# Complementary Approaches to Instructable Agents for Advanced Persistent Threats Detection

**Juan HUANG[1,2], Zhemin AN[1,2], Steven MECKL[1], Gheorghe TECUCI[1,2]*, Dorin MARCU[1]**

[1] Learning Agents Center

[2] Department of Computer Science, George Mason University, Fairfax, Virginia, 22030, USA
jhuang21@gmu.edu, zan2@gmu.edu, smeckl@gmu.edu, tecuci@gmu.edu (**Corresponding author*), dmarcu@gmu.edu

**Abstract:** Large CSOCs (cybersecurity operation centers) must analyze tens of thousands of security incidents per day. Not only that there are not enough cybersecurity analysts available but the average cost of a cybersecurity analyst keeps going up. This paper presents a novel approach to the detection of APTs (advanced persistent threats), where an expert cybersecurity analyst directly teaches (rather than programs) a cognitive agent how to investigate cybersecurity alerts, as the analyst would teach a student, through explained examples of investigations. It then presents two complementary instantiations of this approach, as implemented in ADONIS (Automating the ATT&CK™-based Detection Of Novel Network Intrusions System) and CAAPT (Cognitive Agent for APT detection). ADONIS detects adversary's behavior in terms of MITRE's ATT&CK (Adversarial Tactics, Techniques & Common Knowledge), independent of specific malware and tools. It can therefore detect novel intrusions, but is expected to be less efficient because of the multitude of tactics and techniques that can be employed. CAAPT only detects known malware based on combinations of weak IOCs (indicators of compromise) and, as demonstrated by the experimental results, is efficient. Therefore, once a new malware is detected with ADONIS, its IOCs can be identified and CAAPT can be trained to rapidly detect it. This instructable agents approach promises to significantly reduce the cost of operating the CSOCs and improve their detection performance by automating much of the analysts' investigative activity. It increases the probability of detecting intrusion activity and reduces the false positive detections presented to the analysts who can spend their time on more complex tasks and on teaching the agents.

**Keywords:** Cybersecurity, Intrusion detection, Instructable agent, Evidence-based reasoning, Artificial intelligence.

## 1. Introduction

Network intrusion detection is currently done through a combination of automated and manual techniques. Automated tools use near-real-time data sources, such as host-based and network-based activity logs, along with a variety of detection strategies based on IOCs, rules, anomaly detection, and machine learning to trigger alerts for potentially malicious activity that cybersecurity analysts must investigate. These alerts are good at identifying new attacks but have a high false positive rate. Manual analysis of alerts requires a high level of skill and is resource intensive. The analysts must be skilled at collecting and examining a wide variety of data including disk and memory artifacts, logs, and raw network packet data to determine if an alert is valid. Large CSOCs must analyze tens of thousands of security incidents per day. Not only that there are not enough cybersecurity analysts available but the average cost of a cybersecurity analyst keeps going up.

This paper presents a novel approach to the detection of APTs, where an expert cybersecurity analyst directly teaches (rather than programs) a cognitive agent how to investigate cybersecurity alerts. Section 2 summarizes this approach and presents the overall architecture of the instructable agent. Then Section 3 presents the instantiation of this approach into ADONIS that detects malware based on its behavior expressed in terms of the employed tactics and techniques. Section 4 presents a complementary instantiation of this approach into CAAPT that detects malware based on combinations of weak IOCs. Section 5 presents the evaluation of CAAPT. Finally, Section 6 compares ADONIS and CAAPT, and discusses the expected benefits of the proposed approach to intrusion detection.

## 2. Instructable Agents for APT Detection

An instructable agent for APT detection is a cognitive agent that is directly taught how to investigate cybersecurity alerts by an expert cybersecurity analyst, as the analyst would teach a student, through explained examples of investigations. Once trained, this agent can investigate alerts as the human analyst would, both *autonomously* to allow an *"on the loop"* supervision by an analyst, and *interactively* with the user *"in the loop,"* as a trusted collaborator.

https://doi.org/10.24846/v29i3y202001

This approach builds directly on the research on a computational theory and technology for the development of instructable cognitive agents that are taught rather than programmed (Tecuci, 1988; Tecuci 1998; Boicu et al., 2000; Tecuci et al., 2000; Boicu et al., 2001; Tecuci et al., 2002a; Tecuci et al., 2016a). These cognitive agents can learn complex problem-solving expertise directly from human experts, can support experts and non-experts in problem solving and decision making, can autonomously perform the learned tasks, and can teach their problem-solving expertise to students. Because the resulting agent learns to replicate the problem-solving behavior of its human expert, it is called a Disciple agent. Exemplary Disciple agents have been demonstrated in many domains, including critical thinking education (Tecuci & Keeling, 1999), course of action critiquing (Tecuci et al., 2001), center of gravity analysis (Tecuci et al., 2002b; Tecuci et al., 2005; Tecuci, Boicu & Comello, 2008), intelligence analysis (Tecuci et al., 2008; Tecuci et al., 2011; Tecuci et al., 2016b; Tecuci et al., 2018), and intelligence, surveillance and reconnaissance (Tecuci et al., 2019).

The instructable agent has two main components, a *Mixed-Initiative Learning and Reasoning Assistant* shown in Figure 1, and an *Autonomous Multi-Agent Reasoner* shown in Figure 2.

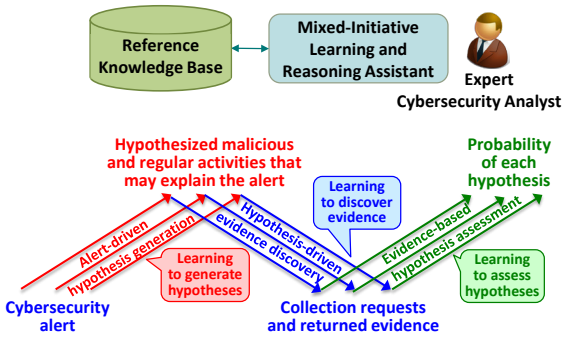



**Figure 1.** Mixed-initiative teaching and learning

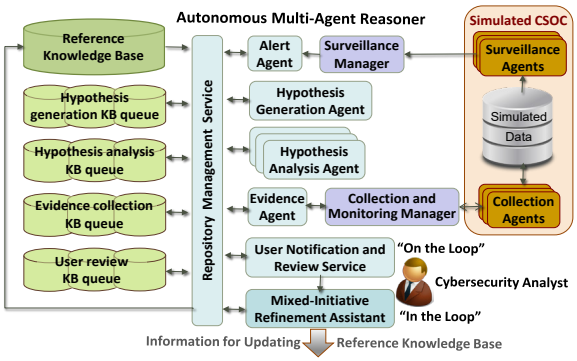



**Figure 2.** Autonomous intrusion detection

A cybersecurity analyst directly teaches the Learning and Reasoning Assistant how to investigate cybersecurity alerts and the assistant develops the Reference Knowledge Base (KB), enabling it to automatically perform similar investigations. The analyst demonstrates the investigation of a specific cybersecurity alert by following the scientific method of hypothesis generation and testing used in intelligence analysis (Tecuci et al., 2016b). First, during *alert-driven hypothesis generation*, the analyst employs *abductive reasoning* that shows that something is *possibly* true, to imagine the possible malicious and non-malicious (regular) activities that may have caused the alert. Then, during *hypothesis-driven evidence discovery*, the analyst employs *deductive reasoning* that shows that something is *necessarily* true, to successively decompose each of these hypothesized activities into simpler and simpler hypotheses. The simplest hypotheses point to evidence that may support either their truthfulness or their falsehood. After that, specialized collection agents are invoked to look for this evidence on the host and network computers. Finally, during *evidence-based hypothesis testing*, the analyst employs *inductive reasoning* that shows that something is *probably* true, to test each hypothesis based on the discovered evidence.

The *Mixed-Initiative Learning and Reasoning Assistant* learns rules for generating hypotheses, rules for evidence collection, and rules for analyzing hypotheses, as ontology-based generalizations of the reasoning steps demonstrated by the analyst (Tecuci et al., 2016a). These rules are stored into the Reference Knowledge Base.

This instructable agent approach is based on apprenticeship learning from examples demonstrated by the instructor. However, as opposed to other instructable agent approaches, such as PLOW (Allen et al., 2007) or LIA (Azaria, Krishnamurthy & Mitchell, 2016) that rely on inductive learning and natural language processing, this approach employs multistrategy learning (Tecuci, 1993; Michalski & Tecuci, 1994) that synergistically integrates inductive learning from examples, learning from explanations, and learning by analogy, in the context of a multi-agent architecture.

## 3. Intrusion Detection Based on Attacker Tactics and Techniques

A transformative development in intrusion detection is represented by MITRE's ATT&CK framework that identifies the tactics and techniques employed by adversaries (Strom et al., 2017). The *tactics* represent the adversary's objectives for performing its actions, such as persist on the attacked system, escalate privileges, evade detection, access credentials, discover information, move laterally, execute files, collect information, exfiltrate data, and command and control.

The *techniques* describe the actions adversaries take to achieve their tactical objectives. Within each tactic category, such as that of executing files, there are a finite number of actions that will enable the adversary to achieve its objective, such as service execution, Windows management instrumentation, and scheduled task. Many of the ATT&CK techniques are legitimate system functions that can also be used for malicious purposes, as opposed to IOCs that are indications of actions known to be caused by, or under the influence of an adversary. Moreover, the ATT&CK techniques represent behavior exhibited by an adversary through remote access tools, scripts, or interaction with a command-line interface, independent of specific adversary malware and tools that are likely to change over time (Strom et al., 2017).

During an intrusion, an adversary decides at every step which technique to use based on its knowledge, information obtained about the target environment, information needed for future actions, and capabilities currently available (Strom et al., 2017). This provides the opportunity to detect patterns of adversary behavior as combinations of specific tactics and techniques. As illustrated in the following sections, a cybersecurity expert can instruct ADONIS to automatically detect such patterns.

### 3.1 Cybersecurity Ontology

When the cybersecurity analyst models the detection process for a specific malware, s/he (assisted by a knowledge engineer) also identifies the ontological knowledge needed by the assistant to automatically perform the same analysis. This results in an ontology specification. An initial ontology is then developed based on this ontology specification, existing cyber ontologies (Obrst, Chase & Markeloff, 2012), CSOC's network configuration, and threat intelligence. The ontology language is an extension of the Resource Description Framework Schema, or RDFS (Allemang & Hendler, 2011; W3C, 2014) with additional features to facilitate learning and evidence representation (Tecuci et al.,, 2016a). Figure 3 shows a fragment of the ontology of ADONIS that is used both as a generalization hierarchy for learning, and for reasoning in conjunction with learned ontology-based rules.

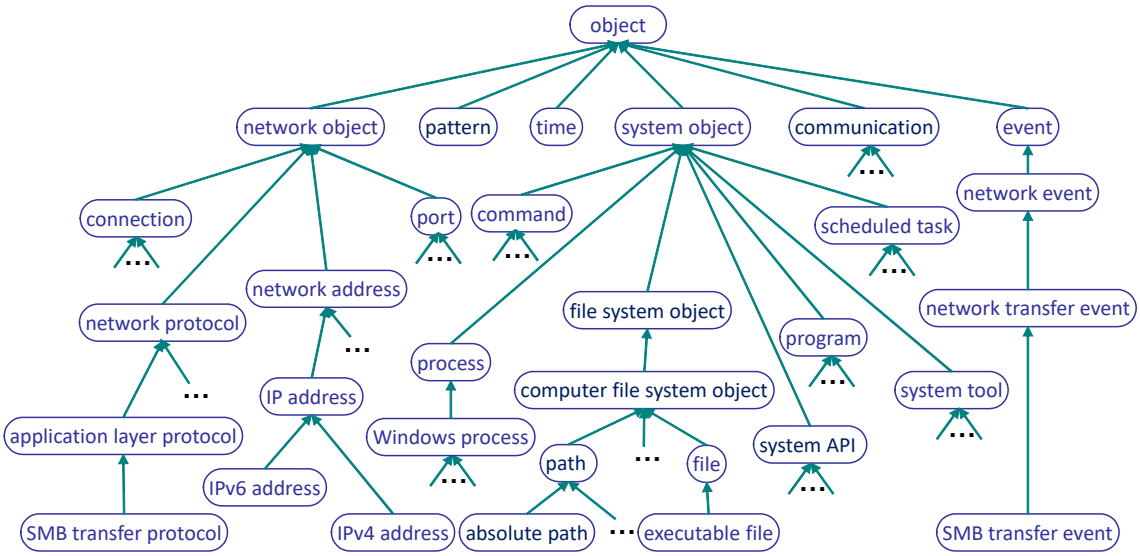



**Figure 3.** Fragment of the cybersecurity ontology used as generalization hierarchy

## 3.2 Hypothesis Generation and Learning

The agent's instruction starts with a cybersecurity alert. These alerts can come from a variety of IDSs (intrusion detection systems), including network-based and host-based IDSs, anti-virus software, endpoint detection and response tools, or complex detection rules built into security information and event management systems. In this illustration the alert was generated by the Zeek IDS (2020) and converted into the JSON (Java Script Object Notation) format which was then represented both as an ontology fragment and as a natural language phrase, as shown at the bottom of Figure 4 (SMB stands for server message block). Next the cybersecurity analyst needs to imagine a malicious activity that may have caused this alert and formulate a question (shown at the top of Figure 4) whose answer is this activity. The other answers (of which one is shown) are alternative activities that may also have caused the alert. From this demonstration of hypothesis generation the agent learns an *alert rule*, an *indicator rule*, and two *question rules*. These rules will enable the agent to generate alternative hypotheses from similar alerts.

## 3.3 Evidence Discovery and Learning

The two hypotheses from the top of Figure 4 (an intrusion hypothesis and a false-positive one) need to be analyzed to determine which one is the most likely. This requires more evidence. The analyst needs to show to the agent how such evidence can be discovered by putting each of the hypotheses to work, as part of the *hypothesis-driven evidence discovery* phase. In essence, each hypothesized activity is successively decomposed into simpler and simpler ones representing specific tactics and techniques which point to evidence that may support their truthfulness. Figure 5 shows the decomposition of the attack hypothesis into two tactics, LATERAL MOVEMENT TACTIC and EXECUTION TACTIC. The Lateral Movement hypothesis is supported by the evidence represented by the alert. However, to further decompose the Execution tactic into corresponding techniques, one needs to identify the process that was created from executable file1. Therefore, a collection request for this process is transmitted to a specialized collection agent that will identify and return process1. From the demonstrated decomposition in Figure 5, the agent learns a *hypothesis analysis rule* and an *evidence collection rule*.

The discovery of process1 enables the decomposition of the EXECUTION TACTIC into

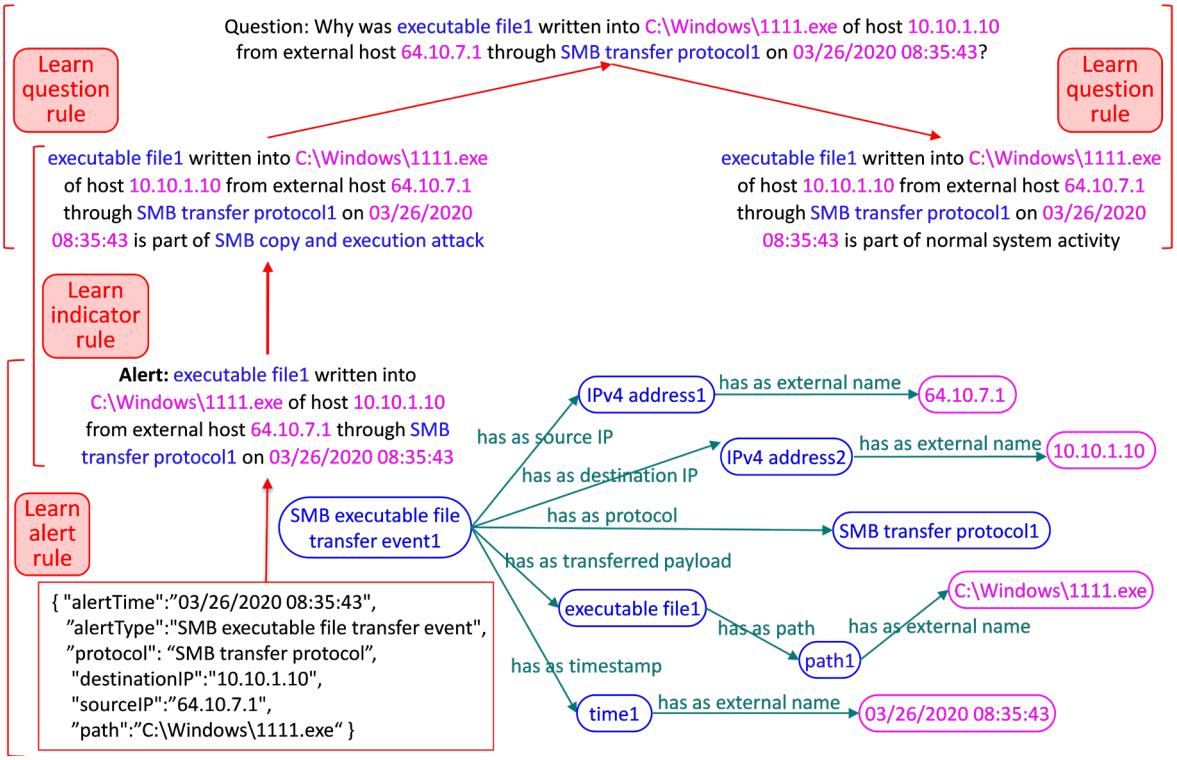



**Figure 4.** Instructing the agent to generate hypotheses

three techniques, SERVICE EXECUTION TECHNIQUE, WINDOWS MANAGEMENT INSTRUMENTATION TECHNIQUE, and SCHEDULED TASK TECHNIQUE, as illustrated in Figure 6. Each technique is further decomposed into simpler hypotheses that point to evidence that may support its truthfulness. Then corresponding collection requests are formulated for specialized collection agents. For example, the Windows Management Instrumentation technique is decomposed into two sub-hypotheses. The second one (i.e. "WMIC command1 is executed in command-line on host 10.10.1.10 through remote procedure call on 3/27/2020 05:07:25") has a favoring argument (that favors its truthfulness) under the left (green) square, consisting of the conjunction of two sub-hypotheses. It also has a disfavoring argument (that disfavors its trustfulness) under the right (pink) square: if the WMIC command1 is executed **locally** on host 10.10.1.10, then it is not executed in command-line through remote procedure call.

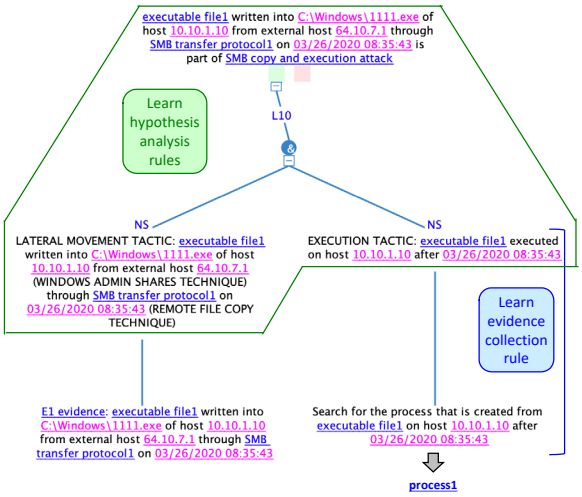



**Figure 5.** Hypothesis-driven evidence discovery and rule learning

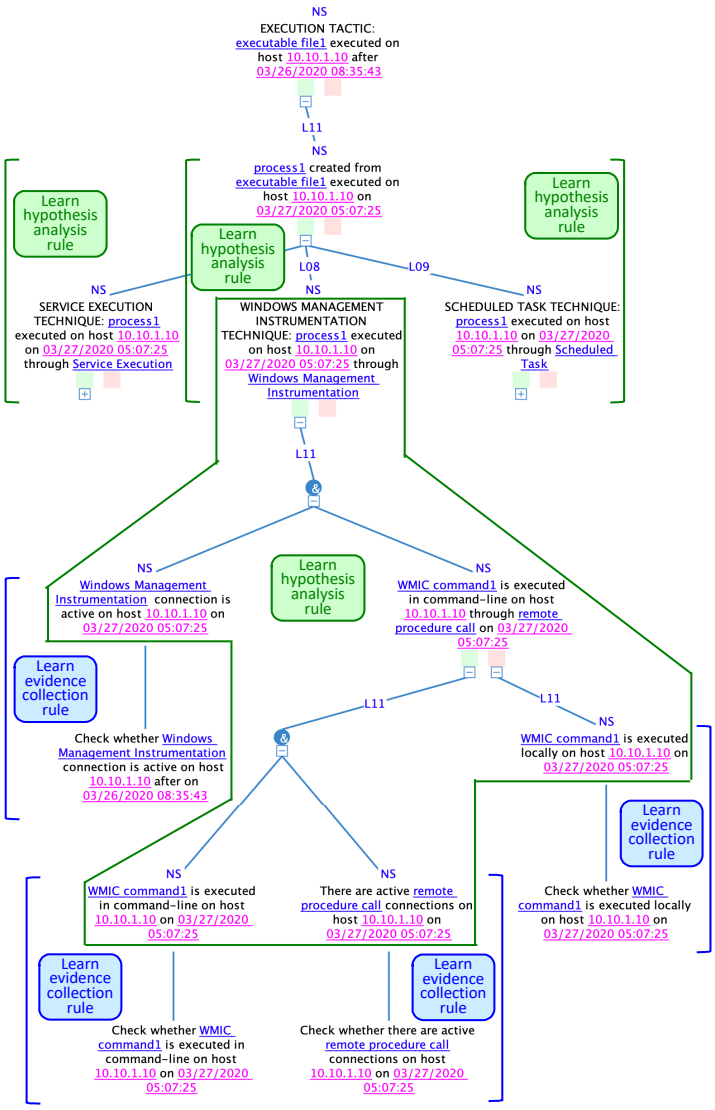



**Figure 6.** Resumed hypothesis-driven evidence discovery and learning

## 3.4 Multistrategy Rule Learning

The vast majority of the current machine learning approaches are heavily statistical and learn single functions from a large number of examples. Such approaches are not applicable for learning analysis rules because large sets of examples to learn from do not exist and would be very difficult to create. Instead, the cybersecurity analyst just points directly to the relevant features of the instances in the example, such as process1 runs on 10.10.1.10, and has as time stamp 3/27/2020 05:07:25. Thus, these features do not need to be discovered through the statistical comparison of a large number of positive and negative examples (that are not available anyway), as done in the inductive learning methods (Alpaydyn, 2020).

From the demonstrated reasoning in Figure 6, the agent learns three *hypothesis analysis rules*, each reducing the Execution tactic to an execution technique, as shown on top of the figure. Then it learns other hypothesis analysis rules, corresponding to these techniques, as well as the corresponding evidence collection rules.

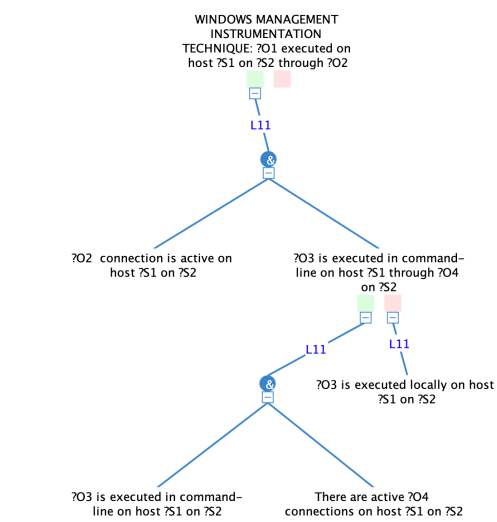
In particular, from the decomposition of the Windows Management Instrumentation hypothesis, shown inside the polygon in Figure 6, the agent learns the hypothesis analysis rule shown in Figure 7. First, the agent replaces each example-specific instance with a different variable (i.e., process1 with ?O1, 10.10.1.10 with ?S1, and 3/27/2020 05:07:25 with ?S2). Then it generates an ontology-based applicability condition for the resulting reasoning pattern that shows the possible values that these variables may take.

Notice however that, instead of a single applicability condition, the agent learns a lower bound and an upper bound for this condition, by using two complementary learning strategies.

The lower bound is generated by employing the strategy of a ***cautious learner*** that wants to minimize the chances of making mistakes when employing the learned rule. In this case the lower bound of ?O1 is obtained as a minimal generalization of process1 in the agent's ontology, which is Windows process. That is, ?O1 can be instantiated by any instance of a Windows process. This strategy increases agent's confidence in the correctness of its reasoning, but the agent may fail to apply the reasoning pattern in situations where it is applicable.

The upper bound of the condition is generated by employing the strategy of an ***aggressive learner*** that wants to maximize the opportunities of employing the learned rule. In this case the upper bound of ?O1 is obtained as a maximal generalization of process1 which is system object. This strategy increases the number of situations where the rule can be applied, but in some of them the reasoning may not be correct.

The two bounds may be refined, and may even become identical, based on additional examples encountered by the agent during its autonomous analysis of new alerts.




**Main Condition Bounds**

| Var | Lower Bound | Upper Bound |
|---|---|---|
| ?O1 | Windows process | system object |
| ?O2 | Windows Management Instrumentation | Windows Management Instrumentation |
| ?O3 | WMIC command1 | system object |
| ?O4 | remote procedure call | network object |
| ?O5 | host | network object |
| ?O6 | IPv4 address | network object |
| ?O7 | time | time |
| ?S1 | any string | any string |
| ?S2 | any string | any string |

| Var | Relationship | Var |
|---|---|---|
| ?O1 | runs on | ?O5 |
| ?O1 | has as timestamp | ?O7 |
| ?O3 | may generate | ?O1 |
| ?O3 | may be generated through | ?O4 |
| ?O5 | has as IP address | ?O6 |
| ?O6 | has as external name | ?S1 |
| ?O7 | has as external name | ?S2 |

**Figure 7.** Hypothesis analysis rule learned from the example argumentation in Figure 6

# 4. Intrusion Detection Based on Combinations of Weak IOCs

Many attacker groups practice *evolutionary development* to adapt their malware to changes in network defense technology or simply to

increase efficiency. These changes in the way malware presents itself on the network and on disk have made it difficult for signature-based intrusion detection tools to detect attacks because the attackers can change static information in their malware faster than defenders can adapt. However, the patterns of behavior change more slowly and with less variance.

CAAPT can be trained to learn patterns of IOCs and can anticipate the changes present in the malware. If one aspect of the malware's behavior changes and becomes undetectable by the agent, the malware can still be detected with some probability based on the remaining observable evidence. This partial detection can further be used, in collaboration with an analyst, to identify the changes in the malware family by searching for additional observable behavior, to extend the initial detection pattern. This cycle of learning, applying, and extending patterns allows the agent to adapt as the malware family evolves.

The previous sections have illustrated the process of instructing ADONIS, which is similar to that of CAAPT. The next subsection will illustrate the hypothesis testing process.

## Evidence-based Hypothesis Testing

Figure 2 presents the autonomous intrusion detection process that will be illustrated in the following. The Zeek IDS generated an alert. Its JSON representation is sent to the Alert Agent by a Surveillance Agent through the Surveillance Manager. The Alert Agent employs a (previously learned) alert rule to generate the ontological representation of this alert and the following basic hypothesis:

Suspicious connection1 from 10.10.1.20 (port 56902) to 10.10.4.101 (port 53) at 08/29/2019 11:42:23 AM, using known APT1 domain ubuntuguru.strangled.net

The Hypotheses Generation Agent employs an indicator rule to abductively generate the following intrusion hypothesis:

connection1 from 10.10.1.20 (port 56902) to 10.10.4.101 (port 53) at 08/29/2019 11:42:23 AM, using known APT1 domain ubuntuguru.strangled.net, is part of APT1 intrusion.

Then it uses three question rules to generate a question and two other alternative hypotheses. After that, the Hypothesis Analysis Agent and the Evidence Agent generate Wigmorean argumentations (Wigmore, 1913; Schum, 2001; Tecuci et al.,, 2016b), like those in Figure 5 and Figure 6, for assessing the generated hypotheses. In a Wigmorean argumentation, the hypothesis to be assessed is decomposed into simpler sub-hypotheses by considering both favoring arguments (supporting the truthfulness of the hypothesis), and disfavoring arguments (supporting the falsehood of the hypothesis). Each argument is an independent strategy of showing that the hypothesis to be assessed is true or false, and is characterized by a specific relevance or strength. An argument consists either of a single sub-hypothesis or a conjunction of sub-hypotheses. The symbol "*" is used to denote a combined indicator representing all the possible combinations of a set of sub-hypotheses (e.g., H1*H2 = H1 or H2 or H1&H2).

The agents employ an intuitive system of Baconian probabilities (Cohen, 1977) with Fuzzy qualifiers (Negoita & Ralescu, 1975; Zadeh, 1983) which are shown in Table 1. Notice that the probability intervals are associated with intuitive names, such as likely (60-65%) or almost certain (95-99%).

**Table 1.** Probability scale

| | | |
|---|---|---|
| L11 | 100% | certain |
| L10 | 95-99% | almost certain |
| L09 | 90-95% | very likely+ |
| L08 | 85-90% | very likely |
| L07 | 80-85% | very likely- |
| L06 | 75-80% | more than likely+ |
| L05 | 70-75% | more than likely |
| L04 | 65-70% | likely+ |
| L03 | 60-65% | likely |
| L02 | 55-60% | likely- |
| L01 | 50-55% | barely likely |
| L00 | 0-50% | lacking support |

A fragment of the generated Wigmorean argumentation for assessing the above intrusion hypothesis is shown in Figure 8. The probabilities of the hypotheses are assessed from bottom-up. First the probabilities of the leaf hypotheses are assessed based on the three credentials of the corresponding evidence: credibility, relevance, and inferential force. The credibility of evidence answers the question: What is the probability that the evidence is true? The relevance of evidence to a hypothesis answers the question: What would be the probability of the hypothesis if the evidence were true?" The inferential force or

weight of the evidence on the hypothesis answers the question: What is the probability of the hypothesis, based only on this evidence? Obviously, an irrelevant item of evidence will have no inferential force. An item of evidence that is not credible will have no inferential force either. Only an item of evidence that is both relevant and credible would indicate that the hypothesis is true. Consistent with both the Baconian and the Fuzzy min/max probability combination rules, the inferential force of an item of evidence on a hypothesis is determined as the minimum between its credibility and its relevance. When there is only one item of favoring evidence for a hypothesis, its inferential force on the hypothesis is also the probability of the hypothesis. In general, however, the probability of the hypothesis would be the result of the balance of probabilities between the combined inferential force (maximum) of the favoring evidence items (represented under the left green square) and the combined inferential force of the disfavoring items (represented under the right pink square).

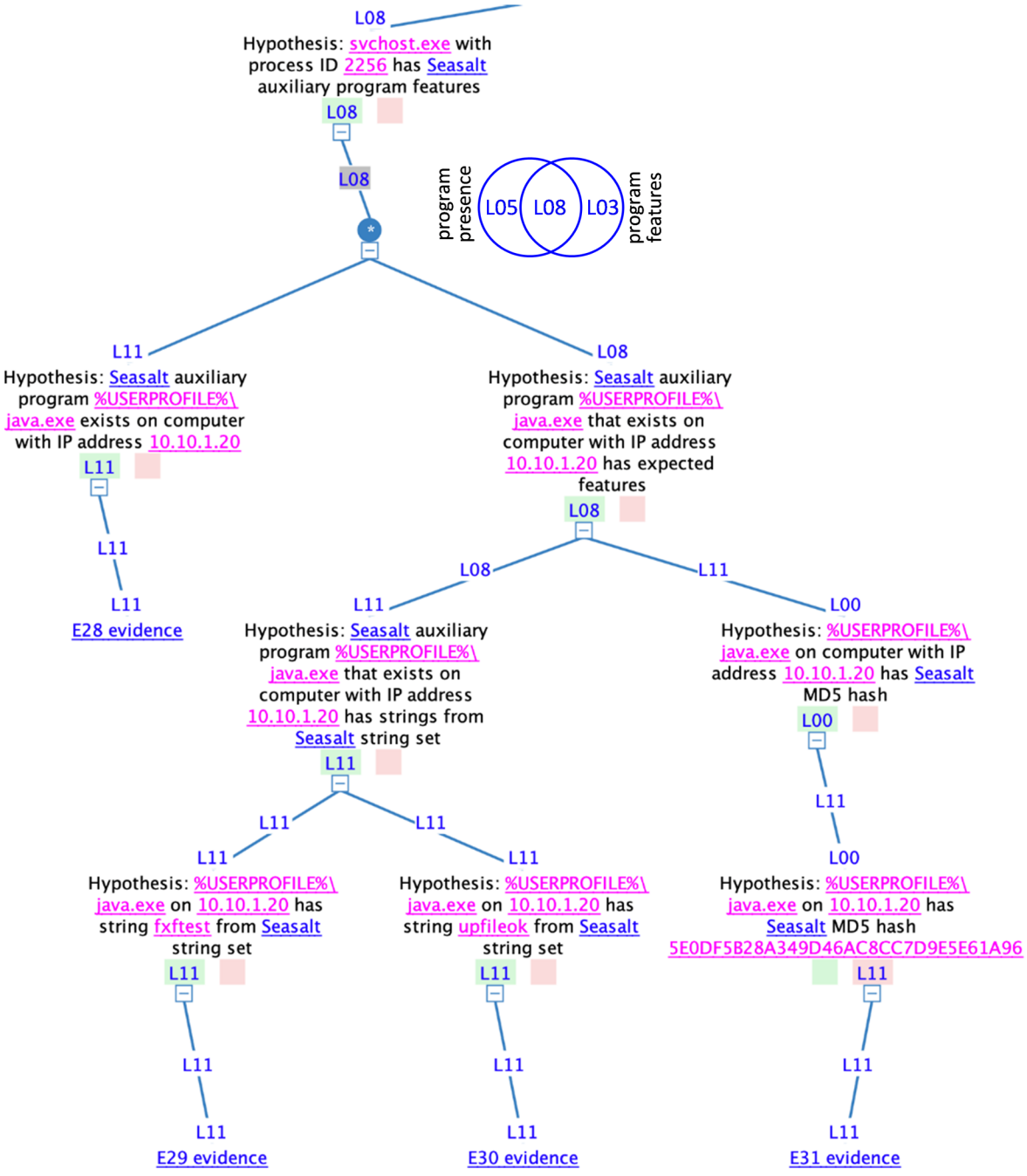



**Figure 8.** Fragment of the analysis of a variant of the Seasalt malware

Once the probabilities of the bottom-level hypotheses have been determined based on evidence, the probabilities of the upper level hypotheses are computed based on the logical structure of the Wigmorean argumentation (conjunctions and disjunctions of hypotheses), using the min-max probability combination rules common to the Fuzzy probability view and the Baconian probability view. These rules are much simpler than the Bayes rule used in the Bayesian probability view (Schum, 2001), or the Dempster-Shafer rule in the Belief Functions probability view (Shafer, 1976).

Notice that the top argument in Figure 8 is a combined indicator (*) argument. If there is evidence of both indicators, their relevance is very likely L08 (85-90%). If there is evidence only for the left-hand side indicator, the relevance is more than likely L05 (70-75%), and if there is evidence only for the right-hand side indicator, the relevance is likely L03 (60-65%). As a result, the probability of the top hypothesis is assessed as the minimum between the relevance of the combined indicator and the probabilities of the sub-hypotheses.

# 5. Evaluation

Evaluating a system like ADONIS or CAAPT is challenging due to a lack of standardized data for use when comparing it with other systems or approaches. It is also challenging due to a lack of similar systems. It is a novel approach with respect to both autonomous evidence-based reasoning in general and APT detection in particular. As such, the only reasonable approach to compare them would be against manual analysis by an expert, but even this is problematic because of lack of data on manual analysis. Currently only CAAPT has been evaluated.

APT1 was chosen as the attack group for this research primarily because of the abundance of freely available unclassified information about it, including IOCs, malware samples, and details of how the group operated (Mandiant, 2013).

Experiments were designed and performed to test both the training of CAAPT and its ability to detect configuration changes in the same malware and new malware versions as the attackers' tool set evolved over time (Meckl et al.,, 2017; Meckl, 2019). The experiments simulated a subset of the historical evolution of the APT1 malware: *Auriga → Auriga variants → Bangat → Bangat variants → Seasalt → Seasalt variants → Kurton → Kurton variants.*

The evaluation experiment started with developing a cyber ontology which, as opposed to the broad and malware-agnostic ontology of ADONIS, was focused on the representation of the APT1 malware and its IOCs. Then CAAPT was trained to analyze the Auriga malware, based on the expertise of Steven Meckl, and its detection capabilities were tested in the sequence of scenarios discussed below.

## 5.1 Auriga Experiment

Scenario 1a consisted of an intrusion with the Auriga malware used in training, to create a baseline for the evaluation. As shown in Table 2, CAAPT detected that very likely (85-90%) there is an intrusion by Auriga or by an unspecified member of the APT1 family (including Auriga).

**Table 2.** Results of the Auriga experiment

| Intrusion By | Detected Intrusion | Duration |
|---|---|---|
| 1a: Auriga malware used in training | Auriga (85-90%) | 143 seconds |
| | APT1 (85-90%) | |
| 2a: Variant of the Auriga malware | Auriga (85-90%) | 121 seconds |
| | APT1 (85-90%) | |
| 3a: Bangat malware | Auriga (75-80%) | 119 seconds |
| | APT1 (85-90%) | |

Scenario 2a consisted of an intrusion by a variant of Auriga that used a different APT1 domain to trigger the security alert, and the malware process %SYSTEMROOT%\Temp\svchost.exe did not contain unique APT1 strings. In this case also CAAPT detected that very likely (85-90%) there is an intrusion by Auriga or by an unspecified member of the APT1 family.

Scenario 3a consisted of an intrusion by the Bangat malware for which CAAPT was not trained. Bangat does not use the library files riodrv32.sys and netui.dll, instead, it uses a different regular expression for its temporary file names, stores its data files in different folders, and uses different Windows Service names for its persistence mechanisms. This time CAAPT

detected that very likely (85-90%) there is an APT1 intrusion, but the probability of being Auriga is only 75-80%.

The last column in Table 2 shows the duration of each experiment. The run time for the generation and evaluation of the analyses was between 1 and 3 seconds. Most of the time was spent waiting for the Collection and Monitoring Manager to return the results requested from the collection agents.

## 5.2 Bangat Experiment

The second experiment modelled the detection of the Bangat intrusion from the first experiment, and extended the APT1 ontology with the representation of the Bangat malware as follows:

1. The experiment started with the KB generated by the last Auriga experiment (Bangat intrusion with the Auriga representation and the rules learned from the Auriga modelling);

2. The ontological representation of Bangat was added into this KB;

3. The analysis of the Bangat intrusion was generated using the ontological representation of Bangat and the rules were learned from the Auriga modelling;

4. The generated analysis was refined and extended to accurately and completely analyze the Bangat malware;

5. New rules to analyze Bangat were learned;

6. CAAPT's detection capabilities were tested in three scenarios, as in the Auriga experiment.

Scenario 1b consisted of an intrusion with the Bangat malware used in training (the one from the Auriga experiment). As shown in Table 3, CAAPT detected that very likely (85-90%) there was an intrusion by Bangat or by an unspecified member of the APT1 family. The probability of being Auriga was more than likely+ (75-80%). This is not a contradiction because these hypotheses are not disjoint. The Bangat malware is an evolution of the Auriga malware and therefore it has many features in common with Auriga. When checking for an intrusion with Auriga, the system looks for the presence of the features of the Auriga malware on the infected computer, but some of these features are also the features of Bangat, so it is possible that the computer is infected by both Auriga and Bangat.

Therefore, Auriga intrusion with probability more than likely+ (75-80%) covers the case where the Auriga intrusion is accompanied by a Bangat intrusion. Similarly, Bangat intrusion with probability very likely (85-90%) is based on the detected Bangat features on the host computer which also includes some Auriga features. Thus, this probability also covers the case when there is both a Bangat and an Auriga intrusion.

**Table 3.** Results of the Bangat experiment

| Intrusion By | Detected Intrusion | Duration |
|---|---|---|
| 1b: Bangat malware used in training | Auriga (75-80%) | 265 seconds |
| | Bangat (85-90%) | |
| | APT1 (85-90%) | |
| 2b: Variant of the Bangat malware | Auriga (75-80%) | 228 seconds |
| | Bangat (85-90%) | |
| | APT1 (85-90%) | |
| 3b: Seasalt malware | Auriga (50-55%) | 274 seconds |
| | Bangat (50-55%) | |
| | APT1 (85-90%) | |

Scenario 2b consisted of an intrusion with a variant of Bangat that had three differences. The alert was triggered with a different domain, the data files used in the first Bangat scenario were not present, and a different temporary file matching the Bangat regular expression was present on the infected host. CAAPT again detected that very likely (85-90%) there was an intrusion by Bangat or by an unspecified member of the APT1 family, and more than likely+ (75-80%) by Auriga.

Scenario 3b consisted of an intrusion by another member of the APT1 family, the Seasalt malware, for which CAAPT was not trained. Seasalt added an auxiliary program, which was started by the Seasalt Windows Service DLL, and the network protocol was changed so it could be detected using a unique HTTP User Agent String. CAAPT detected that very likely (85-90%) there was an intrusion by an unspecified member of the APT1 family, but the probability of being Auriga or Bangat was only barely likely (50-55%).

## 5.3 Seasalt and Kurton Experiments

The Seasalt and Kurton experiments were similar to the above ones and their results are presented

in Table 4 and Table 5. Compared to the other analyzed malware, Kurton has fewer forensic indicators to examine. The largest subset of indicators consists of unique strings, which have less inferential force than other types of indicators. As such, without a matching hash value (which is normal for APT1 intrusions) the highest expected probability for detection of Kurton is 80-85%.

**Table 4.** Results of the Seasalt experiment

| Intrusion By | Detected Intrusion | Duration |
|---|---|---|
| 1s: Seasalt malware used in training | Auriga (50-55%) | 382 seconds |
| | Bangat (50-55%) | |
| | Seasalt (85-90%) | |
| | APT1 (85-90%) | |
| 2s: Variant of the Seasalt malware | Auriga (75-80%) | 406 seconds |
| | Bangat (50-55%) | |
| | Seasalt (85-90%) | |
| | APT1 (85-90%) | |
| 3s: Kurton malware | Auriga (50-55%) | 344 seconds |
| | Bangat (60-65%) | |
| | Seasalt (0-50%) | |
| | APT1 (85-90%) | |

**Table 5.** Results of the Kurton experiment

| Intrusion By | Detected Intrusion | Duration |
|---|---|---|
| 1k: Kurton malware used in training | Auriga (50-55%) | 587 seconds |
| | Bangat (60-65%) | |
| | Seasalt (0-50%) | |
| | Kurton (80-85%) | |
| | APT1 (85-90%) | |
| 2k: Variant of the Kurton malware | Auriga (50-55%) | 631 seconds |
| | Bangat (60-65%) | |
| | Seasalt (0-50%) | |
| | Kurton (80-85%) | |
| | APT1 (85-90%) | |

## 5.4 Summary of the Experimental Results

Overall, CAAPT learned 40 context-independent hypotheses patterns, 2 alert rules, 2 indicator rules, 23 hypotheses analysis rules (some of them with large argument patterns that contain many context-depended hypotheses), 23 collection tasks, and 23 collection rules. 10 collection agents were also defined.

The evolutionary development of APT1 was successfully exploited by CAAPT, as shown by the experimental results. First, *after CAAPT was trained based on one instance of the Auriga malware, it was able to also detect a variant of this malware.* This was the case with all the other three malware programs considered (Bangat, Seasalt, and Kurton) and is a consequence of the learning method employed by CAAPT. Indeed, CAAPT generalizes a specific example and its explanation into a general rule that also covers similar examples which are likely to correspond to variants of the malware used in training.

Second, *CAAPT succeeded to anticipate the changes in the malware by learning patterns of IOCs in the form of hypotheses analysis rules.* If one aspect of the malware's behavior changed and became undetectable by CAAPT, it still detected the malware with some probability based on the remaining observable evidence. For example, as shown in Table 2, after being trained to detect Auriga and invoked to analyze an intrusion with Bangat, CAAPT still reported an APT1 intrusion with a probability of 85-90%, but the probability of being Auriga was lower (75-80%).

In the case of analyzing Seasalt after being trained on Auriga and Bangat, CAAPT still detected an APT1 intrusion with a probability of 85-90%, but the probability of being Auriga or Bangat was of only 50-55% (see Table 3). A similar result was obtained in the case of analyzing Kurton after being trained on Auriga, Bangat, and Seasalt. CAAPT still detected an APT1 intrusion with a probability of 85-90%, but the probability of being Auriga was 50-55%, of being Bangat was 60-65%, and of being Seasalt was 0-50% (see Table 4).

The evolutionary development of APT1 also significantly simplified and accelerated the training of the agent. For example, to train for Auriga detection, CAAPT had to learn 28 context-independent hypotheses patterns, 2 alert rules, 2 indicator rules, 13 hypotheses analysis rules, 15 collection tasks, and 15 collection rules. 8 collection agents had also to be defined.

Many of these were also applicable for the detection of Bangat intrusions. Therefore, to train for Bangat detection, a reduced number of knowledge elements needed to be learned: 1 context-independent hypothesis pattern, 1

hypotheses analysis rule, 1 collection task, and 1 collection rule. The same is true for the training to detect Seasalt and Kurton. The amount of knowledge elements that needed to be learned depended on the amount of change in the new malware. Notice, for example, the 2 alert rules and the 2 indicator rules learned for Auriga were also applicable to Bangat, Seasalt, and Kurton. Also, after defining 8 collection agents to collect evidence for Auriga detection, only two more were needed to cover the collection needs for Bangat, Seasalt, and Kurton.

While CAAPT coverage of malware detection is limited to APT1, and the increase in coverage will also increase the detection time, the times obtained in the experiment are very small and support the hypothesis that a system like CAAPT will significantly speed–up the malware detection process. The total runtime to detect an intrusion increased from around 2 minutes, when CAAPT was checking for Auriga intrusions only, to around 10 minutes when CAAPT was checking for Auriga, Bangat, Seasalt, and Kurton intrusions.

However, the run time for the generation and evaluation of the analyses only increased from around 2 seconds, when CAAPT was checking for Auriga intrusions only, to around 6 seconds when CAAPT was checking for Auriga, Bangat, Seasalt, and Kurton intrusions. As previously mentioned, most of the time is spent by waiting for the Collection and Monitoring Manager to return the results requested from the collection agents.

But time is only part of advantages offered by a system like CAAPT. While professional CSOCs have processes to be followed by analysts to ensure consistent analytical quality, it is natural for analysts to take shortcuts when they believe that the evidence examined early in the process leads to an obvious answer. These analytical leaps can shorten analysis times but can also lead to errors. CAAPT, on the other hand, will follow its learned processes fully every time. This reduces errors and provides consistent analytical results. As the number of evaluated hypotheses grows the increase of the processing time can be mitigated with additional computing power.

# 6. Discussion

Network intrusion detection is a perennial necessity because of the expected zero-day vulnerabilities of computer software. These represent the vulnerabilities that are unknown to, or unaddressed by software developers, and can therefore be exploited by hackers. This paper presented a novel approach to network intrusion detection where an instructable cognitive agent is directly taught by an expert analyst how to investigate cybersecurity alerts as the analyst would teach a student. It also presented two complementary instantiations of this approach, CAAPT and ADONIS.

CAAPT is trained to detect sophisticated APT intrusions based on combinations of weak IOCs and is efficient. However, it can only detect known malware and relies on the knowledge of malware's IOCs that are easily changed by the attackers.

ADONIS, on the other hand, focuses on adversary's behavior, independent of specific malware and tools, and can detect novel intrusions, but it is expected to be less efficient. Therefore, once a new malware is detected, its IOCs can be identified and CAAPT can be rapidly trained to detect it and its variants. The further development of and experimentation with ADONIS is a near-term goal of this research.

# 7. Conclusion

This paper presented an instructable agent approach to APT detection that promises to provide significant benefits to CSOCs by automating much of the analysts' investigative activity, increasing the probability of detecting intrusion activity and reducing the false positive detections presented to the analyst. Human analysts will be able to spend their time on more complex and more engaging analytical tasks and on teaching the agents. Therefore, the cost of operating a CSOC will be significantly reduced.

## Acknowledgements

The research reported in this paper has been supported by the Air Force Research Laboratory, as part of the Autonomous Defensive Cyberspace

Operations program, under contract number FA8750-17-C-0002, and by George Mason University. The reviewers provided detailed comments on this paper. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

## REFERENCES

Allemang, D. & Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Elsevier.

Allen, J., Chambers, N., Ferguson, G., Galescu, L., Jung, H., Swift, M. & Taysom, W. (2007). PLOW: A Collaborative Task Learning Agent. In *Proceedings of the International Conference of the Association for the Advancement of Artificial Intelligence (AAAI)* (pp. 1514-1519).

Alpaydyn, A. (2020). *Introduction to Machine Learning.* MIT Press.

Azaria, A., Krishnamurthy, J. & Mitchell, T. M. (2016). Instructable Intelligent Personal Agent. In *Proceedings of the International Conference of the Association for the Advancement of Artificial intelligence (AAAI)* (pp. 2681-2689).

Boicu, M., Tecuci, G., Marcu, D., Bowman, M., Shyr, P., Ciucu, F. & Levcovici, C. (2000). Disciple-COA: From Agent Programming to Agent Teaching. In *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 73-80).

Boicu, M., Tecuci, G., Stanescu, B., Marcu, D. & Cascaval, C. (2001). Automatic Knowledge Acquisition from Subject Matter Experts. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence* (pp. 69-78).

Cohen, L. J. (1977). *The Probable and the Provable*. Clarendon Press, Oxford.

Mandiant (2013). *APT1*. Available at <*https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf*>, last accessed: 18 June 2020.

Meckl, S. (2019). *Cybersecurity Incident Response Orchestration Using Agile Cognitive Assistants* (Doctoral dissertation, George Mason University).

Meckl, S., Tecuci, G., Marcu, D., Boicu, M. & Zaman, A. B. (2017). Collaborative Cognitive Assistants for Advanced Persistent Threat Detection. In *Proceedings of the AAAI Fall Symposium Series*, (pp.171-178).

Michalski R. S. & Tecuci G. (eds). (1994). *Machine Learning: A Multistrategy Approach*, vol. IV, Morgan Kaufmann.

Negoiţă, C. V. & Ralescu, D. A. (1975). *Applications of Fuzzy Sets to Systems Analysis*. Wiley.

Obrst, L., Chase, P. & Markeloff, R. (2012) Developing an Ontology of the Cyber Security Domain. In *Proceedings of the International Conference on Semantic Technologies for Intelligence, Defense, and Security,* George Mason University (pp.49-56).

Schum, D. A. (2001). *The Evidential Foundations of Probabilistic Reasoning*. Northwestern University Press.

Shafer, G. (1976). *A Mathematical Theory of Evidence,* Princeton University Press.

Strom, B. E., Battaglia, J. A., Kemmerer, M. S., Kupersanin, W., Miller, D. P., Wampler, C., Whitley, S. M. & Wolf, R. D. (2017). Finding Cyber Threats with ATT&CK-based Analytics. *The MITRE Corporation, Technical Report No. MTR170202*.

Tecuci, G. (1988). *Disciple: A Theory, Methodology and System for Learning Expert Knowledge*, Thèse de Docteur en Science, University of Paris-Sud.

Tecuci, G. (1993). Plausible Justification Trees: A Framework for the Deep and Dynamic Integration of Learning Strategies, *Machine Learning Journal, 11*, 237-261.

Tecuci, G. (1998). *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*. Academic Press.

Tecuci, G., Boicu, M., Boicu, C., Marcu, D., Stanescu, B. & Barbulescu, M. (2005). The Disciple–RKF Learning and Reasoning Agent, *Computational Intelligence*, *21*(4), 462-479.

Tecuci, G., Boicu, M., Bowman, M. & Marcu, D. (2001). An Innovative Application from the DARPA Knowledge Bases Program: Rapid Development of a Course of Action Critiquer, *AI Magazine*, *22*(2), 43-61.

Tecuci, G., Boicu, M., Bowman, M., Marcu, D., Shyr, P. & Cascaval, C. (2000). An Experiment in Agent Teaching by Subject Matter Experts, *International Journal of Human-Computer Studies*, *53*(4), 583-610.

Tecuci, G., Boicu, M. & Comello, J. J. (2008). *Agent-Assisted Center of Gravity Analysis*, George Mason University Press.

Tecuci, G., Boicu, M., Marcu, D., Boicu, C. & Barbulescu, M. (2008). Disciple-LTA: Learning, Tutoring and Analytic Assistance, *Journal of Intelligence Community Research and Development*. Available at: <*http://lac.gmu.edu/publications/2008/Disciple-LTA08.pdf*>, last accessed: 10 September 2020.

Tecuci, G., Boicu, M., Marcu, D., Stanescu, B., Boicu, C. & Comello, J. (2002a). Training and Using Disciple Agents: A Case Study in the Military Center of Gravity Analysis Domain, *AI Magazine*, *23*(4), 51-68.

Tecuci, G., Boicu, M., Marcu, D., Stanescu, B., Boicu, C., Comello, J., Lopez, A., Donlon, J. & Cleckner, W. (2002b). Development and Deployment of a Disciple Agent for Center of Gravity Analysis. In *Proceedings of the AAAI/IAAI* (pp. 853-861).

Tecuci, G., Kaiser, L., Marcu, D., Uttamsingh, C. & Boicu, M. (2018). Evidence-based Reasoning in Intelligence Analysis: Structured Methodology and System, *Computing in Science & Engineering*, *20*(6), 9-21.

Tecuci, G. & Keeling, H. (1999). Developing an Intelligent Educational Agent with Disciple, *International Journal of Artificial Intelligence in Education*, *10*(3-4), 221-237.

Tecuci, G., Marcu, D., Boicu, M. & Schum, D. A. (2016a). *Knowledge Engineering: Building Cognitive Assistants for Evidence-Based Reasoning*. Cambridge University Press.

Tecuci, G., Marcu, D., Boicu, M., Schum, D. & Russell, K. (2011). Computational Theory and Cognitive Assistant for Intelligence Analysis. In *Proceedings of the Sixth International Conference on Semantic Technologies for Intelligence, Defense, and Security – STIDS* (pp. 68-75).

Tecuci, G., Meckl, S., Marcu, D. & Boicu, M. (2019). Instructable Cognitive Agents for Autonomous Evidence-Based Reasoning. *Advances in Cognitive Systems*, *8*, 73-92.

Tecuci, G., Schum, D. A., Marcu, D. & Boicu, M. (2016b). *Intelligence Analysis as Discovery of Evidence, Hypotheses, and Arguments: Connecting the Dots*. Cambridge University Press.

Wigmore, J. H. (1913). The Problem of Proof. *Illinois Law Review*, *8*, 77–103.

W3C (2014). Available at: <*http://www.w3.org/TR/rdf-schema/*>, last accessed: 18 June 2020.

Zadeh, L. A. (1983). The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems, *Fuzzy Sets and Systems*, *11*(1-3), 199-227.

Zeek (2020). *Zeek: An Open Source Network Security Monitoring Tool*. Available at: <*www.zeek.org*>, last accessed: 8 September 2020.