An Autonomous Resiliency Toolkit for Cyber Defense Platforms

Fusun Yaman[1], Thomas Eskridge[2], Aaron Adler[1], Michael Atighetchi[1], Borislava I. Simidchieva[1], Sarah Jeter[1], Jennifer Cassetti[3] and Jeffrey DeMatteis[3]
[1]*Raytheon BBN Technologies, Cambridge, MA, U.S.A.*
[2]*Florida Institute of Technology, Melbourne, FL, U.S.A.*
[3]*Air Force Research Lab, Rome, NY, U.S.A.*

Keywords: Grammar Inference, Workflow Learning, Automation, Autonomous Cyber Resiliency, Ontology, Cyber Protection, User Study.

Abstract: Cyber defenders need automation tools that are intuitive, trustworthy, non-intrusive, and reusable. The Behavior-extracting Autonomous Resiliency Toolkit (BART) is such a tool. Its architecture combines existing results and AI techniques including workflow learning, mutli-agent frameworks, knowledge representation, and inference. A user study demonstrates that BART significantly shortens the required time to execute a cyber defense. The study also revealed three types of errors that an automated tool such as BART could prevent: typographical/syntax, procedural, and "hidden." We also describe an emerging application of BART.

1 INTRODUCTION

The odds in cyber defense are inherently stacked against the defender. Adversaries only need to find one vulnerability to exploit a target system, while defenders need to close all vulnerabilities, including currently unknown ones. A significant increase in the amount of automation support may be required in order to make cyber defenders more effective. Yet, current tactics, techniques, and procedures show a significant imbalance in the amount and level of automation between adversaries and defenders.

Cyber attackers frequently automate much of their work through management platforms that enable rapid sharing and reuse of malicious code. Furthermore, malware has evolved to the point where botnets and viruses make autonomous decisions, e.g., to remain dormant if they detect monitoring or to intertwine attacks with regular user activities to stay within the variance of observable parameters.

DISTRIBUTION A. Approved for public release: distribution unlimited. Case Number 88ABW-2018- 0228 20180119 Work sponsored by AFRL under contract FA8750-16-C-0053; the views and conclusions contained in this document are those of the authors and not AFRL or the U.S. Government.

In contrast, many of the activities executed by cyber defenders are manual in nature, limited in scope due to staffing constraints, and slow in detection and response times. This is not only true for the sophisticated tasks of spotting anomalies in large data sets, but also for mundane tasks such as installing software patches and provisioning accounts. This lack of automation for cyber defensive operations puts many of today's computer systems at significant risk of being compromised. Furthermore, based on our interactions with cyber defenders in high-stakes industries (such as electric power transmission system operators and military) useful automation needs to be (1) intuitive, (2) trustworthy, (3) non-intrusive, and (4) reusable. To be intuitive, the system's representations and methods must align with operator's problem-solving approaches and allow the operator to inspect the justification of the system's decisions. To be trustworthy, the software must work as expected consistently and flag problems at every stage if it fails, enabling tracking by the user. To be non-intrusive, the system must not require time investment from the operator to bootstrap (e.g., to build extensive models), instead ideally it would work with existing tools and would not have a steep learning curve. To be effective, the system ought to match or improve performance, such as the time to complete a task and/or quality of task resolution. To be reusable, the system should at minimum

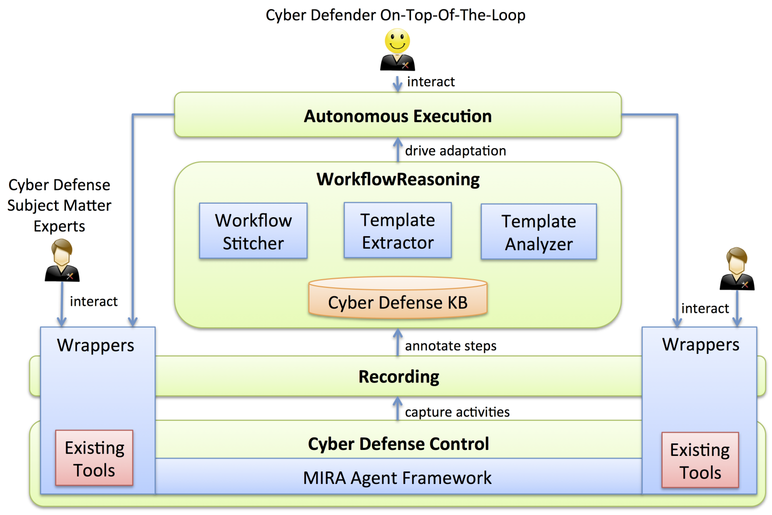


Figure 1: BART functional overview.

work for a specific context, but ideally work in different contexts or allow customization for dynamic environments.

The Behavior-extracting Autonomous Resiliency Toolkit (BART) (Atighetchi et al., 2016) automatically learns workflows by observing operators, extracts fragments that can be reused and customized for different contexts, and autonomously executes learned workflows (Figure 1). BART increases the effectiveness and efficiency of cyber defense through increased speed, equal or higher accuracy, significantly lower cost, and increased coverage. We claim that BART has all four characteristics of a useful cyber defense automation tool. We conducted a user study to measure the effectiveness of BART, which demonstrated a significant reduction in execution time and errors compared to manual execution of tasks.

The rest of the paper describes BART's underlying technologies, the experiment design and results of the user study, related work, and finally conclusions and future work.

2 BART ARCHITECTURE

BART (Figure 1) is composed of multiple integrated components that together provide workflow learning, multi-agent frameworks, and knowledge representation. BART components communicate through the Cyber Defense Knowledge Base (CDKB) using the Learnable Task Modeling Language (LTML) (Burstein et al., 2009a). CDKB, a semantic web triple store developed by us to contain both the workflow building blocks and representations, as well as the cyber defense domain ontology, serves as a shared knowledge library.

The BART component interlingua, LTML, combines features of the Web Ontology Language (OWL) (van Harmelen and McGuinness, 2004), OWL for Services (OWL-S) (Martin et al., 2004), and the Planning Domain Definition Language (PDDL) (Fox and Long, 2003), using a clear, compact syntax to create human-readable representations of web-service procedures and hierarchical task models. LTML was designed to support tools that could learn procedures by demonstration (see Section 4).

2.1 Cyber Defense

Cyber Defense Tools

BART currently supports recording and executing workflows associated with Metasponse, PowerShell, and other tools. *Metasponse* is an incident-response framework developed by AIS Inc., and used by the Government (AIS Inc, 2019). Metasponse manages a set of computers on a network by collecting information and remotely effecting actuators. *PowerShell* is a task-based command-line shell and scripting language specifically designed for Microsoft Windows system administrators and power-users. Lastly, BART can support any tool that provides a *web-service interface*, e.g., the ArcSight Security Information and Event Management (SIEM) tool (ArcSight,).

Cyber Defense Control

In order to monitor and manage existing tools like Metasponse, BART deploys software wrappers around the tools. Using these wrappers, BART can transparently intercept interactions and pass on

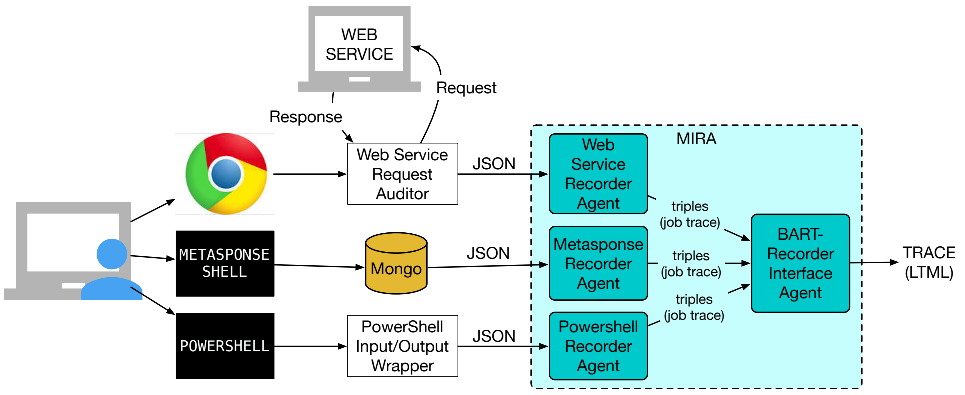

WEB SERVICE
Request
Response
Web Service Request Auditor
JSON
MIRA
Web Service Recorder Agent
triples (job trace)
METASPONSE SHELL
Mongo
JSON
Metasponse Recorder Agent
triples (job trace)
BART-Recorder Interface Agent
TRACE (LTML)
triples (job trace)
POWERSHELL
PowerShell Input/Output Wrapper
JSON
Powershell Recorder Agent

Figure 2: BART records operator behavior from multiple sources.

commands to be executed. BART's Cyber Defense Control component builds on the Mission-aware Infrastructure for Resilient Agents (MIRA) (Carvalho et al., 2015) distributed agent framework to integrate wrapped tools.

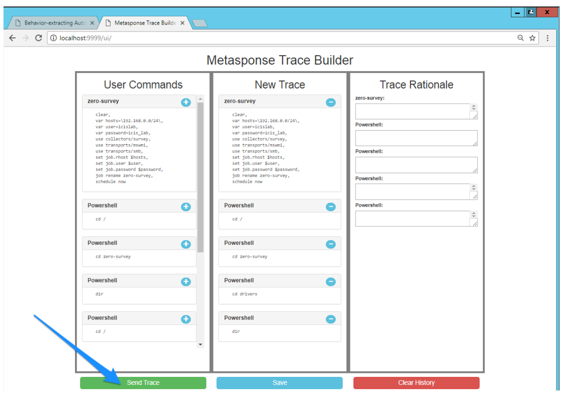
MIRA integration requires the wrappers to provide an agent interface, a capabilities description, and a sensor/actuator interface. The agent interface provides a common API that connects the wrapper to the framework. This enables the specification of methods to monitor and control the operational aspects of the tool. The capabilities description is an ontology (in OWL) describing the capabilities provided by the wrapper. This enables MIRA to determine appropriate actions based on a semantic understanding of sensor or actuator capabilities. Finally, the sensor/actuator interface code links the semantic descriptions with the executable code (e.g., written in Java or any other language) that effects changes in the tool's configuration.

2.2 Trace Recording

As cyber defenders use Metasponse and other tools to complete their assigned tasks, BART records their activities to produce a *trace* (in LTML), i.e., a sequence of observed steps. BART wrappers collect the operator input and program output for each tool and a MIRA agent puts the information into a standardized form. The BART Recorder Interface Agent allows access to the previously observed operations in a standardized way, facilitating the addition of new tools. A overview of the integration is shown in Figure 2.

The recording agents do more than capturing mouse clicks or text typed into a terminal. The agents map such low-level events to a higher level of abstraction, providing useful information for workflow learning. For example, instead of the mouse click location or that a "TCPDump" button was pressed, the recorder indicates that a NetworkDataCollection event was triggered. The NetworkDataCollection concept and its connection to the "TCPDump" button is represented in an ontology. A set of high-level activity definitions in LTML is shared between BART learning and recording components to establish a common understanding.

BART employs two modes for contextual trace generation, batch and manual mode. Batch mode is triggered by special user-generated events in a tool explicitly signaling the beginning and end of recording. This leads to a non-intrusive yet constrained trace generation process. To achieve good learning accuracy in this mode, all captured activity must be relevant and error free. Otherwise, the learned workflow may contain irrelevant or redundant activities. Manual Mode involves displaying all captured activity to the user and letting the user pick and reorder events that will form the trace through a Graphical User Interface (GUI) (Figure 3). This mode requires operator's time and attention but leads to traces with fewer errors.



User Commands:

zero-survey
```
clear,
var hosts=\192.168.0.0/24\,
var user=icislab,
var password=icis_lab,
use collectors/survey,
use transports/mswmi,
use transports/smb,
set job.rhost $hosts,
set job.user $user,
set job.password $password,
job rename zero-survey,
schedule now
```
Powershell
```
cd /
```
Powershell
```
cd zero-survey
```
Powershell
```
dir
```
Powershell
```
cd /
```

New Trace:

zero-survey
```
clear,
var hosts=\192.168.0.0/24\,
var user=icislab,
var password=icis_lab,
use collectors/survey,
use transports/mswmi,
use transports/smb,
set job.rhost $hosts,
set job.user $user,
set job.password $password,
job rename zero-survey,
schedule now
```
Powershell
```
cd /
```
Powershell
```
cd zero-survey
```
Powershell
```
cd drivers
```
Powershell
```
dir
```

Trace Rationale:

zero-survey:

Powershell:

Powershell:

Powershell:

Powershell:

Figure 3: GUI support for activity trace generation.

2.3 Workflow Reasoning

The BART workflow reasoning component uses the Workflow Inference from Traces (WIT) algorithm (Yaman et al., 2009) in the Workflow Stitcher subcomponent to learn a workflow given a set of observed traces (Figure 4). WIT uses grammar inferencing (GI) techniques (specifically model merging) to transform the input sequences into a more general graph containing branches and loops. The learned workflow can generate (in the sense that a grammar is generative) both the demonstrated and unseen sequences. A strength of the GI approach is that it can generalize both from a single trace and multiple traces, enabling learning and adaptation over time. As new traces are recorded, BART can incorporate them naturally and create deviations in already existing workflows. Workflow learning is a powerful technique that enables the system to be intuitive, non-intrusive, and effective. The WIT algorithm has four steps (Figure 5); data dependency analysis, context inference, step generalization and decomposition. The context inference step identifies similar actions in the trace by looking at the type of the action and the types of the data dependencies. This step essentially discovers which of the actions in the trace correspond to the same generalized node in the workflow, e.g., two lookup actions will be different if they get inputs from different survey activities. Similarity detection is the mechanism that allows WIT to generalize from

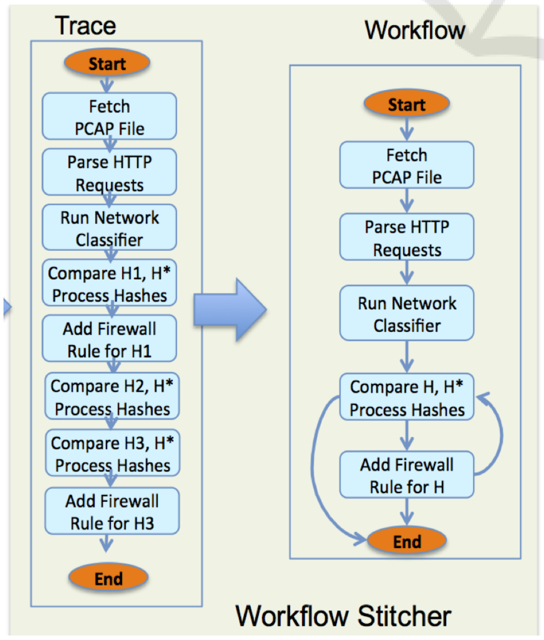


Figure 4: Workflow Stitcher generates workflows from traces.

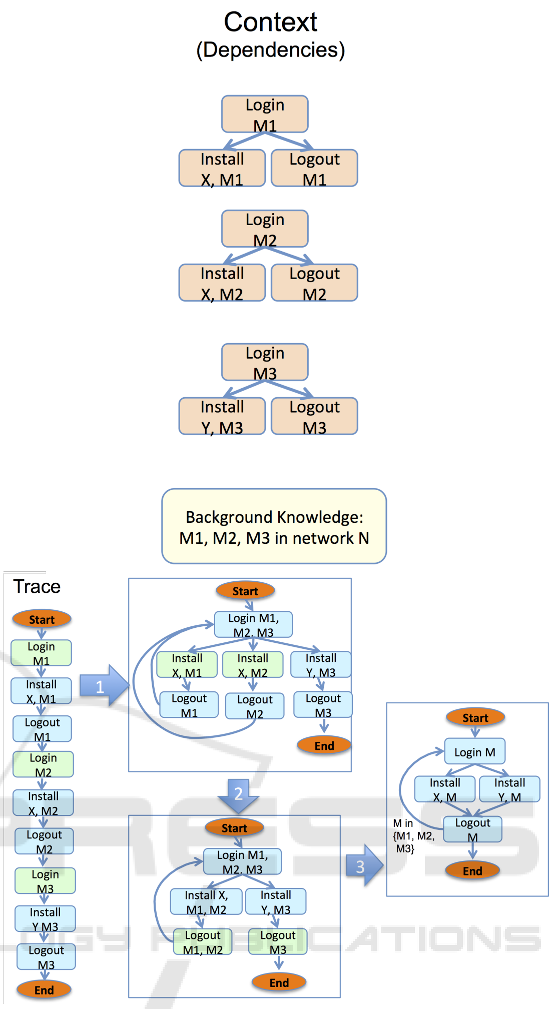


Figure 5: After the data dependency analysis (top), WIT performs context inference and generalization loops (bottom).

a single trace containing multiple executions of the same task. Step generalization starts with a very specific workflow that can only generate the input traces and then iteratively merges the nodes that are identified as similar in the context inference step. It also merges nodes (representing the same type of actions) that satisfy certain proximity constraints (e.g., nodes that have the same immediate predecessor). The second type of merging is inspired by GI algorithms that learn a class of regular grammars using only positive examples from the language. It is possible to infer more similarities at the end of step generalization so context inference and step generalization steps are interleaved until the set of similarities and the learned graph converges.

Finally the decomposition step recursively identifies loops and branches in the learned graph and replaces them with a single node representing a composite action. At the end of this step we end up with a hierarchy of parametric workflows. The decomposition algorithm is based on structural graph analysis and domain independent heuristics specifically for the purpose of identifying loops. The algorithm can decompose the workflow correctly if the workflow does not have any nested loops or the nested loops have single entry and exit points, i.e., each loop iteration starts with a unique activity and ends with another unique activity.

Workflow Execution

BART is able to execute a sequence of operator actions (specified in the learned workflow) in multiple environments in an intuitive, non-intrusive, effective, and reusable manner (see Figure 6). Workflow execution interleaves steps of workflow interpretation and interactions with wrapped actuators. The communication with the actuators is done with LTML which provides a communication standard from the execution engine to the wrapper; the wrapper converts LTML to actuator-specific commands. The BART Execution Engine is responsible for interpreting the higher level logic (such as sequences, branches and loops), handling the data flow (such as passing the output of an activity as an input to another one) and gathering the inputs from the user through the GUI.

The actuation system structure is the reverse of the recording system structure. It starts with HTTP operation requests received by the BART Replayer Interface Agent from the Execution Engine. This agent determines the type of execution needed for the operation and dispatches the request to an operation-specific replayer agent (i.e., Metasponse, Powershell, and Web Service as shown in Figure 6).

2.4 Graphical User Interface

BART's GUI allows reusing and auditing of workflows through visualized traces and workflows, triggering execution, and tracking execution status. The main screen (Figure 7) displays three panels: the trace on the left, the workflow BART learned in the middle, and the execution instance on the right. In order

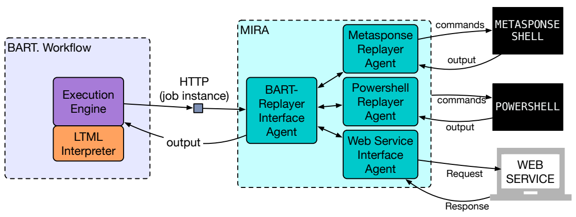
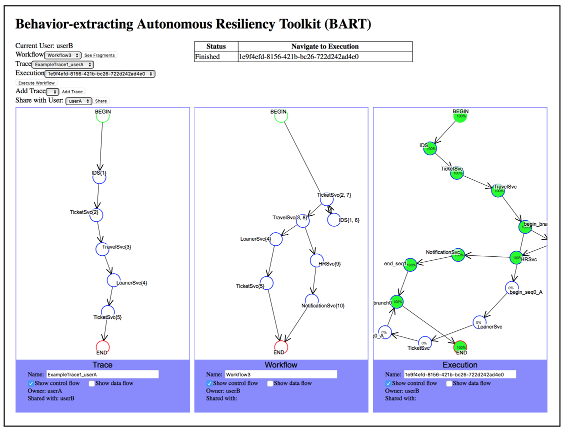


Figure 6: BART can replay actions in several environments.



Behavior-extracting Autonomous Resiliency Toolkit (BART)

Current User: userB
Workflow: Workflow3
Trace: ExampleTrace1_userA
Execution: 1e9f4efd-8156-421b-bc26-722d242ad4e0

Status	Navigate to Execution
Finished	1e9f4efd-8156-421b-bc26-722d242ad4e0

Trace
Name: ExampleTrace1_userA
✓ Show control flow ☐ Show data flow
Owner: userA
Shared with: userB

Workflow
Name: Workflow3
✓ Show control flow ☐ Show data flow
Owner: userB
Shared with:

Execution
Name: 1e9f4efd-8156-421b-bc26-722d242ad4e0
✓ Show control flow ☐ Show data flow
Owner: userB
Shared with:

Figure 7: Displaying a trace and workflow data dependencies.

to build user trust in the system, the GUI contains intuitive views with useful, non-intrusive information. The GUI provides visibility into the data- and control-flow links, and into the links between a workflow, its traces, and its executions.

At the top right of the screen, a table provides visibility into workflow execution status: "Running," "Waiting," "Finished," and "Error." When user input is needed during execution, the status in the table will change to "Waiting," alerting the user to needed information in a non-intrusive manner. User input is provided via an overlay pane on top of the execution panel. The user is prompted for a data value that can be type checked (e.g., making sure a properly formatted IP address is provided). Once the user provides the information, execution continues. Feedback on execution progress is provided by showing progress for steps, branches, and loops using colors and showing percentage completion estimates.

3 USER STUDY

We conducted a user study to collect data demonstrating how the workflows created by BART affect the performance of operators conducting typical "cyber hunting" missions on multiple networks. Participants were divided into two conditions. In the first condition, the participants used the Metasponse and PowerShell tools provided by AIS, Inc. to perform a series of tasks looking for tainted processes on three different computer networks. In the second condition, participants were also trained to use BART to construct workflows that allowed for retargeting and replay. Upon completion of the training, the participants executed the test problem, which involved running a hunting operation for unknown pro-

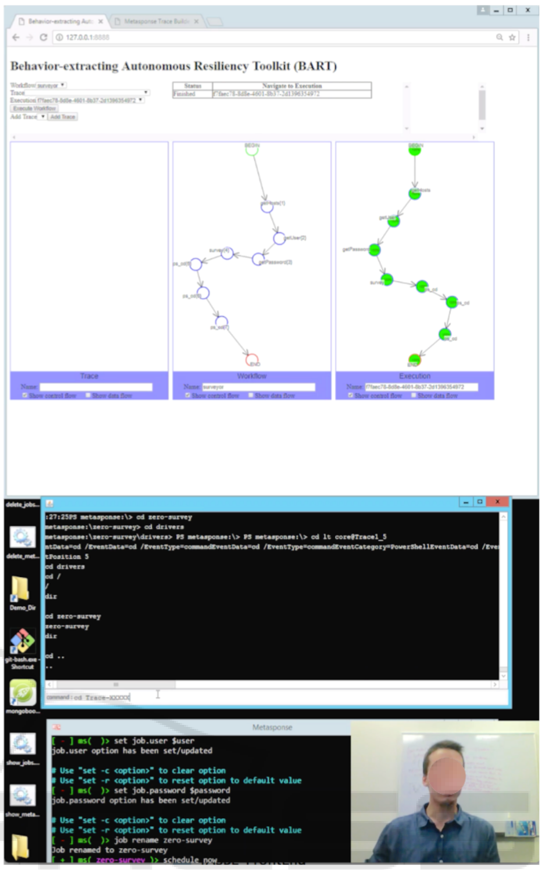
Tool	Commands	Explanation
M	var hosts="10.0.0.0/23"	Identify the target subnet , e.g., 10.0.0.0/23.
M	use collectors/autoruns use transports/mswmi use transports/smb set job.rhost=$hosts job rename autorun-1 schedule now job status autorun-1 job pickup autorun-1	Create a Metasponse job for collecting information on the list of processes start running automatically at start up. Schedule it to run immediately Check for status and save the output of the collection.
M	use collectors/hashproc use transports/mswmi use transports/smb set job.rhost=$hosts job rename hashproc-1 schedule now job status hashproc-1 job pickup hashproc-1	Create a Metasponse job for collecting information on the list of processes that are currently running. Schedule it to run immediately Check for status and save the output of the collection.
P	cd /	Make sure you are in the top directory
P	cd autorun-1	Go into the results of autorun-1 job.
P	$enabled = dir | ? { $_.IsEnabled }	Filter the processes that are enabled.
P	$unknown = $enabled | ? { $_.MD5 } | Get-Artifact -Unknown	Find the enabled processes that are not recognized.
P	cd ..	Get out of autorun-1 job.
P	cd hashproc-1	Go into the results of hashproc-1 job.
P	$unknownHashproc = $unknown | % { $autorun = $_ ; dir -HostIP $autorun.HostIP | ? { $_.FilePath -eq $autorun.ImagePath}}	Find the hash of all unknown processes that auto-started.
P	$knownBad = $unknownHashProc | ? {-Not $_.IsSigned} | ? {-Not $_.IsSignedTrusted}	Find the hash of all the unknowns that is not signed or signed by not trusting entity.
P	$knownBad | Add-Artifact -Blacklist	Add untrusted process hashes into the black list.

Figure 8: Steps for finding suspicious software on a target subnet and adding them into a black list. M - Metasponse, P- Powershell

cesses, provided as part of the test materials. Providing the steps to be executed for the test tasks successfully de-emphasized the skill level of the participants. The study measured the time needed for training, for workflow learning and execution, and for overall execution time. Our hypothesis was that participants using BART+Metasponse would show a significant decrease in total task time.

Figure 8 shows the general steps required to find the unknown processes. This operation requires executing several Metasponse jobs and processing their output in PowerShell to determine whether or not the process should be blacklisted. This completed the task for the selected network. The participant then repeated the hunting process on the two remaining networks. In the second condition, after completing the hunting task on the first network, participants provided the trace to BART so it could learn the hunting workflow. Then the second and third networks could be investigated by replaying the learned workflow in BART.

Figure 9 shows the experimental setup for a participant in the second condition. The top of the figure shows the BART GUI as described in Section 2.4. Below that are command windows for PowerShell and Metasponse, and a video recording of the participant during the experiment. The entire interface was recorded and used for later coding according to the timing and mistake metrics defined in the study.



```
:27:25PS metasponse:\> cd zero-survey
metasponse:\zero-survey> cd drivers
metasponse:\zero-survey\drivers> PS metasponse:\> PS metasponse:\> cd lt core@Trace1_5
ntData=cd /EventData=cd /EventType=commandEventData=cd /EventType=commandEventCategory=PowerShellEventData=cd /Eve
tPosition 5
cd drivers
cd /
/
dir

cd zero-survey
zero-survey
dir

cd ..
..
```

```
command:cd Trace-XXXXX
```

```
[ - ] ms(  )> set job.user $user
job.user option has been set/updated

# Use "set -c <option>" to clear option
# Use "set -r <option>" to reset option to default value
[ - ] ms(  )> set job.password $password
job.password option has been set/updated

# Use "set -c <option>" to clear option
# Use "set -r <option>" to reset option to default value
[ - ] ms(  )> job rename zero-survey
Job renamed to zero-survey
[ + ] ms( zero-survey )> schedule now
```

Figure 9: User study experimental setup, showing user video, PowerShell, Metasponse, and the BART workflow created.

3.1 Results

The participants in the study were 26 students from the Florida Institute of Technology with an average age of 26. The students came from a variety of backgrounds beyond computer science, including mechanical engineering and meteorology. Participants were randomly placed into the Metasponse only condition ($n = 12$), where they used only the Metasponse and PowerShell tools to perform a set of three tasks, or in the Metasponse+BART condition ($n = 14$), where the Metasponse and PowerShell tools were augmented with BART. Table 1 shows results for these participants. Both conditions require training to use the Metaponse and PowerShell tools, but the BART condition includes additional training in using the BART interfaces for trace creation and workflow building and execution.

The time and effort requirements for Task 1 are identical between the two experimental conditions, and therefore we expect no significant differences in the time to complete Task 1 (one-sided F-test, P-value

0.4848, not significant at $p < 0.01$). For later tasks in both conditions, we expect there to be a significant time savings in executing the needed commands. In the Metasponse only condition, operators can reuse previously created jobs and cycle through previously typed commands to re-execute them. In the Metasponse+BART condition, the reuse of a workflow requires only inputing the parameters needed to instantiate a workflow for execution, which averaged just 32 seconds during the experiment (the balance of time in Tasks 2 and 3 is the execution time of the workflow using the provided inputs). With increased experience using the convenient functionality of the BART interface, we expect that time to dramatically decrease to very close to the minimum possible runtime for the workflow. The results showed the anticipated speedup for Metasponse+BART participants in Tasks 2 and 3. The observed time for Task 3 is just slightly over the minimum execution time for the workflow of 2.31 minutes[1]. While there was no significant difference in the times between conditions for Task 1, the differences in times between conditions for Tasks 2 and 3 were significant (one-sided F-test, Task 2 $P < 0.00001$, Task 3 $P < 0.00001$, significant at $p < 0.01$).

Table 1: Timing results from the user study showing mean and standard deviation (in parentheses) in minutes for training, testing, and learning the workflow in BART.

Activity	Type	Metasponse	Metasponse+BART
Training	Metasponse	11.59 (8.22)	12.64 (5.16)
Training	BART	N/A	20.94 (9.26)
Test	Task 1	15.85 (4.18)	15.92 (5.97)
Test	Task 2	9.24 (4.33)	3.04 (1.31)
Test	Task 3	6.36 (1.32)	2.47 (0.51)
Learning	Trace	N/A	2.30 (2.13)
Learning	Workflow	N/A	1.23 (0.60)

3.2 Discussion

While a principle benefit of BART is the increased efficiency in executing tasks, another benefit identified by this study is the reduction of operator mistakes, which in turn contributes to a reduction in the time to complete tasks. During this study, we were interested to find that the errors we observed users making were easily classified into one of three common categories, namely: (1) syntax errors and typos, (2) procedural errors, and (3) errors we identify as "hidden." Typographical and syntax errors are simple to correct once identified, but there is still a time penalty for going back to correct a misspelling or looking up the correct syntax. Procedural errors can be drastically more time-consuming than typographical or syntax errors because they may not be identified until the the participant investigated the task's results and determined that an incorrect set of jobs was executed to complete the task. "Hidden" errors are commands that are almost, but not quite, correct. For example, a variable misspelling from "$knownBad" to "$knowBad" will be executed by PowerShell without any complaint, but this will result in a *null* value being used for computation, which can be time-consuming to discover, investigate, and debug. Since BART can "replay" a workflow it has seen once and learned, all of these errors are much less likely to occur in the Metasponse+BART experiment condition.

Lastly, as is clear from Table 1, the use of BART greatly speeds up the execution of workflows after the first time, but it also provides a helpful guide in reminding the user what the next step ought to be (or giving them a choice if there are multiple options), which helps the user to remember and retain the workflow better without having to look up the corresponding Metasponse history or the scenario documentation they were following. In fact, workflow replay and execution show great promise as a training aide for learning new cyber protection tactics, techniques, and procedures. We can envision replaying certain learned workflows in BART that are considered fundamental as a tutorial or practice course for new cyber protection operators. Sharing and tagging capabilities in BART also enable the rapid sharing of newly developed workflows between different cyber protection teams. Thus, a corpus of relevant workflows could be compiled into different syllabi for the explicit purpose of training.

4 RELATED WORK

The concept underlying BART relates to a number of prior and ongoing research efforts in machine learning, workflow modeling, and cyber security decision making.

4.1 Workflow Learning

Learning of workflows from observable behavior has been an active topic in machine learning. Workflow mining (Van der Aalst et al., 2004; Herbst, 2000) describes the concept of assembling workflows from log data about their execution. Other learning algorithms ranging from grammar induction (Cook

[1]Workflow execution proceeded serially in the BART implementation used for the user study. This minimum time could be further reduced with workflow optimization for parallel execution, but that would not realistically reflect how a human operator performs the workflow.

Table 2: Comparison of candidate languages for CDKB
✓= yes ✗= no – = somewhat.

Candidate	Expressivity	Verbosity	Standards	Maintained	Tools
OWL-S	✓	✗	✓	✗	–
PDDL	✗	✓	✓	✓	✓
LTML	✓	✓	–	–	✓
Little-JIL	–	✓	✗	–	✓
BPEL	–	–	✓	✓	✓
BPMN	–	–	✓	✓	✓
TAEMS	✓	✓	✗	✗	✗

and Wolf, 1995) to bayesian model merging (Herbst and Karagiannis, 1998) are also employed in learning action patterns. Our previous work on Plan Order Induction by Reasoning from One Trial (Burstein et al., 2009b) research project and associated semantics of the Learnable Task Modeling Language (LTML) (Burstein et al., 2009a) provide a foundational framework for learning workflows from single observations of traces. Specifically two complementary one-shot workflow learning algorithms emerged from this effort: WIT (Yaman et al., 2009) and ReCycle (Haigh and Yaman, 2011). While these efforts focus on creating a generic workflow learning capability with minimum number of examples, BART aims to build a specific workflow learning capability that is custom tailored to execution of cyber defense workflows. These capabilities necessitated many extensions to the original framework to support workflow execution, operational integration (learning from live observations), knowledge representation for the cyber defense domain, and sharing and modifying workflows, among others.

4.2 Knowledge Representation and Workflow Definition

There are a number of modeling frameworks used to model workflows (Martin et al., 2004; Fox and Long, 2003; Burstein et al., 2009a; Cass et al., 2000; Andrews et al., 2003; White, 2009; Decker, 1996). We considered these candidates for representing the knowledge in CDKB from the perspective of expressivity (ability to express complex workflows), verbosity (human readability), tools (supporting tools such as parsers or reasoners), community (an active developer community), and standards (based on a standard) before deciding on LTML. Table 2 summarizes our assessment.

An important note is that selecting LTML accomplished two goals: having a workflow definition language with a rigorous enough semantics to support learning and execution, but also having a direct and easy connection to an extensible cyber ontology knowledge base represented in the OWL. LTML is a surface language, which encompasses the features of OWL and OWL-S, making integration with the cyber knowledge base seamless.

4.3 Autonomous Cyber Operations

Related work in automating cyber defense includes use of cognitive reasoning to select defensive actions (Benjamin et al., 2008) and the use of strategies and tactics to implement an adaptive defense (Atighetchi et al., 2004). Our work provides an end-to-end integrated system that learns from observed behavior only. Finally, work on analyzing specific cyber defense workflows (D'Amico and Whitley, 2008) provides context for learning workflows in the cyber defensive operations domain. Such results complement our assessment of useful cyber defense automation characteristics.

In penetration testing circles, automation frameworks such as Metasploit (Maynor, 2011; Kennedy et al., 2011) are a popular time-saving aide and thus widely used; unfortunately, Metasploit has also become widely used by cyber attackers to script together ever more sophisticated attacks out of pieces of malicious code.

Orchestrators and intrusion-detection systems can also perform simple automated incident response tasks, such as automatically shutting down certain ports when an attempted attack is detected. Such systems can be used as building blocks for BART to learn, and later replay, as sophisticated cyber hunting and protection workflows. Therefore a side-by-side comparison of such tools with BART is impractical because of divergent goals and scope.

5 CONCLUSION

BART leverages workflow learning, multi-agent frameworks, knowledge representation, and inference to automatically learn, replay, and adjust workflows for cyber defense. This is accomplished in a manner that is intuitive, trustworthy, non-intrusive, effective, and reusable. The user study demonstrated that BART significantly shortens execution cycles and indicated multiple types of errors that BART can help prevent. While BART workflows may help to prevent cyber crime by providing timely and effective incident response, it is important to note that the goal of the BART framework is to automate some of the more straightforward responses in order to free the human operators to perform tasks that require new skills, new workflows, or creative combinations or adaptations of existing workflows. Therefore, evaluating BART's standalone effectiveness at cyber defense was out of scope for the experiments and user study presented here and would be an interesting avenue of future pursuit.

Since conducting this user study, we have enhanced BART's workflow optimization capabilities to leverage concurrent execution when possible, which further increases speed. We plan to conduct more extensive experimentation to validate the optimizations. The focus of our future work is to transition BART into active use by cyber defenders (both military and civilian).

ACKNOWLEDGMENTS

Many thanks to AIS, Inc. for their technical support and for providing the Metasponse and Powershell tools used in the BART user study.

REFERENCES

AIS Inc (2019). Metasponse user's guide.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., et al. (2003). Business process execution language for web services.

ArcSight. Enterprise Security Manager.

Atighetchi, M., Pal, P., Webber, F., Schantz, R., Jones, C., and Loyall, J. (2004). Adaptive cyberdefense for survival and intrusion tolerance. *IEEE Internet Computing*, 8(6):25–33.

Atighetchi, M., Yaman, F., Simidchieva, B., and Carvalho, M. (2016). An autonomous resiliency toolkit - needs, challenges, and concepts for next generation cyber defense platforms. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 1–6.

Benjamin, D. P., Pal, P., Webber, F., Rubel, P., and Atigetchi, M. (2008). Using a cognitive architecture to automate cyberdefense reasoning. In *Bio-inspired Learning and Intelligent Systems for Security, 2008. BLISS'08. ECSIS Symposium on*, pages 58–63. IEEE.

Burstein, M., Goldman, R. P., McDermott, D. V., McDonald, D., Beal, J., and Maraist, J. (2009a). LTML—a language for representing semantic web service workflow procedures. In *Proceedings ISWC workshop on Semantics for the Rest of Us*.

Burstein, M. H., Yaman, F., Laddaga, R. M., and Bobrow, R. J. (2009b). POIROT: Acquiring workflows by combining models learned from interpreted traces. In *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, pages 129–136, New York, NY, USA. ACM.

Carvalho, M., Eskridge, T. C., Ferguson-Walter, K., and Paltzer, N. (2015). MIRA: a support infrastructure for cyber command and control operations. In *Resilience Week (RWS), 2015*, pages 1–6. IEEE.

Cass, A. G., Staudt Lerner, B., McCall, E. K., Osterweil, L. J., Sutton Jr, S. M., and Wise, A. (2000). Little-JIL/Juliette: A process definition language and interpreter. In *ICSE '00: Proc. 22nd Int. Conf. Softw. Eng.*, pages 754–757.

Cook, J. E. and Wolf, A. L. (1995). Automating process discovery through event-data analysis. In *Proc. of ICSE '95*, pages 73–82, New York, NY, USA.

D'Amico, A. and Whitley, K. (2008). The real work of computer network defense analysts. In *VizSEC 2007*, pages 19–37. Springer.

Decker, K. (1996). Taems: A framework for environment centered analysis & design of coordination mechanisms. *Foundations of distributed artificial intelligence*, pages 429–448.

Fox, M. and Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124.

Haigh, K. Z. and Yaman, F. (2011). RECYCLE: Learning looping workflows from annotated traces. *ACM Trans. Intell. Syst. Technol.*, 2(4):42:1–42:32.

Herbst, J. (2000). A machine learning approach to workflow management. In *ECML*, volume 1810, pages 183–194. Springer.

Herbst, J. and Karagiannis, D. (1998). Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. In *Proc. of DEXA 98*, Washington, DC, USA. IEEE Computer Society.

Kennedy, D., O'Gorman, J., Kearns, D., and Aharoni, M. (2011). *Metasploit: the penetration tester's guide*. No Starch Press.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004). OWL-S: Semantic markup for web services. W3C Member Submission.

Maynor, D. (2011). *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Elsevier.

Van der Aalst, W., Weijters, T., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.

van Harmelen, F. and McGuinness, D. L. (2004). OWL web ontology language overview. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

White, S. A. (2009). Business process modeling notation (BPMN). Technical Report formal/2009-01-03, Business Process Management Initiative (BPMI).

Yaman, F., Oates, T., and Burstein, M. (2009). A context driven approach to workflow mining. In *Proceedings of IJCAI-09*.