS [21]. The latter, developed by the authors, adds support for Schneider Electric's proprietary Modbus extension Unified Messaging Application Services (UMAS), widely used in industry [22], to Zeek. This extension has been previously reverse engineered by the security research community [23] and is used by several tools identified in our study. For ENIP, ICTOOLS parses PCAPs with pyshark [24].

4. Tool Collection & Fingerprinting

To demonstrate ICTOOLS's capabilities, we collect and fingerprint an extensive set of Modbus and ENIP tools in three phases: *Collection*, *Selection*, and *Fingerprinting*.

4.1. Collection

In the *Collection* phase, we: *(i)* search for Modbus and ENIP tools on Google and GitHub using a set of keywords, *(ii)* filter the results to keep manual analysis manageable, and *(iii)* manually assess relevance. Details are provided in Appendix A. Through this process, we identify 138 Modbus and 26 ENIP tools.[1] The complete list can be found in [19].

4.2. Selection

Due to the large number of tools identified, we apply a process to prioritize fingerprint implementation, streamlining development while maximizing coverage in the datasets used for this research (see Section 5.1). Modbus tool selection is guided by their capabilities (Section 4.2.1), enabling us to prioritize popular tools (see Section 4.2.2), and those with advanced capabilities. We consider only tools that can be deployed within 30 minutes and restrict our selection to tools that use functions and related characteristics present in the datasets considered in Section 5.

1. When a tool provides multiple functionalities, e.g., a collection of independent scripts, we treat each as a separate tool.

4.2.1. Modbus Capability Categorization. We determine the Modbus tools' capabilities based on the function codes they use, how they use them, and for what purpose. This categorization scheme follows the Modbus specification, from which we derive four categories: *Diagnostics & Identification*, *Data Reading*, *Data Writing*, and *Advanced Communication*. To account for Schneider Electric's UMAS extension, we extend *Advanced Communication* with UMAS authentication and authorization functions and *Diagnostics & Identification* with UMAS-specific diagnostic operations, and introduce a new category, *Process Modification*, covering functions that modify the control logic of UMAS-enabled devices. Table 3 summarizes the capability scheme; the full mapping of function codes to capabilities can be found in Appendix C.

We also classify how functions are used, as several tools use Modbus functions in ways that diverge from their intended purpose; e.g., ModBusSploit [25] uses *Read Input Registers* for its intended purpose, reporting the requested registers' state, whereas the Metasploit script at [9] uses the same function solely to detect the presence of a Modbus-capable device, without processing the response's content. We annotate tool capabilities as *proper* or *improper*, depending on whether the use of a function serves its intended purpose. We observe two forms of *improper* function usage, both essential to understanding a tool's purpose. The first occurs when a tool uses a function to gather device information rather than the returned data, or to probe which function codes a device supports. Tools showing this behavior are classified as *scanners* and the capability associated with the function used is marked as *improper*. Exceptions are the *Read Device Identification* and *Data Reading* function(s): if used to extract response data they are considered *proper*, although the tool is still classified as *scanner*. The second form of improper usage is inherent to fuzzers, which send randomly crafted, syntactically valid packets to discover protocol implementation flaws. Since these tools are not concerned with the intended use of any function code, all function codes are used improperly by design. We classify these tools as *fuzzers*.

4.2.2. Selection Process. We prioritize tools based on their significance to actors with malicious intent. Tools offering advanced capabilities are more significant because they perform complex tasks, provide higher automation, and likely have greater impact. Conversely, most *scanners* provide information obtainable from Internet scanner databases or generic Modbus clients, making them less critical.

Based on these criteria, we select all 36 Modbus relevant tools that are neither *scanners* nor *fuzzers*. From the 77 identified Modbus *scanners*, we choose the 10 most popular

Table 3. CAPABILITY CATEGORIZATION SCHEME.

Capability	Function Types
Diagnostics & Identification	Identification of device properties, such as vendor and model, and diagnostics of the device.
Data Reading	Reading the specific data types and the values associated with specific Modbus addresses.
Data Writing	Writing the specific data types to specific Modbus addresses.
Advanced Communication	Connection initiation*, device control*, and encapsulation of other protocols.
Process Modification*	Modification of processes running on the device.

*Supported through Schneider Electric's proprietary Modbus functions.

GitHub repositories and all 7 non-GitHub tools, yielding 17 *scanner*-tools. For tools on GitHub, we determine repository popularity by using their combined number of forks, stars, and watchers as ranking method, following Al-Rubaye *et al.* [26]. We exclude all 25 deterministic Modbus *fuzzers* due to low popularity, uncertain automation benefits, and limited relevance outside DoS scenarios. Out of the 53 selected Modbus tools, 7 could not be deployed within 30 minutes. Moreover, 14 Modbus tools use function codes not present in any of the datasets. As a result, we implement the fingerprints of 32 Modbus tools. Following a similar procedure, we implement the fingerprints of 8 ENIP tools. The list of tools and their capabilities (for Modbus tools), including those that were fingerprinted, is provided at [19].

4.3. Fingerprinting

We individually execute each selected tool multiple times targeted at our testbed. We capture the network traffic generated during each execution from which we extract the tool's fingerprint characteristics. If available, we also confirm the found characteristics through source code inspection.

4.3.1. Fingerprinting Testbed. As Modbus server we use a customized version of Conpot [27], a low-interaction industrial honeypot, modified to respond to Schneider Electric's UMAS functions (function code 90 and its subfunctions). We run it in TCP mode, for proper handling of Modbus/TCP PDU values, including requests directed to unit ID 0, which are invalid in serial mode (the default Conpot mode). As ENIP server, we use a Rockwell CompactLogix PLC.

4.3.2. Fingerprinting Characteristics. We consider the following characteristics, summarized in Table 4.
(i) PDU Values. We identify the *deterministic* PDU values: those that remain constant or predictable (e.g., sequentially increasing) among executions. Generally, these are the PDU values related to the purpose of a request, such as the function code and related variables. We also identify *non-deterministic* PDU values, i.e., those that are randomized at every request. During fingerprint matching, these values are treated as wildcards: a matcher will accept any value for these PDU values. In tools with *proper* function usage, this behavior is only found in PDU values that are irrelevant to the purpose of a request. For example, in Modbus, the transaction ID is merely echoed in the response to allow the requester to correlate requests and responses.

Table 4. CHARACTERISTICS CONSIDERED FOR FINGERPRINTING.

Type	Considered Characteristics
Modbus PDU	Function Code, Unit ID, Transaction ID, MEI Type*, Device ID Code*, Reference Number*, Bit/Word Count*
ENIP PDU	Command, Sess. Handle, Status, Sender Ctx, Options, Item Count*, Timeout*, Interface Handle*, CIP Class*, CIP Instance*, CIP Attribute*, CIP Service*, CIP Data*
TCP PDU	Source Port, Stream
Tool	Branching, Configurable, Single/Multiple Request(s)

*If applicable to the protocol function.

(ii) Configurable Parameters. Although parameter customization is limited in most tools, fingerprints often rely on the values of those that are configurable. For example, Modbus' native device identification function has three adjustable PDU values: *unit ID*, *read device ID code*, *transaction ID*. Among tools that use this function, the *transaction ID* is often the only distinguishing feature. For instance, ZGrab2 [28] uses a distinctive transaction ID (23111) in its default configuration, although this value is configurable. To avoid fingerprints that depend on user-specific settings, we construct fingerprints using only a tool's default execution parameters. This choice is discussed further in Section 6.2.4.
(iii) Branching Behavior. We include all execution branches identified through source code inspection, further ensuring coverage of branches for unexpected responses; this way, fingerprints remain valid across different honeypot deployments and for incompatible devices.
(iv) Request Count. We distinguish between single- and multi-request tools to reduce false positives, as one tool's fingerprint may be a subset of another. ICTOOLS's *fingerprint matchers* explicitly verify the expected number of requests. This is particularly relevant for *Diagnostics & Identification* tools, which typically send only one request.

5. Experimental Validation

We perform an in-depth analysis of two Modbus honeypot datasets to evaluate ICTOOLS and showcase its support for ICS honeypot research. We also assess the interaction coverage for an ENIP dataset, showing that ICTOOLS can handle other ICS protocols and highlight its extendability.

5.1. Datasets

We analyze two Modbus honeypot datasets, DS_1 and DS_2, and one ENIP honeypot dataset, DS_3. These three datasets stem from Internet-exposed deployments, i.e., they contain no artificially created interactions. DS_2 consists of four distinct deployment (environments): they either use HoneyICS [6] or Conpot [27] as honeypot, and are exposed either through a cloud- or corporate-owned IP address. To account for the differences in deployment conditions, we split DS_2 by network type and honeypot type: DS_2-ClH (Cloud/HoneyICS), DS_2-ClC (Cloud/Conpot), DS_2-CoH (Corporate/HoneyICS) and DS_2-CoC (Corporate/Conpot). The complete description of these datasets can be found in Appendix B.

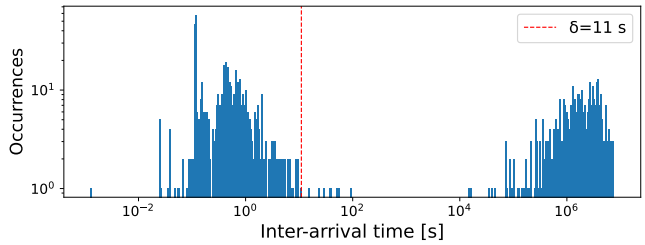


Figure 2. Distribution of inter-arrival times (seconds) between consecutive requests in DS_1 from the same source and destination IP pair.

5.2. ICTOOLS Configuration

A key parameter of ICTOOLS is the threshold δ (Section 3.2.2). We determine δ by analyzing the distribution of inter-arrival times between consecutive requests from the same source and destination IP pair in DS_1 (Figure 2). Four distinct groups emerge. *(i)* the far right shows long intervals typical of Internet scanners. *(ii)* just to the right of the dashed red line, short sequences (15–90 s) likely from actors manually initiating multiple tool executions. *(iii)* on the far left of the figure, rapid requests (<1 s) from tools sending consecutive requests during a single execution. *(iv)* intermediate intervals (1–11 s), reflecting intentional tool delays [29], or network variability. The distribution density up until the 11-second mark is consistent, indicating that this delay between two consecutive requests is not uncommon. Based on these observations, we set δ to 11 seconds, capturing single tool executions while keeping separate unrelated interactions. This threshold is validated for DS_2 and DS_3 in Section 5.3.2.

5.3. ICTOOLS Validation

5.3.1. Fingerprint Accuracy. We verify that each fingerprint matcher identifies its corresponding tool and no others. To this end, we generate a dataset by running the collected tools multiple times against our honeypot testbed (Section 4.3.1) and run ICTOOLS on that dataset. A synthetic dataset is required because the three real-world datasets lack ground truth on the tools used in honeypot interactions. The results show that matchers recognize the expected tool behaviors and that fingerprints are generally distinguishable. Exceptions are five cases whose fingerprints are (partly) subsets of others: *Modscan* and *Fuzzowski - Modbus monitor device* for *Nmap*, *Modscan* for *Kevincali - Modbus flooding demo*, *Scannerl* for *Meeas - PLCScan*, and *Metasploit-Modbus Detect* for *Metasploit-Find Unit Id*. Since matchers operate independently, ICTOOLS may report multiple tools for a single interaction, even when only the tool associated with the superset fingerprint is present. For the first four cases, ICTOOLS's coverage calculation resolves the ambiguity by identifying the subset fingerprint as a false positive (e.g., *Modscan* vs. *Nmap*). The remaining case, *Metasploit-Find Unit Id* [30] and *Metasploit-Modbus Detect* [9], requires special attention. Both tools share identical PDU values in their first request and remain indistinguishable until *Find Unit Id* continues unit ID enumeration. Although this ambiguity could be resolved by considering only executions that contain multiple requests, such an approach would risk misclassifying deliberately terminated executions. Distinguishing such cases would require attributing multiple tool interactions to the same actor, which is beyond ICTOOLS's scope. We therefore retain the more flexible matcher and leave this overlap for analyst interpretation.

Table 5. COVERAGE OUTCOME PER MODBUS DATASET.

DS	SC	SI	MNC	MNI	MOC	MOI	∅	Total
DS_1	1122 (87%)	9 (1%)	1 (0%)	2 (0%)	8 (1%)	83 (6%)	60 (5%)	1285
DS_2-ClH	4437 (83%)	72 (1%)	0 (0%)	0 (0%)	3 (0%)	13 (0%)	827 (15%)	5352
DS_2-ClC	1473 (79%)	26 (1%)	0 (0%)	2 (0%)	2 (0%)	4 (0%)	367 (20%)	1874
DS_2-CoH	632 (77%)	10 (1%)	0 (0%)	1 (0%)	0 (0%)	45 (6%)	129 (16%)	817
DS_2-CoC	630 (78%)	5 (1%)	0 (0%)	0 (0%)	2 (0%)	20 (2%)	150 (19%)	807

S = single tool match, M = multiple tool match. C = complete, I = incomplete. N = no overlap, O = overlap. ∅ = no match.

5.3.2. δ Validation. We assess whether δ derived from DS_1 remains valid for DS_2 and DS_3. An incorrect δ could fragment single tool executions into multiple interactions, or conversely, merge distinct executions into one. Table 5 and Table 8 show the coverage for DS_2 and DS_3 respectively. Similarly to DS_1, only a small fraction of interactions fall within these categories, and their causes are explainable (discussed further in Section 5.4). We therefore conclude that the value of δ = 11 s is also applicable to DS_2 and DS_3.

5.4. Modbus Interaction Analysis

5.4.1. Interaction Coverage. We assess interaction coverage across all five Modbus datasets and further investigate interactions that are not classified as either "single tool match" or "no match". Interactions with a single tool match are further examined in Section 5.4.2.

Table 5 presents the *coverage calculation* for each dataset. Overall, *82% of all interactions are identified as "single tool match, complete coverage" (SC)*. The percentage of matched interactions in DS_1 is notably higher than in DS_2. The difference may stem from the deployment environments: DS_1 is hosted on academic IP addresses, whereas DS_2 is not. Actors may behave more cautiously when targeting academic-owned IPs, for instance by suspecting honeypot deployments, and may therefore limit their interactions. Consistent with this interpretation, Table 6 shows that DS_1 contains relatively more activity from Internet-wide scanning tools such as ZGrab and ZGrab2. A similar interpretation applies to the difference between cloud-provider and corporate-owned IP addresses in DS_2's HoneyICS deployments, where cloud infrastructure appears less likely to attract actors using proprietary tools. This interpretation, however, may not extend to Conpot deployments. Conpot instances may be flagged as honeypots in Internet scanner databases, which could discourage further interaction by potential threat actors.

We observe 122 interactions of type *"single tool match, incomplete coverage" (SI)*. The vast majority of these interactions (118) match with *Scannerl*, a single-request scanning tool using distinct PDU values. Given the iterative nature of the remaining requests in these interactions, it is unlikely

that (at least) two tools (one being *Scannerl*) were used in quick succession. We therefore treat these identifications as false positives in subsequent analyses. The remaining four interactions match with *ZGrab2*. Three result from erroneous Zeek parsing, in which a non-Modbus packet was incorrectly labeled as Modbus. ICTOOLS correctly identifies *ZGrab2*, but the misparsed packet leads to incomplete coverage. We confirm the use of *ZGrab2* because the non-Modbus packets match its service-identification fingerprint, albeit on the Modbus/TCP port. The last interaction contains a generic TCP packet misinterpreted by the honeypot rather than by Zeek. The related non-Modbus packets, however, lack *ZGrab2* characteristic fingerprints; thus, we deem this a false positive.

One interaction is classified as *"multiple tool match, no overlap, complete coverage" (MNC)*, matching both *Redpoint-Modicon* and *Modscan*. Closer inspection shows that the first request aligns with the former tool's fingerprint, while subsequent requests match the latter's. As the time gap between these requests exceeds 500 ms, this interaction is consistent with the sequential execution of these two tools.

Five interactions are classified as *"multiple tool match, no overlap, incomplete coverage" (MNI)*. Two interactions match both *Modscan* and *Scannerl* and include one unknown request–response pair. Given the long but inconsistent inter-request intervals (1.5 s and 2.5 s), these interactions are consistent with sequential execution of these two tools and an unknown single-request tool. The remaining three interactions are discarded, as they consist of packets Zeek incorrectly identifies as Modbus and exhibit packet loss.

Fifteen interactions are classified as *"multiple tool match, overlap, complete coverage" (MOC)*. One interaction matches *Nmap* and its subset fingerprints, thus treated as solely matching *Nmap*. Twelve involve *Redpoint–Modicon* and *Nmap* and appear to have been executed sequentially. However, ICTOOLS flags them as overlapping because *Nmap*'s identification also includes the aforementioned subset fingerprints. Therefore, following the reasoning in Section 5.3, these interactions are reclassified as *MNC* and treated as such in subsequent analyses. The remaining two interactions match both *Metasploit-Find Unit Id* and *Metasploit-Modbus Detect*, as expected when only a single request is observed (Section 5.3). As no subsequent requests follow, we attribute both interactions to *Modbus Detect*.

Lastly, 165 interactions are classified as *"multiple tool match, overlap, incomplete coverage" (MOI)*. The majority of these interactions (162) match *Nmap* and its subset fingerprints. However, several indicators suggest that *Nmap* was not actually used: only a fraction of requests match Nmap's fingerprint, while the complete set of requests exhibits behavior indicative of a single tool. Therefore, these interactions are treated as false positives for further analyses. The remaining three match *Redpoint-Modicon* in addition to a request pattern similar to the one described above. As this request pattern is attributed to a single unidentified tool, we treat these interactions in further analyses as sequential use of *Redpoint-Modicon* and an unidentified tool.

Table 6. OCCURRENCE OF TOOLS IDENTIFIED IN MODBUS DATASETS. ADJUSTED FOR (LIKELY) FALSE POSITIVES.

Tool	DS_1	DS_2-ClH	DS_2-ClC	DS_2-CoH	DS_2-CoC	Total
ZGrab	556	1096	393	95	82	2222
ZGrab2	543	3226	1041	517	539	5866
Nmap - Modbus-discover.nse	8	3	2	0	0	13
Redpoint-Modicon	17	89	29	7	4	146
Metasploit-Banner Grabbing	0	0	0	0	2	2
Metasploit-Modbus Detect	0	0	0	0	2	2
Modscan	90	0	0	7	1	98
Fuzzowski - Modbus monitor	3	28	8	4	1	44
Scannerl - Modbus	9	5	4	3	1	22
Total	1226	4447	1477	633	632	–

5.4.2. Tool Identification. Table 6 reports the tools identified in each Modbus dataset based on the analysis in Section 5.4.1. Across the five datasets, only nine distinct tools were observed, and interactions involving these tools represent *83% of all interactions*. We observe that ZGrab (v1) remains in active use despite being deprecated on July 8th, 2021 [31], several years before either honeypot deployment. This is notable given the availability of ZGrab2, which appears more frequently in DS_2, the more recent dataset. The two Metasploit scripts are identified only in DS_2-CoC, one of the four corporate-exposed honeypots. In contrast, Modscan is not observed in any cloud-based honeypot deployments in DS_2. We further discuss these findings in Section 6.1.1.

5.4.3. Unmatched Interactions and Unidentified Tools. We analyze request patterns of interactions without a match to assess the presence of unidentified tools by applying the same process used to create tool fingerprints (Section 4.3). Across the five datasets, 1,533 interactions do not match any tool fingerprint, among which we identify *358 distinct request patterns*. Of these, *325 occur only once* (i.e., a unique combination of fingerprinting characteristics), more likely the result of tools with at least one non-deterministic PDU value or modified default configurations, while the remaining *33 patterns recur across multiple interactions* (1,208 in total), strongly suggesting they originate from tools with fully deterministic characteristics. We confirmed these patterns do not correspond to tools that have been collected but not implemented into ICTOOLS yet (Section 4.2.2).

5.4.4. Capabilities of Unidentified Tools. We analyze the observable *capabilities* of potentially unidentified tools to determine whether they resemble those identified in Section 4, first mapping the functions used in unmatched interactions according to the procedure described in Section 4.2.1, then assessing whether they are used properly or improperly by comparing them to known tool characteristics; the results inform our interpretation in Section 6.1.2. Table 7 reports the number of interactions in which each capability appears, separated by whether a tool was matched. The (likely) false positives identified in Section 5.4.1 have been reassigned from "Tool Matched" (TM) to "No Tool Matched" (NTM).

Diagnostics & Identification. Most interactions, matched and unmatched, consist only of *Diagnostics & Identification* requests, aligning with the scanner tools identified in Section 5.4.2. Among the distinct patterns, *334 (21 recurring)* fall in this category, exhibiting behavior comparable to

Table 7. CAPABILITIES IN INTERACTIONS WITH- AND WITHOUT TOOL MATCH, PER DATASET. ADJUSTED FOR (LIKELY) FALSE POSITIVES.

Capability	DS_1		DS_2-ClH		DS_2-ClC		DS_2-CoH		DS_2-CoC	
	TM	NTM	TM	NTM	TM	NTM	TM	NTM	TM	NTM
Diagn. & Identif.	1216	56	4443	740	1473	261	633	177	630	127
Data Reading	0	16	0	168	0	140	0	7	2	44
Data Writing	0	0	0	0	0	0	0	0	0	4
Advanced Comm.	0	0	0	0	0	0	0	0	0	0
Process Modification	0	0	0	0	0	0	0	0	0	0

TM = Tool Matched. NTM = No Tool Matched.

known device-identification tools. However, inferring tool capabilities from these interactions remains unreliable, as tools may perform device identification merely to assess exploit compatibility before proceeding.

Data Reading. There is a notable *surplus of unmatched Data Reading interactions* compared to matched ones. Only two matched interactions exhibit *Data Reading* behavior, both linked to *Metasploit-Modbus Detect*. Among unmatched Data Reading interactions, we identify *19 unique patterns*, of which *11 recur*. Nine patterns display clear *improper Data Reading*, similar to the two identified tools. This group includes eight single-request patterns and one multi-request pattern that iterates over unit IDs instead of addresses. Improper behavior dominates: only *13 out of 375* unmatched *Data Reading interactions* appear to perform legitimate process state extraction. The remaining ten patterns vary widely in scale, from short sequences of a few tens of requests to large-scale scans of hundreds of coils, registers, and discrete inputs; these resemble *generic state-mapping scans*, such as *MAS - Read all holding registers* [32], which collect information without interpreting the returned values. One pattern instead targets a specific and narrow holding register range with no clear rationale, consistent with *prior knowledge of device variables*, e.g., as used by *clickplc_modbus* [33].

Data Writing. Only four interactions perform *Data Writing*, all using the same single-request pattern: writing "0xff" to address "1". Without knowledge of the tool or device's internal logic, we cannot determine if this constitutes *proper* or *improper* use, as the operational effect remains unknown.

5.5. ENIP Interaction Analysis

Table 8 presents the coverage calculation for DS_3. Only 20% of ENIP interactions are classified as *"single tool match, complete coverage" (SC)* based on the collected tools, mostly due to the limited number of identified ENIP tools (see Section 4.1); these include three scanning tools: *SCADA-CIP-Discovery* (279), *Digital Bond-Redpoint-ENIP* (213), and *Ethersploit/IP - get device info* (5). Although this coverage appears lower than for the Modbus datasets, manual analysis reveals that most ENIP interactions originate from Internet-wide scanners that, unlike for Modbus, rely on proprietary but still clearly distinguishable ENIP tools. Adding only two fingerprints, for Shodan and Censys, raises coverage from 20% to 65%.

Table 8. COVERAGE OUTCOME FOR DS_3, EXCLUDING AND INCLUDING THE FINGERPRINTS FROM SHODAN AND CENSYS.

	SC	SI	MNC	MNI	MOC	MOI	∅	Total
excl.	497 (20%)	13 (1%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	1981 (80%)	2491
incl.	1609 (65%)	13 (1%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	869 (35%)	2491

S = single tool match, M = multiple tool match. C = complete, I = incomplete. N = no overlap, O = overlap. ∅ = no match.

Table 9. COMPARISON OF INTERACTIONS FLAGGED BY IP-BASED ANALYSIS STRATEGY AND ICTOOLS.

Method	DS_1	DS_2-ClH	DS_2-ClC	DS_2-CoH	DS_2-CoC	Total
ICTOOLS	163 (13%)	915 (17%)	401 (21%)	185 (23%)	177 (22%)	1841 (18%)
GreyNoise	268 (21%)	873 (16%)*	326 (17%)*	66 (8%)*	58 (7%)*	1591 (16%)
Non-scanner IPs	394 (31%)*	2933 (55%)*	1159 (62%)*	336 (41%)*	255 (32%)*	5077 (50%)

*Flag determined at analysis time, not at data collection time.

5.6. Comparison with Other Strategies

We compare ICTOOLS with the two most common honeypot traffic analysis approaches (Section 2): IP-based analysis and function-code-based analysis.

5.6.1. IP-based analysis. Two IP-based analysis strategies are commonly used in honeypot traffic analysis [5]–[7], [12]. The first, as in [12], uses a predefined list of known scanner IPs, classifying all other interactions as malicious. The second, used in [5]–[7], leverages third-party CTI platforms, such as GreyNoise [17], to label IP activity as "noise" (i.e., indicating untargeted, background scanning behavior). We compare these approaches with ICTOOLS, which flags all interactions not labeled as *"single tool, complete coverage" (SC)* to be further analyzed to assess their CTI-relevance. Note that for DS_1, the GreyNoise classifications were obtained at the time of deployment, while for DS_2 and the known scanner IPs [34], the respective classifications were obtained three months after the DS_2 deployment. This may slightly affect accuracy due to changes in IP ownership, particularly for hosting-provider addresses.

Table 9 presents the number of interactions flagged for further investigation per dataset by the three strategies. *GreyNoise flags the fewest interactions*, partly because some Internet scanners use proprietary tools for which we have no fingerprint. For example, distinct patterns from Shodan-owned IP addresses do not match any known fingerprints, and are thus flagged for further investigation. Remarkably, there is only limited overlap between the interactions flagged by GreyNoise and those flagged by ICTOOLS, as illustrated in Table 10. Of the 1,841 interactions flagged by ICTOOLS, *1,711 were classified as noise by GreyNoise*. Conversely, of the 1,591 interactions GreyNoise labeled as not noise, *1,461 were identified by* ICTOOLS *as involving a known tool*, most of which are scanners (cf. Table 6), and thus could also be considered "noise". This indicates that ICTOOLS *complements IP-based strategies*: combining the two approaches reduces the number of interactions requiring further investigation to only *130 (1.3% across all five datasets)*.

Additionally, we observe that *372 interactions flagged by* ICTOOLS *originate from known scanner IPs*. If the patterns confidently attributed to proprietary or unidentified

Table 10. DIFFERENCES BETWEEN THE THREE STRATEGIES.

Flagged by	Not flagged by: ICTOOLS	GreyNoise	Non-scanner IPs
ICTOOLS	–	1711	372
GreyNoise	1461	–	41
Non-scanner IPs	3608	3527	–

tools used by known Internet scanners were implemented in ICTOOLS, the number of interactions flagged by ICTOOLS could be reduced to *at most 1,469*, and potentially even lower if such tools are also used by IPs outside known scanner lists (as suggested by Ferretti *et al.* [4]). Incorporating such an approach into ICTOOLS would therefore reduce analyst workload, potentially outperforming GreyNoise alone.

5.6.2. Function (code)-based analysis. Several works categorize interactions by analyzing the functions (codes) used [4], [5], [35]. This strategy is conceptually straightforward, as functions generally have well-defined purposes, and applying it yields results similar to those in Table 7. However, this method has two major limitations. First, it cannot account for functions that depend on prior responses: Table 6 shows 146 interactions using *Redpoint–Modicon*, which first issues a Read Device Identification request to determine the manufacturer and proceeds only if it is Schneider Electric, as later steps rely on UMAS functions. A function-based approach would infer this behavior only if the honeypot emulated Schneider Electric devices, whereas ICTOOLS enables this inference even when subsequent calls are absent. Second, function-based methods often conflate a function's nominal purpose with that of the tool using it: *Metasploit-Modbus Detect*, observed twice in DS_2-CoC, uses the Read Input Registers function, classified as *Data Reading*, so a function-based strategy would infer a data reading purpose, whereas the tool's real purpose is *Diagnostics & Identification* (see Section 4.2.1); identifying the actual tool enables correctly identifying this purpose, providing a more accurate assessment.

6. Discussion

6.1. Analyses Outcomes

6.1.1. Identified Modbus Tools. Our analysis identified only nine tools across the Modbus datasets (Table 6), with ZGrab and ZGrab2 together accounting for 80% of all interactions. Both are designed for large-scale Internet surveys [28] and commonly used for broad reconnaissance by Internet scanners such as Censys; given their (likely) untargeted nature, these interactions are unlikely to yield valuable CTI. The remaining seven tools are also scanners, probing for Modbus-capable devices or collecting basic device information, consistent with most interactions classified as *Diagnostics & Identification* (Table 7), and provide no unique information beyond what Internet-scanner databases already offer. However, these tools are intended for targeted scanning; their identification should therefore raise suspicion and warrant investigation, illustrating that even a seemingly trivial purpose (e.g., device identification) can justify deeper scrutiny once the tool is known.

6.1.2. Unmatched Interactions. As shown in Table 7, unmatched interactions mostly exhibit *Diagnostics & Identification* or *Data Reading* behavior resembling scanner-like activity (Section 5.4.4), consistent with GreyNoise's classification of most actors as "noise" (Section 5.6.1). Thus, the lack of identified tools likely reflects modified or proprietary tools rather than fundamentally different behavior. However, a smaller subset displays ambiguous behavior: the executed operations are identifiable, but their purpose remains unclear without knowledge of the underlying tool, illustrating that deriving purpose solely from function types may lead to misinterpretation. For example, in DS_1, three interactions (one per exposed PLC) used UMAS-specific *Diagnostics & Identification* functions. These appear benign, but UMAS includes authentication, and a 2022 Kaspersky report revealed an authentication-bypass vulnerability leveraging a *Data Reading* function [10]; although unused here, these interactions may represent similar exploit attempts that terminated after an incompatible response, underscoring the importance of inferring purpose to detect potentially malicious interactions.

6.2. Robustness of ICTOOLS

6.2.1. Completeness of Logs. ICTOOLS can match an interaction to a tool only if all fingerprint characteristics are present, which requires complete logs containing both requests and responses. Fingerprints are ideally derived from requests, as these include most deterministic tool characteristics, but responses are essential for tools with branching or looping behavior. For instance, *Redpoint–Modicon* halts if the target is not a Schneider Electric device, and without that response it is impossible to tell whether subsequent requests were skipped due to branching logic or because a different single-request tool was used. Future work includes extending ICTOOLS with confidence-based matching to analyze incomplete datasets; for Modbus specifically, matchers could sometimes derive required characteristics from responses when requests are missing.

6.2.2. Correctness of Logs. Parsing errors can inflate the number of "no match" interactions, triggering unnecessary investigations. As shown in Section 5.4.1, some packets were incorrectly classified as Modbus by both Zeek and Wireshark, e.g., when the PDU length value indicated a zero byte payload while the payload contained a valid HTTP request. Without raw packet captures, such issues cannot be resolved, underscoring the importance of accurate preprocessing prior to analysis.

6.2.3. Importance of Coverage Outcome Analysis. Ideally, an interaction consists of a single, uniquely identifiable tool execution. In practice this does not always hold, as correctly grouping packets into the same context is inherently difficult [36], confirmed by Section 5.4.1. Tools may run in parallel or sequentially within one interaction, and known tools

may overlap in fingerprints with each other or with unknown tools, causing (incorrect) multi-tool identification. Although ICTOOLS flags such cases, interactions whose coverage outcome is not *"single tool match; complete coverage"* must still be further investigated before use in subsequent data analysis.

6.2.4. Fingerprinting Evasion. Actors may evade fingerprinting by making tools indistinguishable. For example, a tool performing device identification to confirm exploit compatibility, terminating early if incompatible, could disguise its request as originating from ZGrab2. This limitation applies to any device-identification tool, and without follow-up interactions, no concrete CTI can be extracted regardless. Conversely, actors may hide the use of a common tool such as ZGrab2, e.g., by modifying its default *transaction ID* (Section 4.3.2), causing ICTOOLS to miss the tool. However, this deliberate evasion is itself likely to draw attention, as *no match* interactions are more likely to be flagged for further analysis. Both scenarios have no relevant impact on CTI analysis.

6.3. Implications for Research and Practice

6.3.1. Tool-assisted Behavioral Interpretation. Our tool analysis (Section 4.2.1) demonstrates that ICS protocol functions are not always used in alignment with their intended design, e.g., non-identification functions have been employed for device identification, and legitimate functions repurposed for exploitation (Section 6.1.2). Additionally, not all tool functionalities manifest in every interaction. *Redpoint-Modicon* may terminate early, leaving partial traces, while *PlcInjector* [37], designed to inject shellcode into Modbus registers, first performs a memory availability check and aborts if insufficient memory is found, resulting in interactions containing only a *Data Reading* function (the same used by, e.g., *Metasploit-Modbus Detect*). This illustrates how similar function codes can require very different interpretations. These observations underscore the methodological challenge of classifying interactions solely from observed functions and characteristics, and suggest that studies labeling interactions with *Data Reading* functions as malicious, e.g., [5], [12], [18], [35], may produce inaccurate inferences, a challenge not limited to known tools, as Section 5.4.4 shows similar variation among unmatched *Data Reading* interactions, whose purpose remains just as hard to infer.

Limitations are even more prominent in studies relying solely on IP-based strategies. For example, Srinivasa *et al.* [12] label interactions as malicious based only on whether the actor is a known Internet scanner, identifying 4.8M (44%) of interactions in their study, across six protocols, as malicious; the graphs indicate that roughly 600k (13%) of these are Modbus. Assuming these interactions resemble those in our datasets, our comparison in Section 5.6.1 suggests that ≈71% of those Modbus interactions may have been mislabeled, which would reduce their overall reported malicious share from 44% to 40%. Future studies should therefore account for potential mislabeling, e.g., by using more robust labeling strategies such as ICTOOLS, and future research could develop complementary behavioral interpretation strategies for tools ICTOOLS does not yet know.

6.3.2. ICTOOLS as a Complementary Strategy. ICTOOLS can provide context for several interactions that existing strategies would otherwise flag for manual review (Section 5.6.2), directly attributing many of them to specific tools. However, a tool-based approach alone cannot classify interactions where no tool is identified, whereas alternative approaches may still provide complementary insights at different levels of certainty. Combining strategies is therefore particularly effective. Interactions from well known Internet scanners using proprietary tools can be reliably derived through an IP-based strategy. Yet, as Table 10 illustrates, this alone does not capture the full set of likely untargeted, Internet-wide scanning interactions. Instead, identifying ZGrab(2) enables analysts to infer untargeted interactions regardless of the source IP. Overall, combining ICTOOLS with IP-based scanner identification reduces the number of interactions requiring manual review by 71% compared to IP-based identification alone, without introducing additional ambiguity.

6.3.3. Using ICTOOLS Beyond Known Tools. We observe at least 33 recurring Modbus interaction patterns likely originating from unknown tools (Section 5.4.3). Implementing them as distinct fingerprints via a bottom-up approach, similar to Section 5.5, could substantially reduce *no match* interactions and make recurring behavior identifiable or interpretable through comparison with known tools, albeit with the limitations discussed in Section 6.2.

7. Conclusion

Accurately capturing the behavior of honeypot interactions is essential for meaningful CTI extraction, as misclassification can divert analytical effort toward irrelevant interactions and lead to misleading conclusions. In this paper, we propose a tool-based strategy that interprets ICS honeypot interactions by identifying the tools involved, enabling more accurate behavioral assessment. To this end, we introduce ICTOOLS [19], a framework for detecting ICS-capable tools in honeypot network traffic, which we release as an open source project. We assess its effectiveness using three real-world ICS honeypot datasets and compare it with existing detection strategies. Our findings show that ICTOOLS provides valuable context beyond existing methods, complementing current analysis workflows. Future work should extend this approach by systematically extracting recurring unknown patterns from honeypot logs, enabling broader coverage of honeypot traffic.

Acknowledgments

This work has been supported by the INTERSECT project, Grant No. NWA.1162.18.301, funded by the Netherlands Organization for Scientific Research (NWO). Any opinions, findings, conclusions, or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of NWO. We also thank Bureau Veritas Benelux for the access to EtherNet/IP honeypot data.

References

[1] B. Cheswick, "An Evening with Berferd: In Which a Hacker is Lured, Endured, and Studied," in *Proceedings of the USENIX Winter 1992 Conference*, 1992.

[2] Forescout - Vedere Labs, "2025H1 Threat Review: Vulnerabilities, Threat Actors, and Ransomware," Forescout - Vedere Labs, 2025.

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

[4] P. Ferretti, M. Pogliani, and S. Zanero, "Characterizing Background Noise in ICS Traffic Through a Set of Low Interaction Honeypots," in *Workshop on Cyber-Physical Systems Security & Privacy*, ser. CPS-SPC'19. ACM, 2019, p. 51–61.

[5] F. Lupia, M. Lucchese, M. Merro, and N. Zannone, "ICS Honeypot Interactions: A Latitudinal Study," in *International Conference on Big Data (BigData)*. IEEE, 2023, pp. 3025–3034.

[6] M. Lucchese, F. Lupia, M. Merro, F. Paci, N. Zannone, and A. Furfaro, "HoneyICS: A High-interaction Physics-aware Honeynet for Industrial Control Systems," in *International Conference on Availability, Reliability and Security*, ser. ARES '23. ACM, 2023.

[7] O. Cabana, A. M. Youssef, M. Debbabi, B. Lebel, M. Kassouf, and B. L. Agba, "Detecting, Fingerprinting and Tracking Reconnaissance Campaigns Targeting Industrial Control Systems," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Cham: Springer International Publishing, 2019, pp. 89–108.

[8] E. López-Morales, C. Rubio-Medrano, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, and G.-J. Ahn, "HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. ACM, 2020, p. 279–291.

[9] Minded Security, "Metasploit - Modbus Detect," 2011. [Online]. Available: https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/scada/modbusdetect.rb

[10] P. Nesterov, N. Komarov, and A. Muravitsky, "The Secrets of Schneider Electric's UMAS Protocol," 2022. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2022/09/29/the-secrets-of-schneider-electrics-umas-protocol/

[11] B. Arief, M. A. B. Adzmi, and T. Gross, "Understanding Cybercrime from Its Stakeholders' Perspectives: Part 1–Attackers," *IEEE Security & Privacy*, vol. 13, no. 1, pp. 71–76, 2015.

[12] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Interaction matters: a comprehensive analysis and a dataset of hybrid IoT/OT honeypots," in *Annual Computer Security Applications Conference*, ser. ACSAC '22. ACM, 2022, pp. 742–755.

[13] M. Dodson, A. R. Beresford, and M. Vingaard, "Using Global Honeypot Networks to Detect Targeted ICS Attacks," in *2020 12th International Conference on Cyber Conflict (CyCon)*, vol. 1300, 2020, pp. 275–291.

[14] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici, "SIPHON: Towards Scalable High-Interaction Physical Honeypots," in *Workshop on Cyber-Physical System Security*, ser. CPSS '17. ACM, 2017, pp. 57–68.

[15] F. Xiao, E. Chen, and Q. Xu, "S7commTrace: A High Interactive Honeypot for Industrial Control System Based on S7 Protocol," in *Information and Communications Security*. Cham: Springer International Publishing, 2018, pp. 412–423.

[16] F. Ondrikov, D. Donadel, F. Lupia, M. Merro, D. dos Santos, E. Zambon, and N. Zannone, "A Comparative Study of ICS Honeypot Deployments," in *International Workshop on Cyber-Physical Security for Critical Infrastructures Protection (CPS4CIP 2025)*, ser. LNCS. Springer, 2025.

[17] GreyNoise Intelligence, "GreyNoise Threat Intelligence Platform," 2026. [Online]. Available: https://docs.greynoise.io/

[18] S. Bistarelli, E. Bosimini, and F. Santini, "A Medium-Interaction Emulation and Monitoring System for Operational Technology," in *International Conference on Availability, Reliability and Security*, ser. ARES '21. ACM, 2021.

[19] Security Cluster TU Eindhoven, "ICtoolS: fingerprinting ICS tools to characterize honeypot interactions," 2026. [Online]. Available: https://github.com/Security-Cluster-TU-Eindhoven/ICtoolS

[20] CISA, "ICSNPP Modbus." [Online]. Available: https://github.com/cisagov/icsnpp-modbus

[21] Security Cluster TU Eindhoven, "znpa-umas: Zeek Network Protocol Analyzers – Modbus Unity UMAS," 2026. [Online]. Available: https://github.com/Security-Cluster-TU-Eindhoven/znpa-umas

[22] Kaspersky, "Kaspersky ICS CERT reveals "secrets" in Schneider UMAS protocol," https://www.kaspersky.com/about/press-releases/kaspersky-ics-cert-reveals-secrets-in-schneider-umas-protocol, 2022.

[23] lirasenlared, "The Unity (UMAS) protocol (Part I)." [Online]. Available: https://lirasenlared.blogspot.com/2017/08/the-unity-umas-protocol-part-i.html

[24] Kimi Newt, "pyshark," 2023. [Online]. Available: https://github.com/KimiNewt/pyshark

[25] C4l1b4n, "ModBusSploit - read input register," 2021. [Online]. Available: https://github.com/C4l1b4n/ModBusSploit/blob/main/modules/exploit/injection/readInputRegister.py

[26] A. Al-Rubaye and G. Sukthankar, "Scoring Popularity in GitHub," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2020, pp. 217–223.

[27] Conpot Project, "Conpot." [Online]. Available: http://conpot.org

[28] zmap, "ZGrab2," 2025. [Online]. Available: https://github.com/zmap/zgrab2

[29] L. Salazar, S. R. Castro, J. Lozano, K. Koneru, E. Zambon, B. Huang, R. Baldick, M. Krotofil, A. Rojas, and A. A. Cardenas, "A Tale of Two Industroyers: It was the Season of Darkness," in *Symposium on Security and Privacy (S&P)*. IEEE, 2024, pp. 312–330.

[30] EsMnemon, "Metasploit - Find Unit Id," 2012. [Online]. Available: https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/scada/modbus_findunitid.rb

[31] zmap, "ZGrab," 2021. [Online]. Available: https://github.com/zmap/zgrab

[32] R. Allain, "Modbus Attack Scripts," 2018. [Online]. Available: https://github.com/romainallain/mas

[33] Cutaway Security, "clickplc_modbus," 2024. [Online]. Available: https://github.com/cutaway-security/clickplc_modbus

[34] C. Wild, "Scanner IP list," 2025. [Online]. Available: https://github.com/Connie-Wild/scanner-ip-list

[35] S. Bistarelli, E. Bosimini, and F. Santini, "On Identifying Repeated Patterns of OT Attacks with LOGistICS," in *Italian Conference on Cybersecurity (ITASEC 2022)*, vol. 3260. CEUR-WS.org, 2022, pp. 17–29.

[36] P. Olivier and N. Benameur, "Flow level IP traffic characterization," in *Teletraffic Engineering in the Internet Era*, ser. Teletraffic Science and Engineering. Elsevier, 2001, vol. 4, pp. 25–36.

[37] B. Merino, "Modbus Stager: Using PLCs as a payload/shellcode distribution system," 2016. [Online]. Available: https://www.shelliscoming.com/2016/12/modbus-stager-using-plcs-as.html

[38] F. Lupia, M. Lucchese, M. Merro, and N. Zannone, "ICS Honeypot Dataset," 2025. [Online]. Available: https://github.com/fras3c/ICS-Honeypot-Dataset

[39] F. Ondrikov, D. Donadel, F. Lupia, M. Merro, D. dos Santos, E. Zambon, and N. Zannone, "Evaluating ICS honeypot attractiveness: The role of interaction level, network type, and geolocation," *International Journal of Critical Infrastructure Protection*, vol. 55, 2026.

Appendix A.
Tool Collection

A.1. Search Queries

When searching for Modbus-capable tools, it is important to note that many manufacturers implement this protocol, so tools may not be explicitly branded as "Modbus", but instead appear under vendor-specific names. We construct two keyword sets: one with vendor names and another with tool-related keywords for scanning/discovery, penetration testing, traffic generation, and attack toolkits, combined with "Modbus". The second keyword set was selected after initial investigation to minimize overlap while maintaining relevant results. We generate search queries from all pairwise combinations of the two sets presented in Table 11.

We run the search queries on both Google and GitHub. For Google, we use the *custom search API* and scrape the first ten pages per query, as preliminary testing shows relevant results rarely appear beyond this limit. We also include the term "download" via a Google dork to filter out news articles and vendor pages. For GitHub, we extend the default repository, title and description search to README files, which often contain richer tool information. To maximize coverage, we include informal keyword variants (e.g., "pentesting" instead of "penetration testing"), as they appear frequently on GitHub. In total, GitHub and Google return 31,279 results.

A.2. Filtering

We apply a set of filtering criteria to reduce the number of results before manual review. First, we remove duplicate entries. We then exclude clear mismatches, such as vendor websites, webshops, news sites, and social media platforms. To identify such websites, we count how often each base URL appears, rank the most frequent ones, and confirm that these sites do not contain tools. Finally, we filter out URLs containing keywords that indicate the result is unlikely to contain a tool, such as specific file types (e.g., "pdf"), support information (e.g., "support", "faq"), or keywords associated with the previously mentioned categories ("blog", "article", "post"). We identify these keywords during preliminary analysis, which showed that such URLs rarely include actual tools. The complete list of filtered URLs and keywords appears in Table 13. After filtering, 3,562 URLs remain for manual review.

A.3. Determining Result Relevance

We determine the relevance of each result by verifying that the URL contains a Modbus tool or links to one, and then evaluate its relevance through four criteria.

First, the tool must be freely available for download and use; otherwise, it cannot be included in our fingerprinting process. Second, the tool must support the TCP version of Modbus. Multiple versions of Modbus exist, such as Modbus over serial interfaces, and search queries also return tools solely intended for these versions. Since these versions do not appear in packet-switched networks without converters, they are unlikely present in any honeypot data; we therefore exclude them. Third, the tool must automate a task rather than merely provide a means for a user to send arbitrary Modbus messages. While such tools could potentially be used to exploit a system, their intent cannot be inferred. For example, the search queries returned many Modbus "clients", which provide a user-friendly interface for the Modbus protocol while leaving users free to specify function codes and parameters. Because the rationale behind specific function use cannot be determined, we exclude these tools. Finally, we exclude non-deterministic fuzzers, as they do not exhibit deterministic, observable characteristics that can be fingerprinted.

Table 11. KEYWORD SETS USED FOR GENERATING SEARCH QUERIES.

Category	Keywords
Vendors	Siemens, Schneider (Electric), Hitachi, ABB, Rockwell, Yokogawa, Allen-Bradley, Mitsubishi (Electric), Omron, IDEC, B&R Industrial Automation, Koyo Electronics Industries, Delta, EATON, Keyence, LS Electric, Panasonic, Pilz, WAGO (Kontakttechnik), Toshiba, Phoenix Contact, CODESYS, Bosch Rexroth, Red Lion
Scanning/discovery	scanner, discover, vulnerability scanner, scan module
Penetration testing	penetration testing tool, security testing tool
Traffic generation	traffic generator, fuzzer
Attack toolkits	vulnerability exploitation framework, attack toolkit, exploit kit

Appendix B.
Datasets

DS_1. Collected by Lupia *et al.* [5], [38], DS_1 originates from a HoneyICS [6] deployment with three emulated PLCs exposed individually through public academic IPs. The dataset contains 3,818 Modbus packets, captured between June and September, 2023.

DS_2. Collected by Ondrikov *et al.* [16], [39], DS_2 combines traffic from two parallel honeypot deployments: Conpot and HoneyICS, the latter including a real PLC alongside two emulated devices. These deployments were exposed across four environments: three cloud locations (Europe, United States, Asia) and one corporate network, resulting in sixteen public IPs. The Conpot deployment with default parameters runs in Modbus serial mode, i.e., it does not respond to requests for unit ID 0 (see Section 4.3.1). We include this traffic to reflect a common honeypot setup and to demonstrate the robustness of ICTOOLS under these conditions. To account for differences among deployment conditions, we split DS_2 by honeypot type and network type: DS_2-ClH (HoneyICS/cloud), DS_2-ClC (Conpot/cloud), DS_2-CoH (HoneyICS/corporate) and DS_2-CoC (Conpot/corporate). The four subsets include 17,561, 3,766, 7,254, and 2,670 Modbus packets captured between November 2024 and May 2025.

DS_3. Collected by the authors, DS_3 originates from a deployment of a single real Rockwell CompactLogix PLC, exposed via an IP address associated with a popular ISP

Table 12. DATASET SUMMARY.

	DS_1	**DS_2-ClH**	**DS_2-ClC**	**DS_2-CoH**	**DS_2-CoC**	**DS_3**
Protocol packets*	3783	17 181	3366	2273	1445	7601
Unique IPs*	481	2341	1283	533	527	1163
Interactions**						
before preprocessing	1293	5671	2132	3376	991	2498
with packet loss	0	1	0	2516	81	0
invalid	8	318	258	43	103	7
after preprocessing	1285	5352	1874	817	807	2491

*After preprocessing. **Using the δ calculated in Section 5.2

and not linked to a specific institution. This dataset contains 8,623 ENIP packets, captured from February to June and from August to November 2025.

A summary for the three used datasets is reported in Table 12. Statistics are computed using ICTOOLS, as described in Section 5.2.

B.1. Preprocessing

To ensure that only complete interactions are retained for analysis (see Section 3.2.3, further discussed in Section 6.2.1), we perform an initial investigation of the interactions created by ICTOOLS and their raw packet captures. We notice that some interactions in the two DS_2 corporate datasets show packet loss. Moreover, interactions created by ICTOOLS include invalid cases from incorrect honeypot behavior (e.g., responding to non-Modbus packets) or errors in Zeek's parsing (i.e., occasionally misinterpreting non-Modbus packets as Modbus). Since these issues stem from deployment limitations rather than ICTOOLS, we filter out interactions with packet loss (using Zeek's indicator of bytes missed in a stream), interactions consisting solely of invalid function code requests or solely of responses, and interactions with more responses than requests. This preserves partially valid interactions and those from unmodified Conpot deployments, which do not respond to valid Modbus requests for unit ID 0.

Appendix C. Capability to Modbus Function Mapping

Table 14 presents the mapping of capabilities to the Modbus function codes and their names.

Table 13. FILTERED KEYWORDS AND URLS.

Category	Filters
Keywords	pdf, support, faq, article, blog, post, reddit, community, forum, document, docs, publication, news, threads, tiktok, quora, youtube, bulletin, researchgate, slideshare, linkedin, aliexpress, help, ebay, instagram, resume, library, drive, manual, tutorial, stackoverflow, medium, sid.siemens, pcap, wiki
Urls	phoenixcontact.com, learn.microsoft.com, dl.acm.org, dokumen.pub, mdpi.com, academia.edu, eaton.com, hms-networks.com, industrial.omron.eu, se.com, beijerelectronics.com, rockwellautomation.com, yokogawa.com, automationdirect.com, new.abb.com, moxa.com, bannerengineering.com, wago.com, scribd.com, amazon.com, link.springer.com, modbus.org, archive.org, en.wikipedia.org, keyence.com, store.codesys.com, process.honeywell.com, endress.com, us.rs-online.com, siemens.com, deltaww.com, picclick.com, prosoft-technology.com, upwork.com, mitsubishielectric.com, eurotherm.com, digikey.com, vector.com, industrialcyber.co, redlion.net, predig.com, apc.com, nxp.com, slideplayer.com, teltonika-networks.com, tme.eu, mrplc.com, rtautomation.com, seneca.it, opto22.com, belden.com, br-automation.com, us.endress.com, indiamart.com, sciencedirect.com, emerson.com, press.siemens.com, automation24.com, mall.industry.siemens.com

Table 14. MAPPING OF CAPABILITIES TO MODBUS FUNCTION CODES.

Capability	Modbus functions
Diagnostics & Identification	07 (Read Exception Status), 08 (Diagnostics), 11 (Get Comm Event Counter), 12 (Get Comm Event Log), 17 (Report Slave ID), 43/14 (Read Device Identification), 90/02 (READ_ID), 90/03 (READ_PROJECT_INFO), 90/04 (READ_PLC_INFO), 90/06 (READ_CARD_INFO), 90/57 (READ_ETH_MASTER_DATA), 90/80 (MONITOR_PLC), 90/88 (CHECK_PLC), 90/115 (GET_STATUS_MODULE)
Data Reading	01 (Read Coils), 02 (Read Discrete Inputs), 03 (Read Holding Registers), 04 (Read Input Registers), 20 (Read File Record), 24 (Read FIFO Queue), 90/32 (READ_MEMORY_BLOCK), 90/34 (READ_VARIABLES), 90/36 (READ_COILS_REGISTERS), 90/112 (READ_IO_OBJECT)
Data Writing	05 (Write Single Coil), 06 (Write Single Register), 15 (Write Multiple Coils), 16 (Write Multiple Registers), 21 (Write File Record), 90/35 (WRITE_VARIABLES), 90/37 (WRITE_COILS_REGISTERS), 90/113 (WRITE_IO_OBJECT), 22 (Mask Write Register), 23 (Read/Write Multiple Registers)
Advanced Communication	43/13 (CANopen General Reference Request and Response PDU), 90/01 (INIT_COMM), 90/10 (REPEAT), 90/16 (TAKE_PLC_RESERVATION), 90/17 (RELEASE_PLC_RESERVATION), 90/18 (KEEP_ALIVE), 90/64 (START_PLC), 90/65 (STOP_PLC)
Process Modification	90/48 (INITIALIZE_UPLOAD), 90/49 (UPLOAD_BLOCK), 90/50 (END_STRATEGY_UPLOAD), 90/51 (INITIALIZE_DOWNLOAD), 90/52 (DOWNLOAD_BLOCK), 90/53 (END_STRATEGY_DOWNLOAD)