Fall 2015

Cyber security research frameworks for coevolutionary network defense

George Daniel Rush

Department:

CYBER SECURITY RESEARCH FRAMEWORKS FOR COEVOLUTIONARY NETWORK DEFENSE

by

GEORGE DANIEL RUSH

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER SCIENCE

2015

Approved by

Dr. Daniel Tauritz, Advisor
Dr. Sriram Chellappan
Dr. Alexander D. Kent

PUBLICATION THESIS OPTION

This thesis has been prepared in the form of three papers formatted to Missouri S&T specifications.

Paper 1. Pages 3–19 have been published as *DCAFE: A Distributed Cyber Security Automation Framework for Experiments* in the IEEE Thirty-eighth Annual International Computers, Software and Applications Conference Workshops (2014) with Daniel R. Tauritz and Alexander D. Kent.

Paper 2. Pages 20–43 have been published as *Coevolutionary Agent-based Network Defense Lightweight Event System (CANDLES)* in the Seventeenth Annual Conference Companion on Genetic and Evolutionary Computation (2015) with Daniel R. Tauritz and Alexander D. Kent.

Paper 3. Pages 44–78 are intended for submission to the journal Computers & Security as *CANDLES v2: A Framework for Coevolving Attacker and Defender Strategies for Enterprise Computer Networks* with Daniel R. Tauritz and Alexander D. Kent.

ABSTRACT

Cyber security is increasingly a challenge for organizations everywhere. Defense systems that require less expert knowledge and can adapt quickly to threats are strongly needed to combat the rise of cyber attacks. Computational intelligence techniques can be used to rapidly explore potential solutions while searching in a way that is unaffected by human bias.

Several architectures have been created for developing and testing systems used in network security, but most are meant to provide a platform for running cyber security experiments as opposed to automating experiment processes. In the first paper, we propose a framework termed Distributed Cyber Security Automation Framework for Experiments (DCAFE) that enables experiment automation and control in a distributed environment.

Predictive analysis of adversaries is another thorny issue in cyber security. Game theory can be used to mathematically analyze adversary models, but its scalability limitations restrict its use. Computational game theory allows us to scale classical game theory to larger, more complex systems. In the second paper, we propose a framework termed Coevolutionary Agent-based Network Defense Lightweight Event System (CANDLES) that can coevolve attacker and defender agent strategies and capabilities and evaluate potential solutions with a custom network defense simulation.

The third paper is a continuation of the CANDLES project in which we rewrote key parts of the framework. Attackers and defenders have been redesigned to evolve pure strategy, and a new network security simulation is devised which specifies network architecture and adds a temporal aspect. We also add a hill climber algorithm to evaluate the search space and justify the use of a coevolutionary algorithm.

ACKNOWLEDGMENTS

This thesis would never have been possible without the support of the people around me. I would like to thank my advisor, Dr. Daniel Tauritz, for helping me get through the entire research process. His domain expertise, incredible work ethic, and boundless optimism have helped make these ideas into a reality.

Dr. Alexander Kent helped me understand how I could focus my research on network security in a practical way. His expertise in cyber security was crucial, and his encouragement enabled me to work through the most frustrating and difficult parts of the research. I would also thank Dr. Sriram Chellappan for teaching me how to view cyber security problems, especially how to focus on each piece without being overwhelmed by the whole. Many more professors and industry practitioners have taught me important lessons over the years, for which I am eternally grateful.

I would also like to thank two organizations for providing funding for my research: Los Alamos National Laboratory via the Cyber Security Sciences Institute under subcontract 259565 and the Missouri S&T Intelligent Systems Center. This work would not be possible without their contributions.

Last, but not least, I would like to thank my family and friends. Throughout childhood my parents, Clara and Steve, encouraged my curiosity about the world and everything in it, giving me every opportunity they could to learn and grow as a person. My siblings made sure I never got too full of myself, and countless members of my extended family have given me tremendous support over the years. Finally, my friends kept me from taking life too seriously, helping me remember that sometimes it really is about the journey and not the destination. Thank you all, for everything.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION

1. INTRODUCTION

Cyber security is increasingly a challenge for organizations ranging from business to government, as well as individual users. As more devices are connected to the Internet, they become more accessible, but it also implies more potential targets and a larger attack surface area that requires careful fortification. Defense systems that require less expert knowledge and can adapt quickly to threats are strongly needed to combat the rise of cyber attacks. Unfortunately, trying to enumerate all possible attack vectors and defensive strategies is not feasible. Computational intelligence techniques can be used to effectively explore many potential solutions in a reasonable timeframe and may yield unintuitive solutions since they are unaffected by human bias.

Complex configuration schemes and a distributed architecture are often required when developing and testing systems used in network security. Several architectures have been created for this purpose, but most of them are meant to provide a platform for running cyber security experiments as opposed to automating experiment processes. In the first paper, we propose a framework termed Distributed Cyber Security Automation Framework for Experiments (DCAFE) that employs software agents to manage system roles, emulate end users, automate data collection, analyze results, and initiate new experiments without human intervention. The framework was used to run experiments that optimized Windows logging configurations in order to quickly find indicators of compromise. The contribution of the work is the creation of a model for experiment automation and control in a distributed system environment.

Predictive analysis of adversaries is another thorny issue in cyber security. Game theory can be used to mathematically analyze adversary models, but its scalability limitations restrict its use to simple, abstract models. Computational game theory allows us to scale classical game theory to large, complex systems capable of modeling real-world environments. In the second paper, we choose to utilize coevolution, an approach where each player's fitness is dependent on its adversaries, since attackers and defenders in network security fit the adversarial model. Here we propose a framework termed Coevolutionary Agent-based Network Defense Lightweight Event System (CANDLES) that can coevolve attacker and defender agent strategies and capabilities and evaluate potential solutions with a custom, abstract computer network defense simulation.

The third paper is a continuation of the CANDLES project in which we rewrote key parts of the framework. Attackers and defenders have been redesigned to evolve pure strategy, rather than a combination of strategy and available capabilities, or components. A new network security simulation is devised which specifies network architecture as a graph and uses a constrained number of discrete turns to add a temporal aspect. Then we add a hill climber algorithm to evaluate the search space and justify the use of a coevolutionary algorithm.

PAPER

I. DCAFE: A DISTRIBUTED CYBER SECURITY AUTOMATION FRAMEWORK FOR EXPERIMENTS

George Rush[1], Daniel R. Tauritz[1], and Alexander D. Kent[2]

[1]*Natural Computation Laboratory, Department of Computer Science, Missouri University of Science and Technology, Rolla, Missouri, U.S.A.*

[2]*Advanced Computing Solutions, Los Alamos National Laboratory, Los Alamos, New Mexico, U.S.A.*

ABSTRACT

Cyber security has quickly become an overwhelming challenge for governments, businesses, private organizations, and individuals. In an increasingly connected world, the trend is for resources to be accessible from anywhere at any time. Greater access to resources implies more targets and potentially a larger surface area for attacks, which makes securing systems more difficult. Automated and semi-automated solutions are needed to keep up with the deluge of modern threats, but designing such systems requires a distributed architecture to support development and testing. Several such architectures exist, but most only focus on providing a platform for running cyber security experiments as opposed to automating experiment processes. In response to this need, we have built a distributed framework based on software agents which can manage system roles, automate data collection, analyze results, and run new experiments without human intervention. The contribution of this work is the creation of a model for experiment automation and control

in a distributed system environment, and this paper provides a detailed description of our framework based on that model.

1. INTRODUCTION

A significant challenge in developing cyber security software is the difficulty in building a suitable test environment. While powerful network and software platforms exist, they typically focus on *enabling* researchers to run experiments and collect data rather than *managing* the experiments and reacting dynamically based on results. A user for one of these platforms might start by spending weeks setting up an experiment. Then the experiment is run, results are collected and manually analyzed, and tweaks are made to system parameters before running the experiment again. This cycle can continue hundreds or thousands of times and divert limited resources without further automation of the experimental process. As such, it is highly beneficial to automate management of system roles, data collection, result analysis, experimental parameters, and experiment execution.

In this paper, we present the Distributed Cyber Security Automation Framework for Experiments (DCAFE). DCAFE consists of several distributed controls, implemented as software agents, running on a network of virtual machines (VMs). It was initially designed for gathering and analyzing Windows event log data, though that is only one of many possible uses for this system. Our goal was to find an optimal data collection configuration to detect a given attack or network enumeration technique. As such, that experimental objective guided many of our design decisions.

The VMs implemented here represent the attacker, victim, and central control system in the network. The distributed agents are designed to automate the user roles of attacker and victim, gather and analyze event data, and synchronize actions from a central point in the network. However, while our implementation utilizes a synchronous com-

munication model, it is important to note that DCAFE is fully capable of operating in an asynchronous environment. The agents are synchronous in our implementation due to a specific requirement in our experiments; namely, communicating accurate time windows during data collection requires that agents are aware when various actions are completed on the network.

2. RELATED WORK

Several platforms exist for implementing cyber security testbeds, though none directly enable the automation capabilities that we describe. Emulab is a network testbed with extensive auto-configuration and large-scale experimentation capabilities. It allows for the use of emulation with arbitrary network topologies or even live testing on the Internet by using the RON and PlanetLab testbeds [1]. While Emulab provides a plausible foundation for our approach, it is not designed for experiment automation. Our model could be implemented in experiments run with Emulab at the cost of Emulab's far greater hardware and software environment prerequisites. While possible, keeping the model separate means greater applicability to a variety of system environments.

Similarly, Cyber Defense Technology Experimental Research (DETER) is a complex framework used for risky experiment management, experiment health monitoring, and dynamic federation of different testbed facilities [2]. While using this framework might have allowed more exact control and monitoring of the experiment, it is not designed to automate system or user roles.

Several unnamed Supervisory Control and Data Acquisition (SCADA) testbeds have also been designed for single simulation, federated simulation, and emulation/implementation [3]. These are useful as general models, but are specific to SCADA systems at the implementation level. Since our research involves typical desktop and server

operating environments, SCADA frameworks fall outside the experiment scope. Also, experiment automation is not considered by these testbeds.

Strategic and Tactical Resiliency Against Threats to Ubiquitous Systems (STRATUS) is a model-driven framework used for attack detection and mitigation. It utilizes models for missions, components, vulnerabilities, trust levels, attack plans, and host and cluster organization [4]. While this is not relevant to testbed technology, it is comparable to our example implementation as another model designed for attack detection. While STRATUS may be a viable model, we preferred a more lightweight approach.

In general, no existing framework met the needs of our planned experiments. A virtualization platform with internal networking, emulated hardware, and desktop/server operating systems was necessary, as well as a distributed software agent design and several system-specific capabilities. Data had to be contained on-site, and penetration testing had to be possible without disrupting outside systems or networks. Most importantly, the framework had to enable full automation of experimental procedures. DCAFE was developed to meet these requirements.

3. IMPLEMENTATION DETAILS

3.1. System Virtualization. Oracle VM VirtualBox was chosen as the virtualization platform due to its popularity and ease of use [5]. Three VMs were created and connected to an 'internal network' created by Virtualbox. Each VM ran a different operating system (OS): Windows 7, Kali Linux, or Ubuntu Server. Each was chosen due to its unique traits and suitability for a given role in the experiment. Occasionally, programs and frameworks were also installed on each VM to facilitate experimental procedures. System roles and responsibilities are listed in Table 1.1.

Table 1.1. System Roles and Responsibilities

Role	**OS**	**System Responsibilities**	**Agent Responsibilities**
victim	Windows 7	Snare Agent sends event log data to the command server.	Update the Snare configuration when requested by the command server.
attacker	Kali Linux	Execute attack commands for the software agent.	Launch attacks and network scans when requested by the command server.
command server	Ubuntu Server	The service rsyslog is used to receive event log data in Syslog format from the victim.	Isolate and analyze victim event logs, determine Snare configurations to test, send attack commands, and output results.

Windows 7 was chosen for the victim role due to its enormous popularity in business, government, and individual use. Snare Agent for Windows [6], a program designed to collect and transfer event log information in real-time, was also installed and configured to send system log data to a Syslog server in the virtual network. Kali Linux was chosen for the attacker role due to its penetration testing capabilities and popularity with cyber security professionals. Ubuntu Server was chosen for the command server role due to its broad support, documentation, and easy setup for Syslog, SSH, and related capabilities.

It is worth noting that scalability could be a concern depending on the number of available Windows licenses. Also, should tens or hundreds of systems be necessary for a particular experiment, one might prefer an emulation platform with more autoconfiguration capabilities than VirtualBox. In that case, a testbed like Emulab might be a better option.

3.2. Software Agent Design. Distributed software agents are used here to control the experimental process, configure services on the fly, execute actions in various system

roles, and automate data collection and analysis. Several communication models can be used with the agents, including peer-to-peer, client-server, hybrid, and subscription models. Agents can either work autonomously, deciding when to act on their own, or accept commands from other agents. Regardless of the design choices, any agent must be able to execute actions for its given role and communicate with other agents as necessary. One possible implementation is described here.

All of our VMs have a software agent written in Python. These agents are initialized with root privileges, and each plays a particular role in the experimental procedure. All agents communicate with each other as necessary by sending commands and data over network sockets using JavaScript Object Notation (JSON). The attacker and victim agents are 'dumb' agents in the sense that they only take orders from the command agent and reply when requested tasks have been completed. All agents operate synchronously in the sense that only a single agent can perform an action while the others wait, even though other actions may be carried out simultaneously by the operating systems where the agents reside.

The attacker agent receives messages in JSON format from the command agent. These messages contain commands (strings) and a duration in seconds for each command (integers). The attacker agent executes each command in a subshell and then waits for the specified duration. Once all commands are complete, the agent will reply with a completion message to the command agent.

The victim agent also receives messages in JSON format from the command agent. These messages contain Snare Objective Configuration rules. The victim agent stops the Snare service, clears the current configuration from the Windows Registry, inserts new rules, and restarts the Snare service. Then the agent will reply with a completion message to the command agent.

The command agent tracks experiment progress, isolates and analyzes Syslog data, generates Snare configurations and attack commands, and periodically outputs configuration and result information to a file. This is the only intelligent agent, changing its actions according to feedback from experimental data. It coordinates the actions for all other agents and stores all results locally.

3.3. Snare Configuration Generation. A Snare Objective Configuration is composed of an arbitrary number of rules. Multiple options can be selected on a given rule to gather various types of data at different levels, from very broad to extremely detailed. The major categories and relevant values that were manipulated for Snare rules during these experiments are listed in Table 1.2.

Due to the many levels of detail for logging capabilities, it is possible to start with a set of Snare configuration rules that each represent a highly inclusive general category, and then depending on which results are most interesting, explore more narrow configurations. In this iterative process, the first round of configurations might use different event IDs, but include all event sources and return values per rule. The next round might use one event ID and one event source per rule, but still have all return values. Each round can have many iterations depending on which of the previous candidates were accepted and require further investigation, and at every level, only one rule is ever tested at a time. Eventually, a set of extremely narrow but useful rules will remain, and those will be output as results for manual review. Note that this algorithm is greedy in that not every configuration is considered even though all types of data are analyzed for their usefulness. As a result, there are potential circumstances under which this algorithm may return suboptimal results.

The reason why this approach works so well for Snare data is best described with a visual analogy. Suppose Snare configurations can be represented by a map. The largest sections of the map are rules for all-inclusive general categories of data, while smaller sec-

Table 1.2. Important Fields with Corresponding Attribute Values in Snare Rules

Event ID	**Event Source**	**Return Value**
File_Events	Active Directory Service	ActivityTracing
Filtering_Events	Application	Critical
Logon_Logoff	DFS-Replication	Error
Process_Events	Domain Name Server	Failure
Reboot_Events	Legacy FRS	Information
Security_Policy_Events	Security	Success
User_Group_Management_Events	System	Verbose
User_Right_Events		Warning

tions within them are the more specific rules. Data generated by each configuration is a hill in the landscape, with the height of each hill determined by the volume of the data produced. We found through experimentation that most of the landscape is flat, at least for baseline data (generated by system processes) and attack data. This means that the most specific rules rarely generated any data at all, and broad categories tended to either produce useful data quickly or not at all. Due to this property, it was possible to scan the landscape at a high level, then quickly drill down and find the most specific rules possible to provide potential attack indicators.

4. ITERATIVE EXPERIMENT PROCEDURE

The experiment procedure is initialized by powering on all the VMs and initializing the agents on the victim and attacker VMs who await further commands. The main control program on the command server is started by the user, at which point the following automated iterative procedure commences (visualized in Fig. 1.1):

1. The command server determines the current Snare configuration to test and sends an update command to the victim.

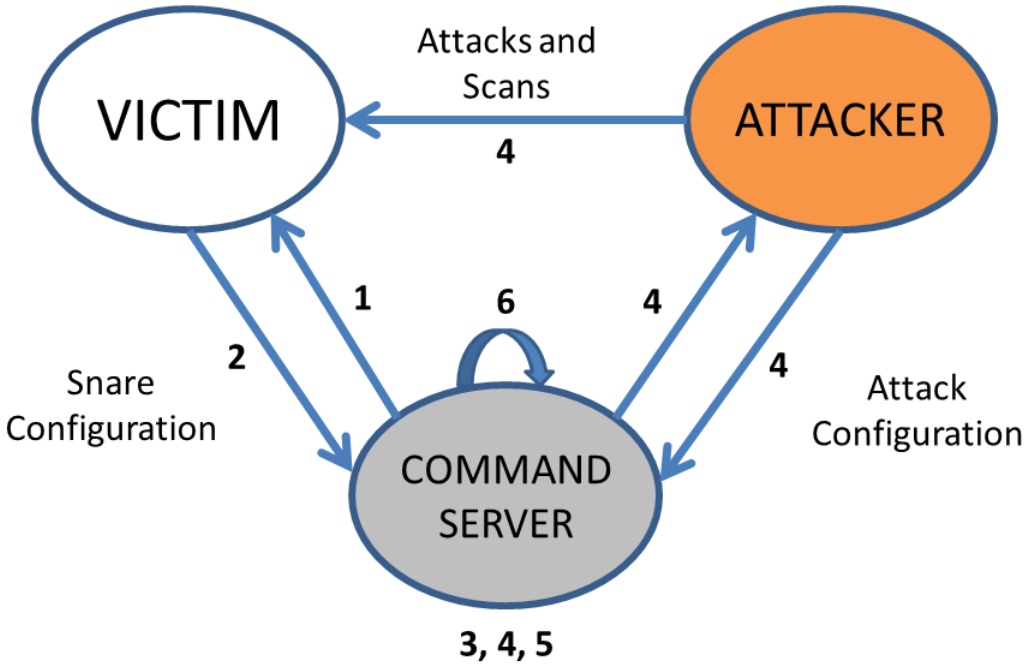


Figure 1.1. System Interaction Diagram

2. The victim updates the configuration and restarts Snare, then sends a completion message to the command server.
3. The command server gathers baseline data. This requires waiting an arbitrary amount of time for events caused by Windows background processes to accumulate. It then copies the results to a separate file for later analysis.
4. The command server gathers attack data. This is done the same way as baseline data, except that a command is sent to the attack machine while data accumulates. Every command in our experiments requires sending some type of data to the victim.

The attack machine executes the command and returns a completion message to the command server.

5. The baseline data is compared to the attack data, and any unique events or event types in the attack data are noted as possible attack indicators.
6. Unless all Snare configurations have been tested or rejected, restart the process from Step 1.

4.1. Event Log Analysis. Analyzing event logs requires comparing baseline data to attack data. Baseline data consists of events generated by normal Windows processes. No user actions were included in the baseline data, though the software agent was running in a command window on the victim machine in order to update Snare configurations as necessary. Both a Pass-the-Hash exploit and a SYN Stealth scan were used to generate attack data in separate runs of the experiment.

The first attack, Pass-the-Hash, is a technique that allows an intruder to authenticate on a remote system with the hashed value for a user's password [7]. Since many systems hash passwords as a security measure, it can be easier to obtain the hashed value as opposed to getting the password in plain text. The second attack, a SYN Stealth scan, is also called a TCP SYN scan or half-open scan. It can reliably determine whether a network port is open, closed, or filtered, and it is considered stealthy since it does not fully establish a connection [8]. This makes it less likely to show up in event logs for systems or applications.

Once baseline and attack behaviors were determined, data gathering and analysis could begin. All analysis starts with raw event data, and an example of this is shown in Fig. 1.2. Note that the victim's hostname is duna, and the event indicates that a network connection was permitted by the Windows Filtering Platform.

Pre-processing was applied to the raw event data in order to facilitate clustering and to reduce the dimensionality of the data. Numerical data and punctuation were stripped

from the events to leave only the terms composed of ASCII text. These were then converted into lowercase and used to build term sets. The example event from Fig. 1.2 is translated into the term set in Fig. 1.3.

Next, the baseline events were clustered, followed by the attack events. If the attack events formed new clusters using the baseline data, then the attack data was considered unique enough to provide a possible attack indicator. Any attack indicator, and its Snare configuration by association, was also given an objective value and stored in the final results if the value exceeded a user-specified threshold.

```
Dec 26 16:21:21 duna MSWinEventLog 4 Security 7 Thu Dec 26 16:21:21
2013 5156 Microsoft-Windows-Security-Auditing N/A N/A Success Audit
duna Filtering Platform Connection The Windows Filtering Platform
has permitted a connection.  Application Information:  Process ID: 4
Application Name:  System Network Information:  Direction:  Inbound
Source Address:  192.168.66.3 Source Port:  445 Destination Address:
192.168.66.4 Destination Port:  37041 Protocol:  6 Filter Information:
Filter Run-Time ID: 66057 Layer Name:  Receive/Accept Layer Run-Time
ID: 44 6
```

Figure 1.2. Example Snare Event

```
{'', 'receive', 'filtering', 'protocol', 'connection', 'auditing',
'process', 'platform', 'dec', 'thu', 'run', 'security', 'name', 'id',
'network', 'source', 'direction', 'port', 'microsoft', 'information',
'permitted', 'time', 'the', 'destination', 'a', 'mswineventlog',
'success', 'windows', 'n', 'inbound', 'system', 'layer', 'address',
'audit', 'filter', 'accept', 'duna', 'has', 'application'}
```

Figure 1.3. Example Event Term Set

4.2. Clustering Algorithm. The clustering method was designed with the event logs in mind, for highly textual data with unreliable timestamps due to clock drift. For this purpose, we built a custom algorithm similar to the crisp nearest prototype classifier [9]

which addresses the specific needs of our experiments. In particular, the natural organization of the data is exhibited through the behavior of an agglomerative clustering algorithm with dynamic classes. Each event can belong to only one cluster, and events start out with no cluster membership initially. There is no labeled data, and the number of clusters is not specified since they are dynamically generated. Each cluster consists of one or more events, but is represented by a prototype for ease of calculations and lower space complexity. The prototype is a set containing all terms from the cluster members. A similarity threshold is used when deciding if an event should join a cluster, and the similarity measure is defined in (1.1).

$$similarity = \frac{(number\ of\ matching\ terms) \times 2}{(total\ number\ of\ terms)} \tag{1.1}$$

The similarity measure is symmetric in that the order of the input and prototype does not affect the result. This method also prevents any cluster from spreading too far, a property which encourages formation of smaller and more distinct clusters. Pseudocode for the clustering algorithm is provided in Algorithm 1. Note that any cluster prototype and event term set are unlikely to have the same number of terms, and the similarity can never exceed 1.0. The reason for multiplying the number of matching terms by two is to normalize the range of similarity values to [0,1]. If the similarity exceeds the similarity threshold for the current iteration, then the event joins the cluster. If the event in question is not found to be similar enough to any existing cluster, a new cluster is formed with the event as its only member.

4.3. Objective Value Calculation. The objective value of the Snare configuration for a given event set is calculated by starting with a low similarity threshold (0.1), forming clusters with the baseline data, and attempting to form new clusters with the attack data. Should zero new clusters be formed from the attack data, the threshold is incrementally increased, and the clustering process is repeated. The objective value is calculated as the

inverse of the similarity threshold required for the attack data to form new clusters, as defined in (1.2). Pseudocode for the objective value algorithm is provided in Algorithm 2.

$$objective\ value = \frac{1}{(current\ similarity\ threshold)} \tag{1.2}$$

The result is that attack data with highly distinctive events compared to the baseline will form new clusters at a low similarity threshold, consequently receiving a high objective value. Should new clusters not form at any threshold (or if zero events are available for a given Snare configuration), the objective value is zero. Any Snare configuration with an objective value of zero is immediately rejected, while all others are stored for later analysis. In practice, due to the flat data landscape, we found most Snare configurations to have an objective value of zero.

5. PRELIMINARY RESULTS

Initial experiments run using DCAFE have resulted in finding several indicators of compromise representing known exploits. While we cannot elaborate on those results at present due to their sensitive nature, they have provided useful direction for later experiments and validation of DCAFE's usefulness in experiment automation.

6. CONCLUSION AND FUTURE WORK

This paper introduces DCAFE, a practical, light-weight, distributed framework for the automation of cyber security research experiments, based on software agents, which can manage system roles, automate data collection, analyze results, and run new experiments without human intervention. For a given attack specification, DCAFE has demonstrated its ability to discover event log configurations containing indicators of compromise, which can be fruitfully employed by attack detection mechanisms.

DCAFE currently provides a model and example implementation for distributed software agents used to automate the experimental process. This paper is meant to act as a guide for others to design similar agents and implement the technologies described here. Unfortunately, the source code from our own experiments cannot currently be released due to the sensitive nature of the work. As such, future work might involve the creation of a standard software package that other researchers could download and use as a base for their own experiment designs.

7. ACKNOWLEDGMENT

This work was supported in part by Los Alamos National Laboratory via the Cyber Security Sciences Institute under subcontract 259565 and in part by the Missouri S&T Intelligent Systems Center.

Algorithm 1 Clustering Algorithm

```
function CALCULATESIMILARITY(setA, setB)
    numOfMatches ← 0
    for each termA in setA do
        if termA in setB then
            numOfMatches ← numOfMatches + 2
        end if
    end for
    total ← SIZE(setA) + SIZE(setB)
    return numOfMatches/total
end function

function FORMCLUSTERS(eventList, clusterList, clusterFormationThreshold)
    for each event in eventList do
        bestMatch ← None
        bestScore ← −1
        for each cluster in clusterList do
            similarity ← CALCULATESIMILARITY(cluster, event)
            if similarity ≥ clusterFormationThreshold and similarity > bestScore then
                bestMatch ← cluster
                bestScore ← similarity
                break
            end if
        end for
        if bestMatch = None then
            newCluster ← EVENTCLUSTER(event)
            clusterList.APPEND(newCluster)
        else
            bestMatch.signature ← bestMatch.signature.UNION(event)
        end if
    end for
    return clusterList
end function
```

Algorithm 2 Objective Value Algorithm

```
SIMILARITY_THRESHOLD_STEP_SIZE ← 0.1
SIMILARITY_THRESHOLD_MAX ← 1.0

function CALCULATEOBJECTIVEVALUE(baselineEvents, attackEvents)
    objectiveValue ← 0.0
    similarityThreshold ← SIMILARITY_THRESHOLD_STEP_SIZE
    while similarityThreshold ≤ SIMILARITY_THRESHOLD_MAX do
        normalClusters ← FORMCLUSTERS(baselineEvents, similarityThreshold,
            None)
        allClusters ← FORMCLUSTERS(attackEvents, similarityThreshold,
            normalClusters)
        if LENGTH(allClusters) > LENGTH(normalClusters) then
            objectiveValue ← 1.0/similarityThreshold
            break
        end if
        similarityThreshold ← similarityThreshold+
            SIMILARITY_THRESHOLD_STEP_SIZE
    end while
    return objectiveValue
end function
```

BIBLIOGRAPHY

[1] Emulab.net - emulab - network emulation testbed home. https://www.emulab.net/. Accessed: 2014-02-24.

[2] Terry Benzel, Bob Braden, Ted Faber, Jelena Mirkovic, Steve Schwab, Karen Sollins, and John Wroclawski. Current Developments in DETER Cybersecurity Testbed Technology. In *Proceedings of the Cybersecurity Applications & Technology Conference For Homeland Security (CATCH)*, pages 57–70. IEEE, 2009.

[3] Annarita Giani, Gabor Karsai, Tanya Roosta, Aakash Shah, Bruno Sinopoli, and Jon Wiley. A Testbed for Secure and Robust SCADA Systems. *ACM SIGBED Review - Special issue on the 14th IEEE real-time and embedded technology and applications symposium (RTAS'08) WIP session*, 5(2), July 2008.

[4] Mark Burstein, Robert Goldman, Paul Robertson, Robert Laddaga, Robert Balzer, Neil Goldman, Christopher Geib, Ugur Kuter, David Mcdonald, John Maraist, et al. STRATUS: Strategic and Tactical Resiliency against Threats to Ubiquitous Systems. In *Proceedings of the Sixth International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, pages 47–54. IEEE, 2012.

[5] Oracle vm virtualbox. https://www.virtualbox.org/. Accessed: 2014-02-24.

[6] Snare eventlog agent for windows - event log transfer to snare & syslog servers - open source. http://www.intersectalliance.com/projects/BackLogNT/. Accessed: 2014-02-24.

[7] Psexec pass the hash - metasploit unleashed. http://www.offensive-security.com/metasploit-unleashed/PSExec_Pass_The_Hash. Accessed: 2014-05-06.

[8] Port scanning techniques. http://nmap.org/book/man-port-scanning-techniques.html. Accessed: 2014-05-06.

[9] James M Keller, Michael R Gray, and James A Givens. A Fuzzy *K*-Nearest Neighbor Algorithm. *Transactions on Systems, Man, and Cybernetics*, SMC-15(4):580–585, July/August 1985.

II. COEVOLUTIONARY AGENT-BASED NETWORK DEFENSE LIGHTWEIGHT EVENT SYSTEM (CANDLES)

George Rush[1], Daniel R. Tauritz[1], and Alexander D. Kent[2]

[1]*Natural Computation Laboratory, Department of Computer Science, Missouri University of Science and Technology, Rolla, Missouri, U.S.A.*

[2]*Cyber Futures Laboratory, Los Alamos National Laboratory, Los Alamos, New Mexico, U.S.A.*

ABSTRACT

Predicting an adversary's capabilities, intentions, and probable vectors of attack is in general a complex and arduous task. Cyber space is particularly vulnerable to unforeseen attacks, as most computer networks have a large, complex, opaque attack surface area and are therefore extremely difficult to analyze. Abstract adversarial models which capture the pertinent features needed for analysis, can reduce the complexity sufficiently to make analysis feasible. Game theory allows for mathematical analysis of adversarial models; however, its scalability limitations restrict its use to simple, abstract models. Computational game theory is focused on scaling classical game theory to large, complex systems capable of modeling real-world environments; one promising approach is coevolution where each player's fitness is dependent on its adversaries. In this paper, we propose the Coevolutionary Agent-based Network Defense Lightweight Event System (CANDLES), a framework designed to coevolve attacker and defender agent strategies and evaluate potential solutions with a custom, abstract computer network defense simulation. By performing a qualitative analysis of the result data, we provide a proof of concept for the applicability of coevolu-

tion in planning for, and defending against, novel attacker strategies in computer network security.

1. INTRODUCTION

Great strides are needed in the defensive tools and technologies available to cyber security practitioners, as the asymmetric nature of cyber warfare [1] puts defending practitioners at a distinct disadvantage; i.e., cyber attackers get to decide when and where to attack, without the need for physical presence providing advance notice to the cyber defenders who must scramble to quickly determine that an attack is occurring, select an appropriate defense, and execute it. In cyber security, as in many security-related fields, it can be prudent to evaluate worst-case attack scenarios. Whether facing insider threats or sophisticated adversaries outside one's networks, it is important to develop threat and attack models such that one can predict an adversary's capabilities, intentions, and probably vectors of attack. Cyber space is particularly vulnerable to unforseen attacks due to most computer networks having a large, complex, opaque attack surface area and therefore being extremely difficult to analyze. Invariably, the large number of variables involved in both offensive and defensive strategies makes an exhaustive search of all strategies infeasible. Abstract adversarial models which capture the pertinent features needed to analyze such strategies, can reduce the complexity sufficiently to make analysis feasible.

Game theory allows for mathematical analysis of adversarial models; however, its scalability limitations restrict its use to simple, abstract models. Computational game theory is focused on scaling classical game theory to large, complex systems capable of modeling real-world environments; one promising approach is coevolution where each player's fitness is dependent on its adversaries. Coevolution can be employed to explore the as-

sociated search spaces, examining strategic capabilities and estimating how each side will adapt in response to moves made by an opponent.

It makes sense to focus resources on securing systems such that attacks become less effective. There are several well-known ways to harden systems against attack, from limiting access permissions to patching software and reducing exposed network services. These are useful measures to reduce the attack surface and make it easier to defend vital assets. Yet for all the efforts taken to secure systems, intrusions still occur in many organizations, often without their knowledge [1]. Once it is acknowledged that intrusions cannot be stopped entirely, awareness, damage mitigation, and disaster recovery become important goals. In particular, knowing how an adversary could break into a network and devising possible counter-strategies is vital. In this paper, we propose the Coevolutionary Agent-based Network Defense Lightweight Event System (CANDLES), a framework designed to coevolve attacker and defender agent strategies and evaluate potential solutions with a custom, abstract computer network defense simulation. Our approach coevolves two populations containing attacker and defender strategies in a network defense scenario, and the fitness of strategies is determined through simulations. In this way, unique, near-optimal strategies for both sides can be discovered. By performing a qualitative analysis of the result data, we provide a proof of concept for the applicability of coevolution in planning for, and defending against, novel worst-case attacker strategies in computer network security, allowing an end user to develop strategies and set up test scenarios. This offers new possibilities for analyzing and reinforcing defensive capabilities against unknown threats.

2. RELATED WORK

Coevolution has been used to find better placements for Flexible AC Transmission System (FACTS) devices [2], which are used to prevent cascading blackouts in electric

transmission systems [3], and also to evolve attackers and defenders for graph-based network theory models [4]. Defenses have been designed that can predict the behavior of adaptive adversaries using a combination of game theory and machine learning [5], and machine learning has likewise been used to predict the nature of relationships in adversarial social networks [6]. Strategies have been developed for defending against Distributed Denial-of-Service (DDoS) attacks using a Bayesian game theoretic framework [7], and hidden Markov models have been developed to detect cyber attacks in network traffic [8].

Current approaches to developing computer network defense strategies largely focus on either pure game theoretic models [9] or real world implementations and emulated systems [10, 11]. Emulated systems more accurately reflect real world conditions, but require a fair amount of resources and configuration knowledge. Game theoretic models are mathematically elegant, but they do not scale well to larger solution spaces. Our approach falls somewhere in the middle, as our experiments employ a custom network security simulation. Coevolution allows us to approximate game theory solutions for large search spaces, thus providing scalability beyond the limits of classic game theory, and the simulation provides a more realistic strategy evaluation without the configuration difficulty of most emulated systems.

3. METHODOLOGY

There are two main parts to the CANDLES framework: the coevolutionary algorithm (CoEA) and the network security simulation. A CoEA was chosen over other stochastic methods since it most closely represents the natural dynamic between attackers and defenders in cyber space. Namely, they evolve their capabilities over time in response to actions taken by opponents. The network security simulation was developed to evaluate

potential solutions in the CoEA, and it is used to simulate cyber defense scenarios given an abstract set of capabilities for both attackers and defenders.

In our model, attacker capabilities include exploits and reconnaissance, or recon. Recon techniques identify features of defender machines or networks and increase the chance of exploit success. Exploits are used to compromise a vulnerability and deliver a payload used to exfiltrate data. In our simulation, a vulnerability may be specific to a service or operating system, and there are no payloads designed to damage enemy systems. This is meant to model a stealthy attacker with advanced intrusion capabilities and a desire to gather as much information as possible. Attacker profit is defined in the simulation as the total amount of information exfiltrated from the defender's machines.

Defender capabilities include detection systems, mitigation techniques, and shutting down machines. Detection systems attempt to detect both exploits and reconnaissance and inform the defender. Upon detection, dynamic mitigation techniques can be used to attempt to stop the attack. Should the attack not be detected or if dynamic mitigation fails, then static mitigation techniques may still prevent the attack from working. Dynamic mitigation is meant to represent an Intrusion Prevention System (IPS), whereas static mitigation represents passive defenses like closed network ports, software patches, or limited user privileges. Paranoia is also used to represent how alert a defender is after exploits have been detected, and high paranoia will eventually cause the defender to shut down targeted machines.

3.1. Classes. This section describes classes that provide the structural basis for the rest of the framework.

3.1.1. Action. The Action class represents a possible action by any entity in the simulation, and it specifies both a target and a technique. Since currently only Attackers use Actions, techniques can be either Recon Techniques or Exploits.

3.1.2. Attacker. The Attacker class is initialized with an Attacker Solution and executes Actions during the simulation. Which Actions to use are determined using capabilities in the Attacker Solution, and state information is used to track reconnaissance information and event history. The structure for this class is visualized in Fig. 1.4.

3.1.3. Attacker Solution. An Attacker Solution specifies attack capabilities by defining available Recon Techniques and Exploits, and it specifies strategy through a target list of Defender Machines. In general, an attacker should not attack too many targets since that raises the Defender's paranoia level, which makes it harder to extract information once machines are shut down. On the other hand, attacking too few targets or targets of low value will not provide enough profit to be worthwhile. Attacker Solutions compose the first of two populations in the CoEA, and they are initialized using population seeds as described in Subsection 4.4.

3.1.4. Defender. The Defender class is initialized with a Defender Solution and responds to Actions by an attacker using detection systems and mitigation techniques. State information is used here to track the paranoia level. Paranoia is a mechanism that increases with detected exploits and allows the Defender to shut down machines upon reaching certain thresholds. Note that shutting down machines incurs a passive cost for productivity loss, but it prevents further data exfiltration. The structure for this class is visualized in Fig. 1.5.

3.1.5. Defender Machine. Each Defender Machine has a unique ID, intrinsic value, operating system, and list of available services. State information tracks whether the machine is active and any status effects incurred as a result of Exploits or Recon Techniques.

3.1.6. Defender Solution. A Defender Solution specifies a Defender's capabilities by defining available Detection Systems and Dynamic Mitigations, and it specifies strategy

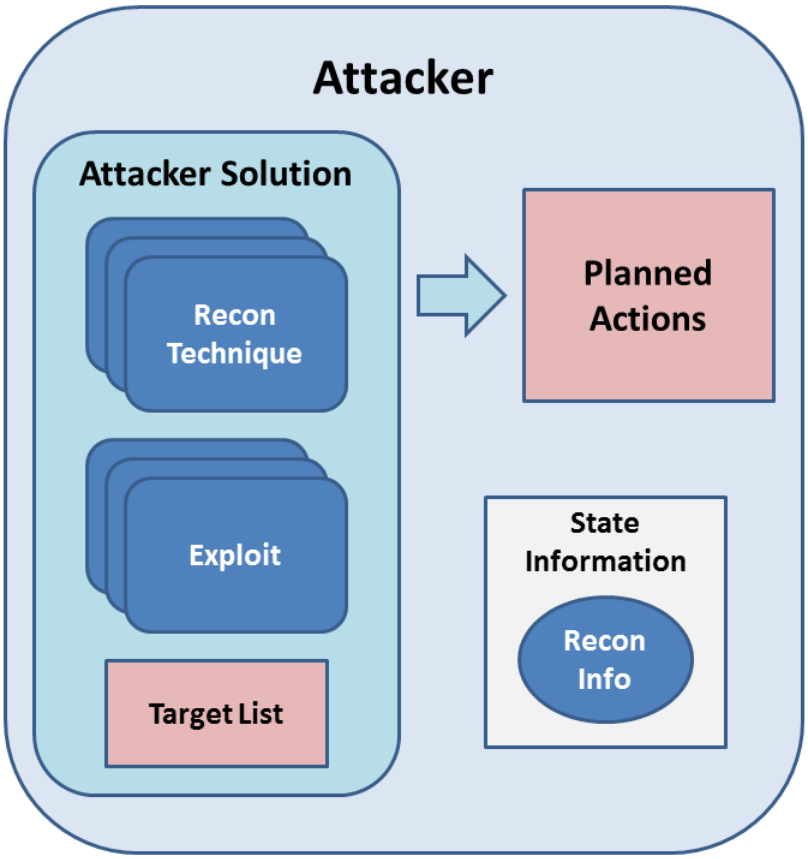


Figure 1.4. Attacker Diagram

through suspected targets and paranoia thresholds. Suspected targets are used to perform extra defensive measures with a one-time cost, and paranoia thresholds are used to determine when to shut down machines. Defender Solutions compose the second population in the CoEA, and they are initialized using population seeds as described in Subsection 4.4.

3.1.7. Detection System. Each Detection System has a one-time installation cost and four separate detection probabilities for the following events against any target: successful recon, failed recon, successful exploit, and failed exploit.

3.1.8. Dynamic Mitigation. Each Dynamic Mitigation has a local execution cost, a user cost (to represent spent effort or time), and a probability of success. Note that

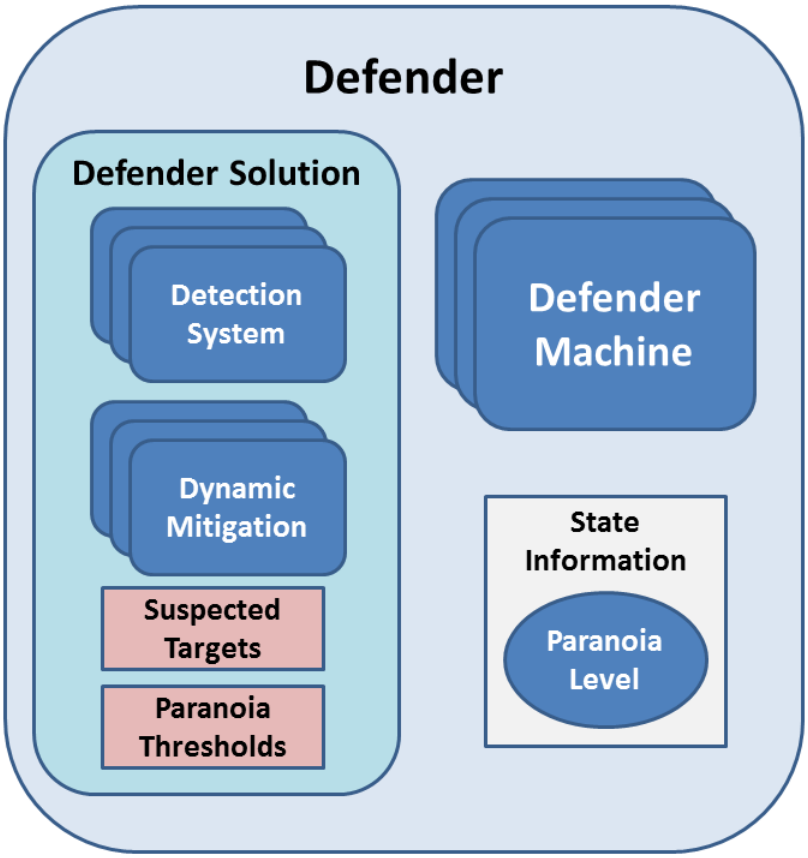


Figure 1.5. Defender Diagram

Dynamic Mitigations can also be used as a static defensive measure on suspected targets. This means the costs are only incurred once per suspected target, but they are incurred regardless of whether or not that target is attacked.

3.1.9. Exploit. Each Exploit has a value multiplier, a probability of success, and one or more constraints. The value multiplier indicates the utility of the payload since a higher multiplier will extract more value, representing information, from a given Defender Machine. The constraints specify the operating system or services required for the Exploit to function correctly.

3.1.10. Recon Technique. Each Recon Technique has type information and a probability of success. Type information indicates which effect the Recon Technique will have if successful, and the three possible effects are to increase the probability of exploit success, identify the OS, or identify running services.

3.2. Coevolutionary Algorithm. The CoEA evolves a population of Attacker Solutions against a population of Defender Solutions. Attacker Solutions use targets, Recon Techniques, and Exploits while Defender Solutions use paranoia thresholds, a budget, suspected targets, Detection Systems, and Dynamic Mitigations. Note that in both solution classes, targets are references to Defender Machines. Both Attacker and Defender Solutions are initialized from population seeds as covered in Subsection 4.4.

Parent selection is random, and survivor selection uses truncation. For recombination, a random subset of values are chosen for each variable from each parent. For mutation, a target or technique is randomly added or removed from the given solution. Defender Solutions can also mutate paranoia thresholds to any value with a step size of 0.1 in the range [0, 1]. Note that paranoia thresholds specify the necessary paranoia level for a Defender to shut down machines, whereas the paranoia level itself just specifies how paranoid the Defender is at any time during a given simulation.

Fitness evaluation for any individual requires measuring its performance multiple times against several opponent solutions in the network security simulation and averaging the results. The exact number of opponents and number of simulations per opponent are determined by the CoEA configuration (listed in Subsection 4.3). Pseudocode for the fitness function is provided in Algorithm 3.

One unique aspect of our CoEA is that the fitness function is asymmetric. The attacker only attempts to maximize its profit (representative of information gained), but the defender wants to both minimize the attacker's profit and minimize its own costs. To do

Algorithm 3 Fitness Function

```
function CALCULATEFITNESS(individual, enemyPopulation)
    randomlyChosenOpponents ← CHOOSEOPPONENTS(enemyPopulation,
        NUM_OF_OPPONENTS)
    allOpponentAverages ← None
    for each opponent in randomlyChosenOpponents do
        simulationResults ← None
        for i in RANGE(NUM_OF_SIMULATIONS) do
            defenderCosts, attackerProfit ← RUNSIMULATION(individual, opponent)
            if individual.class = Attacker then
                result ← −attackerProfit
            else if individual.class = Defender then
                result ← −attackerProfit − defenderCosts
            end if
            simulationResults.APPEND(result)
        end for
        opponentAverage ← AVERAGE(simulationResults)
        allOpponentAverages.APPEND(opponentAverage)
    end for
    return AVERAGE(allOpponentAverages)
end function
```

this, the fitness value *per simulation* is calculated as shown in (1.3) and (1.4).

$$attacker\ fitness = -(attacker\ profit) \tag{1.3}$$

$$defender\ fitness = -(attacker\ profit) - (defender\ costs) \tag{1.4}$$

It is important to point out that fitness is determined on a scale of $(-\infty, 0]$, and attackers always want to minimize their fitness value while defenders want to maximize theirs. This mirrors reality in that the best case for defenders in cyber security is having zero losses from attacks and zero resources spent on defense, resulting in a maximum fitness value of zero. Attackers attempt to minimize their fitness value so that the attacker profit can be represented the same way in both equations.

3.3. Network Security Simulation. The network security simulation follows this process:

1. Attacker and Defender are initialized from solutions.
2. Defender prepares initial defenses.
3. Event loop begins:
   (a) Attacker performs an action (or does nothing).
   (b) Defender responds as necessary.
   (c) Passive costs are calculated.
   (d) Exit the loop if Attacker does not act.

During initialization, the Attacker will build attack sequences based on all possible combinations of available targets and techniques. It should be emphasized that the Attacker Solution only specifies the targets and techniques available for composing Actions in the attack sequence, and the Attacker's strategy for building attack sequences is static. The Defender then prepares its initial defense capabilities, which involves calculating installation costs for detection systems and static defensive measures. At this point, the main event loop begins. During each iteration of the event loop, the Attacker will start by looking up its next planned Action. If there are recon Actions remaining, those will be given priority. Should an exploit Action be selected, it will be executed only if the exploit constraints match any known recon information for the target. The Defender will respond to either recon or exploits by attempting to detect and mitigate them. If the event is successfully detected, dynamic mitigations are applied. If dynamic mitigations fail or if the event is not detected, then static mitigations may still take effect for suspected targets. The Defender also has a paranoia level which increases when exploits are detected. Should paranoia reach certain thresholds, then targeted machines will be shut down upon further detection of exploits or data exfiltration. Passive costs are also calculated for each iteration of the event

loop. The only passive costs are currently those for productivity loss by inactive Defender Machines, which are meant to balance out the tendency of a Defender Solution to simply shut down machines in order to block all attacks. Finally, should the Attacker have zero planned actions remaining, the event loop terminates.

4. EXPERIMENT PROCEDURE

The objective of our experiments is to demonstrate that coevolution can be used to explore and evaluate strategies in a cyber defense simulation. To this end, we have designed a number of experiments to explore various offensive and defensive scenarios.

4.1. Experiment Variables. The variables for individual experiments are the attacker population seed, defender population seed, and whether or not each population evolves (listed in Table 1.3). Population seeds are individual solutions used as a template for all population members during initialization, and they are listed as weak or strong depending on their capabilities. We also examine both static and dynamic populations in order to examine evolution in different contexts. It is easier, for example, to determine how well a population is evolving against a static opponent since it is not considered a moving target.

4.2. Result Data Format. There are 30 CoEA runs per experiment, and several types of result data are provided for each run. During a single run of the CoEA, the best Attacker Solution and Defender Solution are stored from each generation. After the run is complete, all the best solutions are output to file and used to generate a CIAO plot (Current Individual vs. Ancestral Opponents), which is used to visually convey the progress of two populations during coevolution [12]. The Defender Machine configuration is also output to provide contextual information if needed during analysis.

Table 1.3. Experiment Configuration Parameters

Experiment ID	Attacker Pop. Seed	Defender Pop. Seed	Attacker Evolution	Defender Evolution
X1	Weak	Weak	Static	Static
X2	Weak	Weak	Static	Dynamic
X3	Weak	Weak	Dynamic	Static
X4	Weak	Weak	Dynamic	Dynamic
X5	Weak	Strong	Static	Static
X6	Weak	Strong	Static	Dynamic
X7	Weak	Strong	Dynamic	Static
X8	Weak	Strong	Dynamic	Dynamic
X9	Strong	Weak	Static	Static
X10	Strong	Weak	Static	Dynamic
X11	Strong	Weak	Dynamic	Static
X12	Strong	Weak	Dynamic	Dynamic
X13	Strong	Strong	Static	Static
X14	Strong	Strong	Static	Dynamic
X15	Strong	Strong	Dynamic	Static
X16	Strong	Strong	Dynamic	Dynamic

To briefly explain CIAO plots, an example is provided in Fig. 1.6. This particular plot is from the attacker's perspective in experiment X8. For this perspective, darker regions indicate more success for the attacker, and the attacker's generation is plotted along the increasing y-axis while the defender's generation is plotted along the increasing x-axis. If this were the defender's perspective, darker regions would indicate more success for the defender, and the defender's generation would be plotted along the increasing y-axis while the attacker's generation would be along the increasing x-axis. In this CIAO plot, the attacker is most effective towards the last of its generations against the earliest generations of defenders as indicated by the nearly black pixels.

4.3. Important Configuration Parameters. All experiments ran with certain fixed values defined in the configuration, and this section will examine those values.

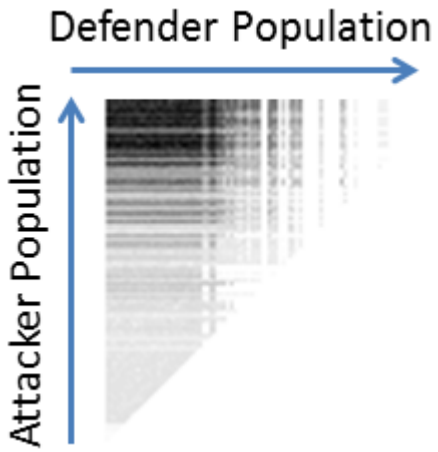


Figure 1.6. Example CIAO Plot

- 30 CoEA Runs

  This specifies the number of times a full CoEA is run per experiment. 30 CoEAs will result in 60 different CIAO plots (one each for attacker and defender perspective). This means that outliers can be identified, but when discussing results, we chose a representative pair of plots for each experiment.
- 10 Defender Machines

  This specifies how many machines the Defender has running on their network.
- 10 Fitness Opponents

  This is the number of opponents used to evaluate a given population member for fitness.
- 5 Simulation Runs

  This is the number of simulation runs between any two opponents during fitness evaluation. With 10 fitness opponents, this means that 50 simulation runs are needed to evaluate any population member.
- 100 Generations per CoEA

- Attacker Population Size of 100
- Defender Population Size of 100
- 30 Parents

  This is the number of parents selected from each population per generation, and each pair of parents produces a single offspring (i.e., 15 offspring for each population).
- Random Parent Selection
- Truncation Survivor Selection
- No Pre-Defined Defender Machines

  This is used to specify a Defender Machine configuration for the network security simulation. Since one is not given, the program will generate a random set of Defender Machines for the duration of each CoEA.

Many of these constants were chosen to meet time constraints in the experiments or to simplify the model, though there are a few exceptions. Random parent selection was chosen to add genetic variety to the offspring, and using truncation for survivor selection ensured that the population would attempt to improve over time, or at least not throw away the best solutions. There were never any pre-defined settings for the Defender Machines aside from the total number, as this forced solutions to be robust enough to succeed against multiple network configurations.

4.4. Population Seeds. Both of the initial populations were generated using either strong or weak solutions for each experiment. The design of each population seed is presented here.

4.4.1. Weak Attacker Seed. The weak attacker solution has only three out of ten possible targets and zero recon techniques or exploits. All useful capabilities must be developed through mutation.

4.4.2. Strong Attacker Seed. The strong attacker solution has all ten defender machines as targets, one recon technique for each possible effect, and an exploit with the highest possible multiplier for the current configuration. All techniques have a success rate of 100 percent. While not a perfect solution, this provides the attacker with a powerful set of starting capabilities.

4.4.3. Weak Defender Seed. The weak defender solution starts both shutdown thresholds at 1.0 (the highest level), which means that the defender will wait the longest possible time before shutting down machines despite detecting exploits and data exfiltration many times. Only three out of ten machines are listed as suspected targets, and there are no detection systems or dynamic mitigation techniques. Like the weak attacker seed, all useful capabilities must be developed through mutation.

4.4.4. Strong Defender Seed. The strong defender solution starts both shutdown thresholds at 0.5, which is meant to strike a balance between increasing security and minimizing productivity losses from having to shut down machines. It also has a cheap detection system with perfect detection rates against all recon techniques and exploits, and it has an even cheaper mitigation technique with a success rate of 100 percent. All ten machines are listed as suspected targets. This provides a nearly perfect defense.

5. RESULTS

Typical CIAO plots for each experiment are shown in Tables 1.4 and 1.5. The reason for having CIAO plots with both the attacker and defender perspectives is because each population uses a slightly different fitness function (as explained in Subsection 3.2).

5.1. Entirely White or Black CIAO Plots. CIAO plots are entirely white or black for ten experiments, indicating that little or no evolution occurred. The following is an examination of these cases.

5.1.1. X1, X5, X9, and X13. These are experiments where both populations were static (no evolution was allowed to occur), so this behavior is entirely expected.

5.1.2. X2, X6, and X14. These are experiments with a static attacker and dynamic defender. In X2, both sides start with weak population seeds. Since the weak attacker essentially starts out with zero attack capabilities, there is no motivation for the defender to evolve. In X6, the defender is strong while the attacker is still weak, so the same thing happens. In X14, both sides start out strong, and this leads to a different problem: If both sides are currently near their peak capabilities, there is little room to improve via evolution.

5.1.3. X7 and X15. These are experiments with a dynamic attacker and static defender. In X7, the attacker starts out weak while the defender starts out strong. While this provides plenty of room for the attacker to evolve, the defender was simply too powerful. The attacker never managed to evolve stronger capabilities since it could never gain a foothold in the defender's network. In X15, both sides start out strong, so neither has room to evolve past their peak capabilities.

5.1.4. X16. In this experiment, both populations were dynamic and started out strong. However, since both started near their peak, there was no room to evolve for their best members.

5.2. Gradient CIAO Plots. CIAO plots have gradients for six experiments, indicating that evolution occurred for one or both sides. The following is an examination of these cases.

5.2.1. X3 and X11. These are experiments with a dynamic attacker and static defender. In X3, both sides start out weak, which allows the attacker to evolve many new

Table 1.4. Typical Resulting CIAO Plots (Attacker Perspective)

X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

Table 1.5. Typical Resulting CIAO Plots (Defender Perspective)

X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

capabilities and produce a smooth gradient in the corresponding CIAO plots. In X11, attackers started out strong while defenders started out weak. As a result, the attacker's best members only evolved a few times since they were already strong. This created CIAO plots with a few large bands in the gradient. In both cases, these CIAO plots show clear evolution by the attacker.

5.2.2. X10. In this experiment, the defender was dynamic and started out weak while the attacker was static and started out strong. Because of the attacker's strength, the defender had an incentive to evolve better capabilities, and since the defender started out weak, it had a lot of room to evolve. Also, the attacker being static made it easier for the defender to improve since it was not chasing a moving target. This led to a smooth gradient

in the corresponding CIAO plots representing the clearest evolution by the defender in all experiments.

5.2.3. X4, X8, and X12. These are experiments where both populations were dynamic. In X4, both populations started out weak, which leads to a lot of evolution on both sides. This experiment's CIAO plots show the attacker doing better against earlier defenders, whereas the later defenders eventually became strong enough that the attackers had trouble keeping up.

In X8, the attacker started out weak while the defender started out strong. The attacker managed to evolve enough to do better against earlier opponents, though against the defenders compensated for this by the end of the CoEA. Interestingly, the defender's CIAO plot in X8 is mostly black since the defender was doing well nearly the entire time according to its own fitness estimates.

In X12, the attacker started out strong while the defender started out weak. In the attacker's CIAO plot, it is clear that the attacker was getting weaker over time according to its own fitness estimates. This is because it started out strong while the defender was weak, meaning that the attacker did not have much room to evolve and was also chasing a moving target. The defender's CIAO plot for X12 has mixed gradients, which means that the attacker was at least attempting to counter the defender's evolution. However, it still shows that the defender eventually evolved to do well against almost all the attackers.

5.3. Outlier CIAO Plots. There were a few outlier CIAO plots for experiments X14 and X16, and examples of them are displayed in Table 1.6. In both experiments, each side starts out with a strong population seed, though the attacker is static for X14 while the defender is dynamic in both experiments. The CoEA starts out with the attackers doing decently well early on, but then they are clearly dominated by all later defender solutions. This is consistent with other CIAO plots for X14 and X16 in the sense that little evolution

seems to occur, at least after a certain point. With little room for improvement, it appears that the defender will always dominate due to a nearly perfect defense.

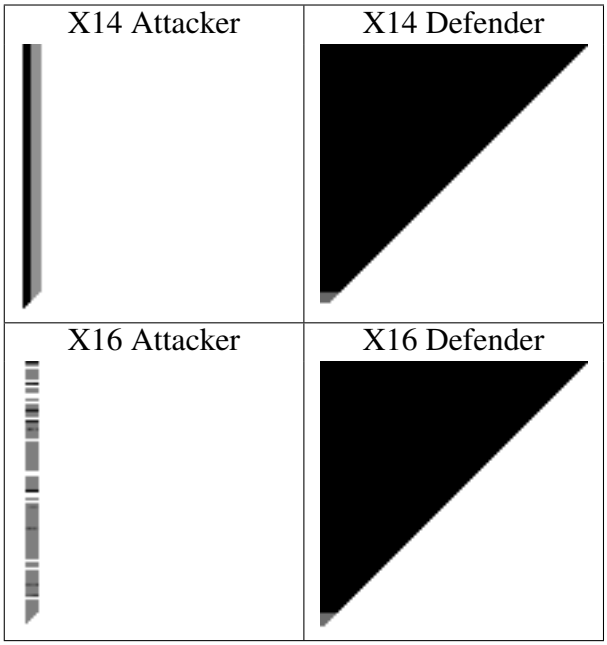
6. DISCUSSION

The results show that coevolution can be used to explore strategies by both attackers and defenders in network security, but there are still several opportunities for improvement. First, we chose to develop our own network security simulation as a compromise between a purely mathematical model and real world testing. Unfortunately, there seem to be few realistic simulations with the capabilities to test different types of cyber attacks. This is an important point since both the scenarios and strategies need to map accurately to real world entities. With our current simulation, it is possible to connect individual components in solutions to offensive and defensive capabilities in modern networks, but it will likely be an inaccurate or inconsistent mapping.

Second, the attacker and defender solutions mostly contained capabilities and a few important thresholds. The fundamental strategy for both sides does not change according to the solutions. Attackers always attempt to maximize their own profit by attacking a specific list of targets with all reconnaissance techniques and exploits available to them. Exploits will only be ignored for a given target if reconnaissance on that target has shown it to violate the exploit's own constraints (e.g. cannot use a SQL exploit if that service does not exist on the target). Defenders only respond to attacks rather than taking independent actions, and beyond detection and mitigation techniques, their only available option for stopping attacks is to shut down machines entirely.

Third, there are a number of parameter combinations that can affect the simulation or CoEA in interesting ways. One can change the number of defender machines, the number of generations, parent and survivor selection methods, the fitness function for either

Table 1.6. Outlier CIAO Plots



side, population sizes, etc. So far only limited hand tuning has been performed, due to the high computational cost of running experiments.

7. CONCLUSION AND FUTURE WORK

The goal of this work is to demonstrate the viability of developing cyber defense strategies by applying coevolution to network security simulations. We created our own simulation to test attacker and defender capabilities since existing frameworks did not meet our needs, and the results have shown that coevolution is a capable model in this solution space. In particular, experiments X3, X7, X11, and X15 are significant since they reflect the current situation in cyber security. This is because they model a dynamic attacker and static defender, an accurate situation as a defender will typically deploy defenses once and rarely change them while attackers constantly update their capabilities.

Note that our project is designed as a proof of concept. The network simulation in CANDLES is purposely somewhat abstract for the sake of simplicity. By utilizing coevolution with a more accurate simulation, it is likely that solutions with a stronger mapping to real world systems would emerge. Therefore, in future work, the first priority is to increase the fidelity of the simulation, such as introducing the notion of time (currently the simulation only recognizes order). Another important goal would be to develop more of the general strategy and less of the capabilities for attackers and defenders, as this could lead to more dynamic behavior on both sides. Finally, an extensive sensitivity study needs to be performed for identifying high-quality CoEA and simulation parameters.

8. ACKNOWLEDGMENTS

This work was supported in part by Los Alamos National Laboratory via the Cyber Security Sciences Institute under subcontract 259565 and in part by the Missouri S&T Intelligent Systems Center.

BIBLIOGRAPHY

[1] Andrew T Phillips. Now Hear This–The Asymmetric Nature of Cyber Warfare. In *US Naval Institute*, volume 138/10/1,316, October 2012.

[2] Travis Service and Daniel Tauritz. Increasing Infrastructure Resilience Through Competitive Coevolution. *New Mathematics and Natural Computation*, 5(2):441–457, July 2009.

[3] Narain G Hingorani, Laszlo Gyugyi, and Mohamed El-Hawary. *Understanding FACTS: Concepts and Technology of Flexible AC Transmission Systems*. Wiley-IEEE Press, December 1999.

[4] Holly Arnold, David Masad, Giuliano Andrea Pagani, Johannes Schmidt, and Elena Stepanova. NetAttack: Co-Evolution of Network and Attacker. In *Proceedings of the Santa Fe Institute Complex Systems Summer School 2013*.

[5] Richard Colbaugh and Kristin Glass. Predictability-Oriented Defense Against Adaptive Adversaries. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2721–2727, 2012.

[6] Richard Colbaugh and Kristin Glass. Leveraging Sociological Models for Prediction I: Inferring Adversarial Relationships. In *2012 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 66–71. IEEE, 2012.

[7] Guanhua Yan, Ritchie Lee, Alex Kent, and David Wolpert. Towards a Bayesian Network Game Framework for Evaluating DDoS Attacks and Defense. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, pages 553–566, 2012.

[8] Justin Grana, David Wolpert, Joshua Neil, Dongping Xie, Tanmoy Bhattacharya, and Russell Bent. HMMs for Optimal Detection of Cybernet Attacks. Technical Report SFI-2014-06-022, Santa Fe Institute, June 2014.

[9] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L Rivest. FlipIt: The Game of "Stealthy Takeover". *Journal of Cryptology*, 26(4):655–713, 2013.

[10] Terry Benzel, Bob Braden, Ted Faber, Jelena Mirkovic, Steve Schwab, Karen Sollins, and John Wroclawski. Current Developments in DETER Cybersecurity Testbed Technology. In *Proceedings of the Cybersecurity Applications & Technology Conference For Homeland Security (CATCH)*, pages 57–70. IEEE, 2009.

[11] Lori Pridmore, Patrick Lardieri, and Robert Hollister. National Cyber Range (NCR) Automated Test Tools: Implications and Application to Network-centric Support Tools. In *Proceedings of the 2010 IEEE Systems Readiness Technology Conference (AUTOTESTCON)*, pages 1–4, September 2010.

[12] Dave Cliff and Geoffrey F Miller. Tracking the Red Queen: Measurements of Adaptive Progress in Co-Evolutionary Simulations. In *Advances In Artificial Life*, pages 200–218. Springer, 1995.

III. CANDLES V2: A FRAMEWORK FOR COEVOLVING ATTACKER AND DEFENDER STRATEGIES FOR ENTERPRISE COMPUTER NETWORKS[1]

George Rush[1], Daniel R. Tauritz[1], and Alexander D. Kent[2]

[1]*Natural Computation Laboratory, Department of Computer Science, Missouri University of Science and Technology, Rolla, Missouri, U.S.A.*

[2]*Los Alamos National Laboratory, Los Alamos, New Mexico, U.S.A.*

ABSTRACT

Predicting an adversary's capabilities, intentions, and probable vectors of attack is in general a complex and arduous task. Cyber space is particularly vulnerable to unforeseen attacks, as most computer networks have a large, complex, opaque attack surface area and are therefore extremely difficult to analyze. Abstract adversarial models which capture the pertinent features needed for analysis can reduce the complexity sufficiently to make analysis feasible. Game theory allows for mathematical analysis of adversarial models; however, its scalability limitations restrict its use to simple, abstract models. Computational game theory is focused on scaling classical game theory to large, complex systems capable of modeling real-world environments; one promising approach is coevolution where each player's fitness is dependent on its adversaries. In this paper, we propose CANDLES v2, a framework designed to coevolve attacker and defender agent strategies for enterprise computer networks. We utilize a custom hill climber algorithm to determine the modality and complexity of the search space, and a coevolutionary algorithm is employed to find near-optimal strategies for both attackers and defenders. By performing both quantitative

[1]This is a significantly extended version of the authors' SECDEF2015 workshop paper [1].

and qualitative analysis of the result data, we provide a proof of concept for the applicability of coevolution in planning for, and defending against, novel attacker strategies in computer network security.

1. INTRODUCTION

Great strides are needed in the defensive tools and technologies available to cyber security practitioners, as the asymmetric nature of cyber warfare [2] puts defending practitioners at a distinct disadvantage; i.e., cyber attackers get to decide when and where to attack, without the need for physical presence providing advance notice to the cyber defenders who must scramble to quickly determine that an attack is occurring, select an appropriate defense, and execute it. In cyber security, as in many security-related fields, it can be prudent to evaluate worst-case attack scenarios. Whether facing insider threats or sophisticated adversaries outside one's networks, it is important to develop threat and attack models such that one can predict an adversary's capabilities, intentions, and probable vectors of attack. Cyber space is particularly vulnerable to unforseen attacks due to most computer networks having a large, complex, opaque attack surface area and therefore being extremely difficult to analyze. Invariably, the large number of variables involved in both offensive and defensive strategies makes an exhaustive search of all strategies infeasible. Abstract adversarial models which capture the pertinent features needed to analyze such strategies, can reduce the complexity sufficiently to make analysis feasible.

Game theory allows for mathematical analysis of adversarial models; however, its scalability limitations restrict its use to simple, abstract models. Computational game theory is focused on scaling classical game theory to large, complex systems capable of modeling real-world environments; one promising approach is coevolution where each player's fitness is dependent on its adversaries. Coevolution can be employed to explore the as-

sociated search spaces, examining strategic capabilities and estimating how each side will adapt in response to moves made by an opponent.

It makes sense to focus resources on securing systems such that attacks become less effective. There are several well-known ways to harden systems against attack, from limiting access permissions to patching software and reducing exposed network services. These are useful measures to reduce the attack surface and make it easier to defend vital assets. Yet for all the efforts taken to secure systems, intrusions still occur in many organizations, often without their knowledge [2]. Once it is acknowledged that intrusions cannot be stopped entirely, awareness, damage mitigation, and disaster recovery become important goals. In particular, knowing how an adversary could break into a network and devising possible counter-strategies is vital. In this paper, we propose CANDLES v2, a framework designed to coevolve attacker and defender agent strategies for enterprise computer networks. Our approach coevolves two populations containing attacker and defender strategies in a network defense scenario, and the fitness of strategies is determined through a custom, abstract simulation. In this way, unique, near-optimal strategies for both sides can be discovered.

In our framework, attacker strategies focus on what action to take in a given turn based upon available state information. Example actions might include discovering neighbors, connecting to other machines, and launching exploits. Example state information might include recent action history or the number of uncompromised neighbors. Defender strategies are more static, focusing on changes made to the network at the beginning of a simulation. These changes can include adding or removing network connections, adding intrusion prevention systems (IPSs), and fortifying individual machines. This models real world capabilities in that defenders tend to make changes periodically and rarely update them, but attackers take a more active approach that requires more immediate information

to be available. This design is intended to make the simulations more realistic and provide greater insight into attacker and defender behavior.

We have also implemented a custom hill climber algorithm which is used to determine the modality and complexity of the search space for attacker and defender strategies. By showing that the search space is multimodal and has high variation, we justify the use of evolutionary algorithms to explore possible solutions.

By performing both quantitative and qualitative analysis of the result data, we provide a proof of concept for the applicability of coevolution in planning for, and defending against, novel worst-case attacker strategies in computer network security, allowing an end user to develop strategies and set up test scenarios. This offers new possibilities for analyzing and reinforcing defensive capabilities against unknown threats.

2. RELATED WORK

Coevolution has been used to find better placements for Flexible AC Transmission System (FACTS) devices [3], which are used to prevent cascading blackouts in electric transmission systems [4], and also to evolve attackers and defenders for graph-based network theory models [5]. Co-optimization has also been used to generalize coevolution in such a way that it can be used with other black-box function optimization techniques [6], and we used co-optimization to construct our custom hill climber algorithm as discussed in Subsection 3.10. Defenses have been designed that can predict the behavior of adaptive adversaries using a combination of game theory and machine learning [7], and machine learning has likewise been used to predict the nature of relationships in adversarial social networks [8]. Strategies have been developed for defending against Distributed Denial-of-Service (DDoS) attacks using a Bayesian game theoretic framework [9], and hidden Markov models have been developed to detect cyber attacks in network traffic [10]. Pareto

optimization with Vulnerability Dependency Graphs (VDGs) has been used to simultaneously maximize productivity and minimize patching costs in enterprise networks [11]. A previous version of the CANDLES framework was used to coevolve strategies for attackers and defenders in computer networks, though it focused on evolution of both strategy and available capabilities rather than pure strategy [1].

Current approaches to developing computer network defense strategies largely focus on either pure game theoretic models [12, 13] or real world implementations and emulated systems [14, 15]. Emulated systems more accurately reflect real world conditions, but require a fair amount of resources and configuration knowledge. Game theoretic models are mathematically elegant, but they do not scale well to larger search spaces. Our approach falls somewhere in the middle, as our experiments employ a custom network security simulation. Coevolution allows us to approximate game theory solutions for large search spaces, thus providing scalability beyond the limits of classic game theory, and the simulation provides a more realistic strategy evaluation without the configuration difficulty of most emulated systems.

3. METHODOLOGY

The two primary components of the CANDLES framework are: (a) a network security simulation which quantifies the quality of attacker and defender solutions, thus creating a search gradient giving rise to a graduated search space, and (b) an optimization algorithm to search that graduated space of attacker and defender solutions for (sub)optimal solutions. We investigated two optimization algorithms: a hill climber for the initial exploration of the search space which revealed multimodality, thus justifying a more computationally expensive, but also more powerful, second algorithm, namely a CoEvolutionary Algorithm (CoEA) to represent the natural dynamics between attackers and defenders in cyber space.

A custom network security simulation is used to evaluate potential solutions for both the CoEA and hill climber. Attackers and defenders only change their strategies for the simulation rather than available capabilities, a purposeful constraint. The reason is that by focusing on keeping capabilities static, each side is forced to make strategic trade-offs which provide more interesting and realistic observations.

As attacker and defender strategies are used by everything else in CANDLES, those will be covered in more detail first, followed by the attacker and defender classes since they utilize strategies. Then the CoEA, hill climber, and network security simulation will be discussed.

3.1. Attacker Strategy. An attacker strategy is represented by a binary decision tree. Each internal node stores a condition which can evaluate to true or false, and based on how that condition evaluates, a path is chosen to traverse down the tree. Each leaf node has an action to be executed if it is reached. The attacker can only execute one action per turn. All conditions and actions that an attacker strategy can use are listed in Tables 1.7 and 1.8.

Any attacker strategy can be translated as fully executable Python code, and an example is shown in Fig. 1.7. One can see multiple places in this example where code is either unreachable or reaches the same action no matter how a condition evaluates. This is allowed in order to preserve genetic diversity since correcting or optimizing attacker strategies might remove code which is useful when recombined with other strategies.

3.2. Defender Strategy. A defender strategy first requires the creation of a network template. This network template includes all defender machines (nodes) and any connections between them (edges). Based on changes to this template, the defender strategy is defined by four item sets:

- Edges to add
- Edges to remove

Table 1.7. Attacker Conditions

Variable	Operators	Possible Values
Uniform Random Value From [0.0, 1.0]	<=, >	Value From [0.0, 1.0]
Number Of Uncompromised Neighbors	>	0
Size Of Discovered Network	<=, >	Value From [1, (Number Of Defender Machines)]
Previous Action	==	Any Attacker Action

Table 1.8. Attacker Actions

Action	Description
Discover Exploit	Generates a new random exploit. This is useful if the attacker has no exploits that work on a given target.
Discover Neighbors	Adds neighboring machines to the discovered network. This does not gather any information beyond the existence of adjacent nodes and edges.
Jump To Random Node	Attempt to connect to a random machine in the network. This may fail if the discovered network is outdated.
Launch Exploit	Attempts to compromise neighboring machine. This may fail if the discovered network is outdated, if an intrusion prevention system (IPS) blocks the scan, or if the target node is resistant to attack due to being fortified by the defender.
Scan Neighbor	Scan neighboring machine for reconnaissance information. Like exploits, target scans may fail if the discovered network is outdated, if an IPS blocks the scan, or if the target node is resistant to attack.
Search Local Machine	Extract value from the current machine. The amount of value extracted depends on the configuration.
Wait	Do nothing this turn. This is useful for letting defender paranoia decrease over time.

- Edges with intrusion prevention systems (IPSs)
- Machines to harden against attack

Adding and removing edges changes the network connectivity. Adding IPSs to various edges allows the defender to block exploits and scans, as well as raise paranoia

```
if 'jumpToRandomNode' in self.previousAction:
  if self.discoveredNetwork.number_of_nodes() > 13:
    if self.discoveredNetwork.number_of_nodes() <= 11:
      self.launchExploit(config)
    else:
      self.searchLocalMachine(results, config)
  else:
    if random.uniform(0.0, 1.0) > 0.815565908109849:
      self.discoverNeighbors()
    else:
      if 'launchExploit' in self.previousAction:
        self.searchLocalMachine(results, config)
      else:
        self.searchLocalMachine(results, config)
else:
  if 'discoverNeighbors' in self.previousAction:
    if 'discoverNeighbors' in self.previousAction:
      self.launchExploit(config)
    else:
      self.searchLocalMachine(results, config)
  else:
    if 'launchExploit' in self.previousAction:
      self.jumpToRandomNode()
    else:
      self.discoverNeighbors()
```

Figure 1.7. Example Attacker Strategy

depending on what is detected. Paranoia is covered further in Subsection 3.4. Hardening machines makes it less likely that attacks of any kind will succeed against specific targets.

An example defender strategy is shown in Fig. 1.8. Defender machines are represented by their numeric IDs, and edges are represented as tuples of IDs. Note that edges to remove are not listed here. Since the default network template starts with zero edges, removing edges would not make sense in this context.

33 edges to add: `(0, 0), (0, 7), (1, 8), (1, 16), (1, 17), (2, 12), (2, 14), (3, 4), (3, 8), (3, 11), (3, 13), (5, 5), (5, 9), (5, 18), (6, 9), (6, 12), (6, 13), (6, 17), (6, 18), (6, 19), (7, 15), (9, 19), (10, 10), (10, 13), (10, 14), (11, 12), (11, 15), (11, 19), (13, 15), (14, 19), (15, 15), (15, 18), (15, 19)`
2 edges with an IPS: `(0, 0), (10, 18)`
4 machines to fortify: `6, 7, 13, 15`

Figure 1.8. Example Defender Strategy

3.3. Attacker. The attacker stores state information and executes actions during a simulation based on its strategy. The attacker uses several pieces of information to accomplish this:

- Attacker strategy
- Exploits
- Discovered network
- Connection trace
- Previous action information

The attacker strategy is described in Subsection 3.1. Exploits only specify the constraints necessary for successful execution, which is important to note since all exploits have the same probability of success once their constraints are met. The discovered network represents an attacker's understanding of the defender's network, and it can change according to the results of neighbor discovery and machine scans. The discovered network may also need to be corrected periodically since the defender is capable of shutting down machines during the simulation, which can disrupt other actions by the attacker. The connection trace represents the attacker's current path through the defender network, which is useful in case a machine is shut down which is in the attacker's path. The attacker's most recent action is stored as state knowledge for attacker strategies.

3.4. Defender. The defender sets up the network before the simulation begins, so much of the defensive strategy is passive. However, the defender can shut down machines during the simulation if paranoia reaches certain thresholds. To accomplish this, the defender stores several pieces of information:

- Defender strategy
- Paranoia
- Attack history

The defender strategy is described in Subsection 3.2. Paranoia indicates how nervous the defender is based on detected scans or attacks. Attack history stores the source, destination, and type of any detected attacks.

3.5. Network Security Simulation. The network security simulation consists of two phases: preparation and active. Both are discussed below, as well as the data returned from each run of the simulation.

3.5.1. Preparation Phase. During this phase, variables are initialized and configured as necessary. The network template is copied to allow modifications, and the attacker's starting location is chosen based on the current configuration. By default, the attacker will start from a perimeter node connected to the Internet, as this is meant to simulate an attack from an external source. Default exploits are generated for the attacker, and both the attacker and defender are initialized with their respective strategies. The defender then sets up passive defenses across the network. These include adding and removing connections, adding intrusion prevention systems, and hardening machines against attack.

3.5.2. Active Phase. The active phase consists of an event loop that runs for a set number of iterations, essentially turns. There are four steps to each turn:

1. Attacker action
2. Defender paranoia checks

3. Defender paranoia decay
4. Productivity loss calculation

Each turn, the attacker will take the action determined by its current strategy (as described in Subsections 3.1 and 3.3). Next the defender will check its paranoia level, and if the machine shutdown threshold is met, it will find the most recent detected attack involving currently active machines and shut them down. This includes both the attack source, because it is compromised, and the target, to prevent further compromise. Then the defender will decay its paranoia, reducing it according to some value specified in the configuration. Note that paranoia can never drop below zero. Finally, the defender calculates productivity loss due to machines that have been shut down or disconnected from the Internet. Since only one node in the network connects to the defender's Internet service provider, all machines must be active and have a valid path to that node in order to have Internet connectivity.

3.5.3. Return Values. Two values and one object are returned from any run of the simulation:

- Total attacker profit
- Total defender cost
- Final network

The total attacker profit includes all value that the attacker extracted by searching compromised machines. In this case, extracting value from compromised machines represents the theft of data from an opponent's network. Since there are diminishing returns for searching the same machine multiple times, this encourages the attacker to compromise more machines. At the same time, compromising more machines means taking actions that make the attacker more noticeable. This can raise defender paranoia and make it more likely that compromised machines will be shut down.

The total defender cost includes all productivity losses from every turn of the simulation. A defender has an incentive to shut down compromised machines in order to prevent further compromise or data theft. However, productivity loss is the trade-off since inactive machines are useless. Also, an inactive machine cannot transfer data, so any path through it to the Internet is no longer valid. This means that shutting down one machine can potentially disconnect many others, leading to massive productivity loss for the remainder of the simulation.

The final network is the defender network along with all of its state information at the end of the simulation. This data is used to draw an image of the network, creating a visual representation of all nodes and edges. Specific colors and notation are used for relevant network features as covered in Subsection 3.6.

3.6. Network Graph Visualizations. Network graph visualizations are used to represent networks either before modification by attackers and defenders or at the end of a network security simulation to show the final result. An example network graph visualization from the end of a simulation is shown in Fig. 1.9. Several properties are used to make the graph easier to interpret, and color is one of the most important. All nodes start out blue by default, and compromised nodes are red. The perimeter node connected to the Internet is initially lime green, though since the attacker's starting node is orange and takes a higher priority, the perimeter node will typically be orange unless the configuration specifies a different starting point for the attacker. Finally, nodes which are inactive due to being shut down are black.

A few properties are useful besides color. Fortified nodes are larger at roughly twice the diameter of unfortified nodes, and edges containing an IPS are drawn bold (greater edge width).

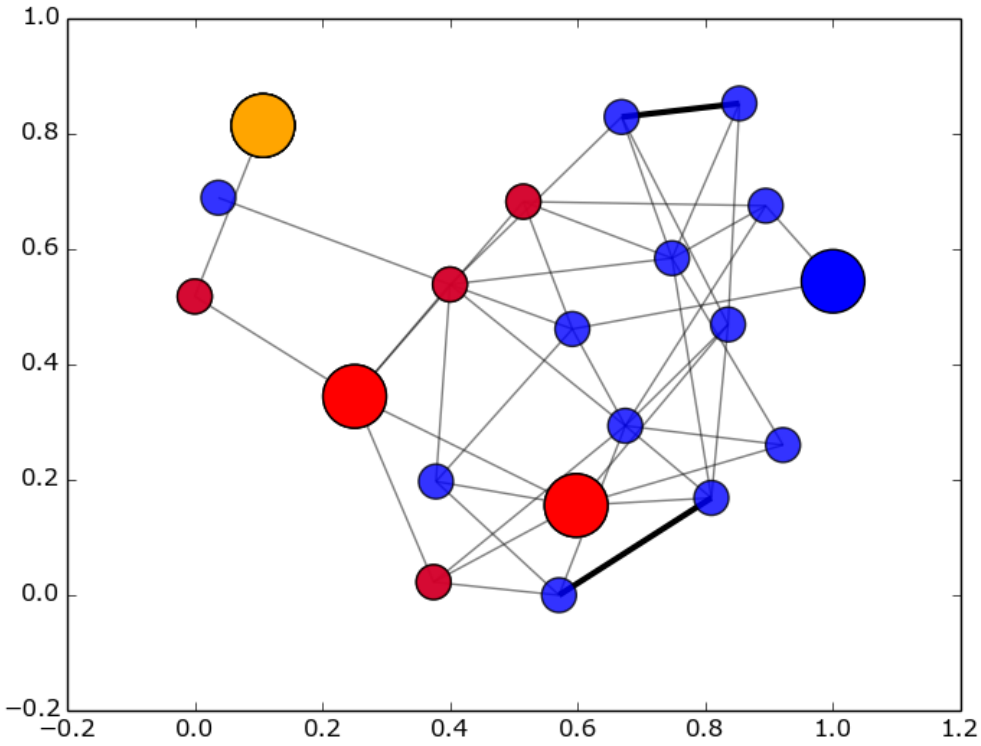


Figure 1.9. Example Network Graph Visualization

3.7. Fitness Function. The fitness function is asymmetric, with the attacker minimizing one fitness value while the defender maximizes another fitness value. Attacker profit is an abstract representation of the amount of data that attackers are able to exfiltrate after searching compromised machines. Since attacker profit is good for attackers and bad for defenders, both sides take it into account. Without parsimony pressure, attacker strategies were getting too large and difficult to analyze. Applying parsimony pressure made the resulting attacker strategies smaller and easier to understand while still allowing more effective strategies to thrive. Also, since all defenders have the same resources available, defender costs are represented by productivity loss as described in Subsection 3.5.3. The

fitness values are calculated as shown in Equations (1.5), (1.6), and (1.7).

$$attacker\ fitness = -(attacker\ profit) + (attacker\ parsimony) \tag{1.5}$$

$$attacker\ parsimony = (parsimony\ weight) * (attacker\ tree\ size) \tag{1.6}$$

$$defender\ fitness = -(attacker\ profit) - (defender\ costs) \tag{1.7}$$

Fitness is determined on a scale of $(-\infty, 0]$. Defenders attempt to maximize their fitness value since having zero losses from data theft and zero defender costs is the best case scenario. Attackers want to have as large a profit as possible, and they attempt to minimize their fitness value so that the attacker profit can be represented the same way in both equations.

3.8. Investigating the Search Space. A custom hill climber algorithm has been employed to determine the modality and complexity of the search space. The hill climber implementation is described in Subsection 3.10, and a nested version of it is described in Subsection 3.11. Results from its application are shown in Section 5 and are later discussed in Subsection 6.2. To briefly summarize the results, there is a high standard deviation in both the number of iterations and final fitness values for hill climber runs across nearly all experimental configurations. This includes testing against both dynamic and static opponents, since the search space is essentially a moving target for each side when both opponents are dynamic. From these results, the search space is determined to be multimodal with a high level of complexity. Due to this, coevolution can be used to effectively search for near-optimal solutions.

3.9. Coevolutionary Algorithm. The CoEA evolves a population of attacker strategies against a population of defender strategies using the methods described below.

3.9.1. Parent and Survivor Selection. Parent and survivor selection can be random or utilize truncation. By default, parent selection is random, and survivor selection

utilizes truncation. This ensures that the best population members are never discarded, but genetic variety is still introduced into offspring. However, these values are also manipulated as experimental variables.

3.9.2. Attacker Recombination. Attacker recombination uses subtree swapping. The subtree root nodes are selected randomly from two attacker decision trees, which means that subtrees of different sizes may be swapped. This makes it possible to grow or shrink the size of the recombined decision trees, leading to greater variety in the possible solutions. It also helps keep strategic components intact, making it more likely that useful genetic data will be introduced in the resulting offspring. Although this technique creates two resulting trees with swapped data, only one is randomly chosen and returned from the function.

3.9.3. Attacker Mutation. Attacker mutation selects a random node from the attacker decision tree and randomizes the data in that node. The randomization will not change whether the node is a condition or action since that could break the decision tree. It will instead generate a new condition or action as appropriate and assign it to the node. Note that multiple nodes may be randomized per mutation as determined by the relevant config value.

3.9.4. Defender Recombination. Defender strategies are composed of sets. To create a new offspring, two parents are chosen, and the union of each of their variables is used to create a pool of new values. From these values, a new random subset is chosen for each variable. For network connections being added and removed, the set size can change in the offspring. However, the set size is kept constant for edges to receive intrusion prevention systems and machines to harden against attack. This is because having a larger subset of IPSs and hardened machines would give offspring an unreasonable advantage.

3.9.5. Defender Mutation. To begin, one of the sets composing the defender strategy is randomly chosen for mutation. If the sets are the network connections to be added or removed, then edges may be randomly added or removed from those sets. If the sets are for intrusion prevention systems or hardened machines, then one item in the set is randomly chosen and replaced. Note that multiple changes can occur per mutation as determined by the relevant config value.

3.10. Hill Climber. Our hill climbing algorithm is a variation of first-choice hill climbing, in which successors are generated randomly until an improvement on the current state is found [16]. Our algorithm also uses co-optimization since both attackers and defenders require at least one opponent to be evaluated for fitness [6].

The hill climber begins each run by randomly generating a network template, defender strategy, and attacker strategy. Depending on the configuration, the algorithm will then attempt to improve either or both strategies until no further improvements can be made (i.e., climbing the hill). By default, both strategies are improved simultaneously.

Due to the stochastic nature of the network security simulation, it is not possible to say with certainty that one solution is statistically better than another without many computationally intensive calculations. Also, it would be infeasible to test every possible change to a specific strategy. To improve efficiency, the algorithm will only attempt to improve a solution 100 consecutive times before giving up during any iteration. However, it is possible that the attacker may improve for several iterations before the defender improves in a later iteration, or vice versa. During a run in which both sides are dynamic, it only terminates when neither side improves during an iteration. This is necessary since the fitness landscape changes with any improvement to either side, requiring the evaluation of both candidates even if only one side has improved in the previous iteration.

Likewise to improve efficiency, rather than attempt every possible change to a solution, possible improvements are found by performing random mutations using the same operators as the CoEA. This is because in some cases it would be more computationally intensive to compute all possible changes than to attempt 100 random mutations. The relevant mutation operators are described in Subsections 3.9.3 and 3.9.5.

3.11. Nested Hill Climber. The nested hill climber is an extension of our hill climbing algorithm in which 30 static opponents are tested against random individuals over 30 runs. Since the static opponents do not change, this provides a constant fitness landscape. Evaluating the search space becomes much easier as a result. It is referred to as 'nested' because the original hill climber is nested inside multiple loops to accomplish this.

4. EXPERIMENT PROCEDURE

The objective of our experiments is to use coevolution and hill climbing to explore and evaluate strategies in a cyber defense simulation. The hill climber is also used to determine the modality and complexity of the search space, justifying the use of a CoEA.

4.1. Experiment Variables. The experiment variables are configuration parameters. Several parameters are tested with multiple values for both the hill climber and CoEA algorithms, though there are a few exceptions in which a parameter only affects the CoEA. Each parameter with corresponding values is listed here, and each value has its experiment ID listed in parentheses:

- Attacker Parsimony Pressure

  0 (X1), 0.5 (X2), 1 (X3)
- Attacker Starting Node

  perimeter (X4), random (X5)

- Intrusion Prevention System (IPS) Ratio

  0.1 (X6), 0.25 (X7), 0.5 (X8)
- Machine Shutdown Threshold

  0.01 (X9), 0.1 (X10), 1.0 (X11)
- Network Topology

  linear (X12), random (X13), ring (X14), star (X15), tree (X16), and zero connectivity (X17)
- Parent Selection (CoEA Only)

  random (X18), truncation (X19)
- Productivity Loss Multiplier

  0 (X20), 0.5 (X21), 1 (X22)
- Survivor Selection (CoEA Only)

  random (X23), truncation (X24)

Only one configuration parameter is tested at a time, and all other parameters use a set of default values. The only difference between the default values for the hill climber and CoEA is that the former tests fitness against one opponent instead of five. This is because only one opponent exists during the hill climber algorithm, rather than an entire opposing population. The CoEA will instead perform random sampling against five opponents for each individual being evaluated. Also, each experiment is run 30 times due to the stochastic nature of both the algorithms and the network security simulation. This provides a sufficient sample size for statistical analysis where appropriate.

4.2. Result Data Format – CoEA. During a single run of the CoEA, the best attacker and defender strategies are stored for each generation. A network graph visualization is output once at the beginning of each run before any changes are made, and one is also output every ten generations in order to provide visual feedback of typical network

defense simulations between the best strategies. After the run is complete, all the best solutions are output to file and used to generate two CIAO plots (Current Individual vs. Ancestral Opponents), which are used to visually convey the progress of two populations during coevolution [17]. One CIAO plot shows the attacker perspective while the other shows defender perspective, an important contrast since the fitness function is asymmetric.

To briefly explain CIAO plots, an example is provided in Fig. 1.10. Assume this particular plot is from an attacker's perspective. From this perspective, darker regions indicate more success for the attacker, and the attacker's generation is plotted along the increasing y-axis while the defender's generation is plotted along the increasing x-axis. If this were the defender's perspective, darker regions would indicate more success for the defender, and the defender's generation would be plotted along the increasing y-axis while the attacker's generation would be along the increasing x-axis. In this CIAO plot, the attacker would be most effective towards the last of its generations against the earliest generations of defenders as indicated by the nearly black pixels.

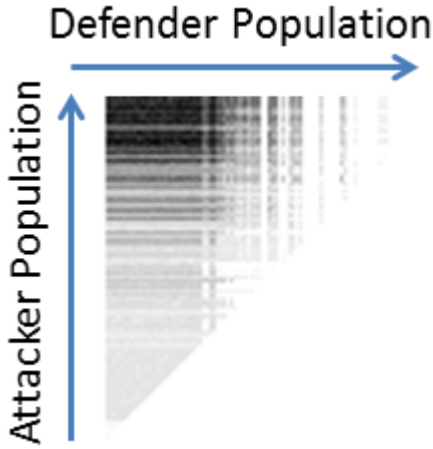


Figure 1.10. Example CIAO Plot

4.3. Result Data Format – Hill Climber. During a single run of the hill climber, the final attacker and defender strategies are output to file. Fitness values for each iteration

are also output for each strategy, which provides a way to measure the fitness values and number of iterations across one or more runs. A network graph visualization is output once at the beginning of each run before any changes are made, and one is also output at the end of each run to show the result of a typical network defense simulation between the final strategies.

5. RESULTS

This section contains results from the experiments described in Section 4. Results are organized by experimental variable and corresponding algorithm: the coevolutionary algorithm (CoEA) or the hill climber (HC). The reason that the nested hill climber results are not discussed in more detail is that they reflect similar results to the hill climber, and their main purpose is to demonstrate variability in the final fitness values of hill climber results, even when one side is kept static.

There are several result tables containing information organized by experiment ID. Tables 1.9 and 1.10 show typical CIAO plots from the attacker and defender perspectives for each CoEA experiment. Table 1.11 lists statistics for the number of iterations across all 30 runs for each hill climber experiment. Tables 1.12 and 1.13 show statistics for the final fitness values of strategies across all 30 runs for each nested hill climber experiment. Table 1.12 is specific to experiments with static attackers while Table 1.13 is specific to static defenders. However, since each nested hill climber tests 30 static opponents against random opponents, statistics are first calculated per run (for each set of 30 static opponents) and then averaged over all runs for that experiment ID.

5.1. Attacker Parsimony Pressure – CoEA (X1–X3). Attacker parsimony pressure was a configuration parameter added to reduce attacker strategies down to a more manageable size. At higher levels, the attacker did tend to evolve simpler strategies, but

Table 1.9. Typical Resulting CIAO Plots (Attacker Perspective)

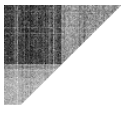
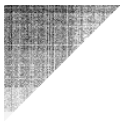
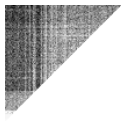
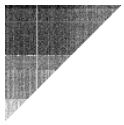
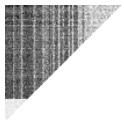
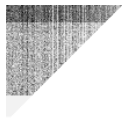
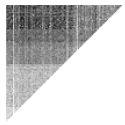
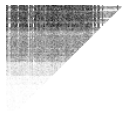
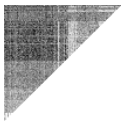
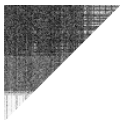
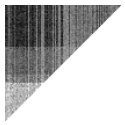
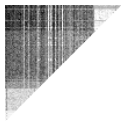
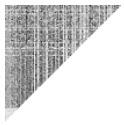
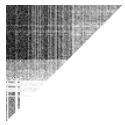
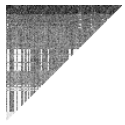
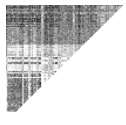
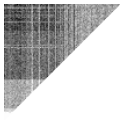
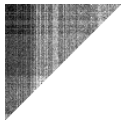
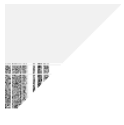
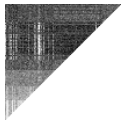
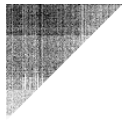
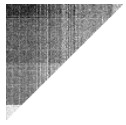
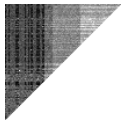
X1 	X2 	X3 	X4 	X5 	X6 	X7 	X8 
X9 	X10 	X11 	X12 	X13 	X14 	X15 	X16 
X17 	X18 	X19 	X20 	X21 	X22 	X23 	X24 

Table 1.10. Typical Resulting CIAO Plots (Defender Perspective)

X1 	X2 	X3 	X4 	X5 	X6 	X7 	X8 
X9 	X10 	X11 	X12 	X13 	X14 	X15 	X16 
X17	X18	X19	X20	X21	X22	X23	X24

there was also a noticeable effect with regards to stagnation. Particularly with zero parsimony pressure, the attacker was a bit more free to try unusual strategies, even if that meant making them larger. This meant that the attacker tended to get stuck more often when applying higher parsimony pressure, even though it made the strategies easier to understand.

Table 1.11. Hill Climber Statistics for Number of Iterations

Experiment ID	Median	Average	Standard Deviation
X1	2.0	371.7	1248.2
X2	2.0	137.5	524.3
X3	2.0	701.0	2992.1
X4	2.0	5.1	13.1
X5	3798.5	6174.5	5918.6
X6	2.0	99.0	519.2
X7	2.0	690.0	3391.9
X8	2.0	548.5	1530.8
X9	2.0	501.3	1595.6
X10	2.0	293.0	1065.9
X11	2.0	418.7	1424.5
X12	1.0	411.7	1288.0
X13	2.0	242.7	635.2
X14	1.5	356.7	1081.6
X15	2.0	221.8	713.8
X16	2.0	314.6	1436.7
X17	2.0	326.5	1166.0
X18	n/a	n/a	n/a
X19	n/a	n/a	n/a
X20	1.5	59.5	307.6
X21	2.0	1042.2	4470.7
X22	2.0	651.4	2067.9
X23	n/a	n/a	n/a
X24	n/a	n/a	n/a

5.2. Attacker Parsimony Pressure – HC (X1–X3). For the hill climber, the attacker did not perform significantly better or worse at any level of parsimony pressure. However, this is a bit unfair to test with the hill climber. Since it changes strategies using the same mutation operator as the CoEA and recombination is never used, the attacker strategy trees can never grow or shrink. As a result, parsimony pressure applies a uniform penalty to all solutions, essentially having zero effect on the algorithm. Across all result sets, the number of iterations was a median of two with a standard deviation over 500.

What this seems to indicate is a wildly variable search space in which the ability to find better solutions largely depends upon the starting points for each side. In visual terms, the fitness landscape is probably flat in several places with large jagged mountains placed intermittently. As it turns out, this search space topography persists throughout nearly all other hill climber results.

5.3. Attacker Starting Node – CoEA (X4–X5). It was difficult to determine if the attacker had an advantage when the starting node was random as opposed to starting from the perimeter (attacking from the Internet). There were several runs for both configurations where each side stagnated and made little, if any, progress. It is also interesting that both sides had more variable fitness values when the attacker's starting node was random, though it makes sense as the advantage for each side would change from simulation to simulation, making it harder to measure.

5.4. Attacker Starting Node – HC (X4–X5). This was one instance where the hill climber's results deviated strongly from the CoEA. The attacker completely dominated the network when using a random starting node, whereas starting at the perimeter rarely yielded any progress. One possible explanation is that having a random starting point allows the attacker to test its strategy in more positions and therefore adapt faster. Another possibility is that regardless of the attacker's strategy, starting from many points in the network makes it more likely that a well-positioned starting node will make the attacker appear to have a better strategy when the starting point actually made the difference. Either way, this allows the attacker to continue getting better for a longer period of time until it develops strong strategies.

Another unusual point of this experiment involved the number of iterations across results. For a starting node at the perimeter, the median number of iterations was 2.0 with a standard deviation of 13.1. This indicates that almost no useful improvements were found

Table 1.12. Nested Hill Climber Statistics for Final Fitness Values (Static Attacker)

Experiment ID	Avg Median	Avg Average	Avg Standard Deviation
X1	-6.6	-43.3	142.1
X2	-7.2	-59.4	236.3
X3	-9.0	-48.3	181.2
X4	-16.6	-58.0	169.4
X5	-10.6	-44.0	129.6
X6	-9.9	-59.2	212.7
X7	-10.1	-44.7	144.1
X8	-19.0	-58.6	159.8
X9	-17.2	-72.5	216.6
X10	-17.3	-55.1	159.0
X11	-14.1	-66.0	193.8
X12	-11.0	-17.6	34.7
X13	-7.4	-14.7	38.2
X14	-11.3	-12.1	4.3
X15	-16.7	-17.3	2.3
X16	-16.5	-16.6	4.1
X17	-3.9	-38.5	150.8
X18	n/a	n/a	n/a
X19	n/a	n/a	n/a
X20	-8.3	-8.2	0.5
X21	-6.2	-27.4	93.1
X22	-12.3	-56.7	187.9
X23	n/a	n/a	n/a
X24	n/a	n/a	n/a

for attacker solutions with a standard configuration starting from the perimeter. However, for a random starting node, the median number of iterations was 3798.5 with a standard deviation of 5918.6. This disparity between the results seems to support the notion that a random starting node enables the attacker to progress more consistently over time.

5.5. IPS Ratio – CoEA (X6–X8). The IPS ratio indicates the number of intrusion prevention systems to place in the network relative to the number of nodes. This parameter seemed to make little difference in the CoEA at any level, and there were several runs in

Table 1.13. Nested Hill Climber Statistics for Final Fitness Values (Static Defender)

Experiment ID	Avg Median	Avg Average	Avg Standard Deviation
X1	-56.3	-88.8	118.6
X2	-37.7	-63.9	99.0
X3	-39.2	-69.0	101.1
X4	-39.1	-73.5	109.1
X5	-74.9	-163.6	166.4
X6	-41.9	-71.6	99.0
X7	-41.6	-70.1	92.8
X8	-38.6	-65.4	97.8
X9	-30.1	-60.0	92.4
X10	-34.0	-68.9	109.7
X11	-40.7	-66.1	96.7
X12	-32.9	-57.1	100.6
X13	-34.0	-65.4	98.1
X14	-37.4	-70.9	112.6
X15	-40.8	-65.0	91.5
X16	-38.1	-65.5	97.0
X17	-33.5	-54.9	86.2
X18	n/a	n/a	n/a
X19	n/a	n/a	n/a
X20	-35.8	-62.4	89.6
X21	-26.1	-56.0	103.2
X22	-28.3	-52.4	89.5
X23	n/a	n/a	n/a
X24	n/a	n/a	n/a

which little or no evolution occurred. Since placing an IPS near the attacker's starting point can lead to shutting down at least the target (and therefore the network connection itself), the defender has to be careful not to cut off the rest of the network. This usually leads to the defender either placing the IPSs further from the Internet connection, or connecting the Internet to the rest of the network by multiple edges to prevent cutting off all machines and incurring massive productivity losses.

5.6. IPS Ratio – HC (X6–X8). The IPS ratio seemed to make little difference at any level, and the end result was similar to the CoEA. For all IPS ratio values, the number of iterations across result sets was a median of two and a standard deviation over 500.

5.7. Machine Shutdown Threshold – CoEA (X9–X11). Machine shutdown threshold seemed to have little effect on the CoEA. Overall it followed a similar pattern as the placement of intrusion prevention systems, probably for much the same reason. Setting the shutdown threshold too low makes it too easy to disconnect lots of machines from the network and incur productivity losses.

5.8. Machine Shutdown Threshold – HC (X9–X11). The machine shutdown threshold made little difference here, much like with the CoEA. For all machine shutdown thresholds, the number of iterations across result sets was a median of two with a standard deviation over 1000.

5.9. Network Topology – CoEA (X12–X17). For the linear topology, the defender often did the best by changing none of the edges. Since all machines are connected from the beginning, there is little point in adding new edges other than redundancy, and removing edges would create a productivity loss. Also, by having machines connected in a line, the attacker is forced to compromise each available machine consecutively in order to reach any others. Despite this, attackers were often successful in compromising several machines and exfiltrating data. This is probably because it is impossible to completely isolate the attacker without cutting off large portions of the network.

The ring topology had similar results to the linear topology, which is unsurprising considering the only difference in the ring is an extra edge from the starting point to the other end of the network.

The star topology is designed to be the worst one for the defender. From the beginning, every machine is accessible with only one hop from the starting node, and the starting

node cannot be shut down since that would disconnect the entire network. Due to this, the defender often reforms the network template in a completely different format to counter attacker efforts. Sometimes this involves isolating the starting point, and in some cases the resulting configuration ends up closer to a ring topology.

The tree topology yielded nothing particularly unusual. It is a relatively sparse network, providing a smaller attack surface area than most random topologies, but still providing more machines to attack at once than a linear or ring topology.

The zero connectivity model did surprisingly badly for the defender. While final solutions tended to connect all nodes and prevent productivity loss, the attacker either evolved well and took over most of the network or evolved barely at all. Once an attacker found a working strategy, it tended to take over everything with little resistance from the defender. This seems to indicate that it is best to start from a topology that keeps all machines connected while providing the smallest possible attack surface.

The random network topology had mixed results, as expected from an unpredictable starting configuration.

5.10. Network Topology – HC (X12–X17). Individual results for different network topologies were roughly the same for the hill climber as the CoEA. The median number of hill climber iterations ranges from one to two depending on the network topology, and the standard deviation is over 600 iterations for all result sets.

5.11. Parent Selection – CoEA (X18–X19). There was generally more evolution on both sides when parent selection was random. This is expected since survivor selection utilizes truncation by default, and parent selection has to be random in order to introduce new genetic material.

5.12. Productivity Loss Multiplier – CoEA (X20–X22). When the productivity loss multiplier was zero, there was little evolution for either side, but this was because the

defender almost immediately found a winning strategy: disconnect all nodes. This can be seen in the example CIAO plots for X20 where evolution stops halfway through the run. Once the multiplier was increased to 0.5 and 1.0 respectively, resulting networks were mostly connected. The multiplier 0.5 resulted in the best competition between both sides, probably because the lower penalty allowed the defender to experiment a bit more with changes to the network. Further changes by the defender stimulated more frequent changes in attacker strategy, which then started a virtuous cycle of greater evolution for both sides.

5.13. Productivity Loss Multiplier – HC (X20–X22). There was a significant difference between CoEA and hill climber results for a productivity loss multiplier of zero. Several of the final networks for the hill climber were connected even though there was no penalty for disconnecting all nodes. This is because while the CoEA allows both sides plenty of time to find new improvements based on a set of randomized solutions, the hill climber only has a single randomized solution per run for each side. If neither side can mutate in a way that provides an improvement to performance, the hill climber will stop after one iteration. This means that if the attacker could not improve with any single mutation, then the defender had no advantage from removing edges and therefore no reason to change anything.

The multiplier of zero also had the fewest iterations with a median of 1.5 per run and a standard deviation of 307.6. The multipliers for 0.5 and 1.0 had a median of 2.0 iterations with standard deviations of 4470.7 and 2067.9 respectively. As in other hill climber runs, this demonstrates extreme variability in the search space.

5.14. Survivor Selection – CoEA (X23–X24). Using random survivor selection led to an unusual striping effect in the CIAO plots for X23. Black, white, and gray horizontal stripes alternate as solutions of different quality are found, lost, and discovered again over multiple generations. For truncation, evolution is more consistent in the cases where

evolution occurs at all. However, there are still instances in which little to no evolution occurs due to a lack of real competition.

6. DISCUSSION

A few common patterns emerged among attacker and defender strategies. Defenders often attempt to isolate the attacker source by placing several nodes in a line between the source and the rest of the network. This forces the attacker to compromise each machine in turn to attack other parts of the network, a process called pivoting. Machines are also reinforced more near the attacker source since they are more likely to be attacked. Since the attacker is starting from the Internet, this strategy is a kind of perimeter defense. Also, when placing an intrusion prevention system near the attacker source, it is common to add multiple connections to prevent disconnecting the entire network in the case a detected attack causes machines to be shut down. This reflects the real world conundrum of how to block attacks from the Internet while still maintaining the connection for legitimate traffic.

Attacker strategies typically employed a variety of actions in order to deal with different situations. All actions in Table 1.8 appeared at least some of the time. However, one unusual result would occasionally appear in which the attacker would create a tree with only a single action: to search the local machine. Since the attacker starts with only one compromised node on any network, it was apparently easier in some cases to stay in one place and scan for data. Since compromising other machines requires multiple actions, coming up with a strategy to gain access to other machines can be somewhat tricky. Searching the local machine is also the only direct way to extract value, so it is not entirely unexpected that the attacker might choose to do this.

For attacker conditions, one of the most common was checking if the previous action matched a given value. This indicates a strong need for context, which makes sense

given that attacker actions work best when executed in a specific order. For example, one might first scan a neighboring machine before launching an exploit, or jump to a random compromised machine before searching locally for valuable data.

6.1. Coevolutionary Algorithm. The CoEA found several useful solutions, but it also had the issue of evolving poorly in many cases for randomly initialized solutions. Many randomly generated strategies were weak, which created little incentive for either side to get stronger. In other cases, attackers improved quickly, but defenders never found the right combination of changes to mitigate attacks. In general, needing fewer changes made it easier for the defender to do well. This seems to indicate that the choice of network template is important to the defender's evolution and is especially apparent in the network topology experiments, since defenders made the fewest changes to the best starting topologies: linear and ring. Those topologies tended to have some of the best evolution since defenders were starting from a stronger position which made improvements easier to find, and attackers had reason to develop better strategies against inherently stronger opponents.

6.2. Hill Climber. Overall the hill climber had a high standard deviation for the number of iterations across all result sets. This indicates that the fitness landscape is flat in many places with jagged mountains in between. Points in flat spaces represent strategies that are essentially useless and require many changes to become helpful at all. For defenders this could mean changing multiple edges, IPSs, or fortified machines, where attackers may have several actions or conditions in their decision trees that need to be modified in order to gain an advantage. In contrast, a point on a mountain or hill represents a decent strategy that just needs to be optimized.

This process is further complicated by having a fitness landscape that changes depending on the opponent. A strategy that is useful against one opponent may be ineffective or even counterproductive against another opponent. This means that the fitness landscape

itself shifts as the opponent changes, making a hill climber ineffective for testing robustness of strategies against multiple opponents. A CoEA is better suited for that type of analysis since it evolves a set of strategies for each side over multiple generations.

7. LIMITATIONS

CIAO plots are used to visually convey the progress of two populations during coevolution [17]. While this is useful for quick visual analysis, comparison of two or more CIAO plots is difficult from a statistical perspective. In particular, comparing the rate of coevolution from different runs is difficult without examining all values, and as such, it would be worthwhile to find methods of performing quantitative analysis on experimental data using coevolution.

Experiments were also constrained by computational complexity due to the stochastic nature of the network defense simulation. Introducing randomness made it possible to simulate a probability of success for any action without understanding its underlying mechanism, but it also made it more difficult to accurately measure strategy fitness since it could change from one simulation to another. The CoEA performed random sampling against five opponents for each individual being evaluated, and results from each opponent were averaged over five simulations. This meant that a single individual took 25 simulations to evaluate, and it had to be evaluated multiple times since any change in one population could change the fitness values for the other population. This raises the question of how accurate a fitness estimate has to be in order to be useful. Reducing the number of required simulations is desirable since it reduces experiment run time, but increasing the number of simulations generates a clearer CIAO plot with less noise.

The matter is further complicated when considering outliers. Sometimes the extreme cases are more interesting from a security point of view, since they represent either

the worst or best case scenarios. Worst case scenarios might benefit from further evaluation within our framework, though it is unclear how that would be accomplished without involving human expertise.

The fidelity of the network representation is also important to consider when running these simulations. The network was represented by a graph and did not attempt to simulate application protocols or even traffic rates across various edges. Instead it mostly served to keep track of available connections and intrusion prevention systems, as well as the status of defender machines. This abstraction kept the focus on features that were essential to the analysis and reduced the complexity of implementation, but it did mean losing the detail and accuracy obtained by emulating network traffic in a realistic way.

8. CONCLUSION AND FUTURE WORK

This work has shown certain existing strategies in network security to be highly effective. For defenders, perimeter security should be strong, and making pivoting more difficult will mitigate the amount of damage that can be done in a given time span. For attackers, it is best to have a diverse set of capabilities since both the opponent's network topology and machine characteristics can be diverse and unpredictable. Stealth is extremely useful when possible since it avoids raising defender paranoia, making it easier to further infiltrate the network and exfiltrate data.

No entirely new strategies were found using our simulations, though it is difficult to say whether that is due to the design of the simulation or the nature of network security. Adding more capabilities to both sides could lead to more interesting observations, if not new strategies. As such, future work might include adding the capability for defenders to deploy honeypots or host-based firewalls, as well as potentially simulating the behavior of end users in an organization. For the attacker, it might make sense to test a scenario starting

outside the network. Rather than starting with a single compromised machine and breaking through internal defenses, perhaps an attacker could sneak in via a phishing attempt or insider attack. Increasing the realism of both sides' capabilities and tradeoffs might also be helpful, such as allowing defenders to fortify more machines with a fixed cost or allowing an attacker to encrypt network traffic to avoid detection.

Regarding our implemention of the coevolutionary algorithm, future work might include switching to a multiobjective model. By doing this, one could more effectively search for the best tradeoff between defense costs and attack mitigation for defenders. A Pareto front of the best solutions could be established, and defenders could choose the best solution from that front to fit their needs.

Other future work might include performing a complexity study to determine whether the runtime of CANDLES is linear, superlinear, or sublinear depending on the size of the network. While our network only had twenty nodes, it would be interesting to measure the results when hundreds or even thousands of nodes are deployed in the defender's network. The analysis could be instructive when building other simulations or attempting to apply CANDLES to a large enterprise network.

9. ACKNOWLEDGMENTS

This work was supported in part by Los Alamos National Laboratory via the Cyber Security Sciences Institute under subcontract 259565 and in part by the Missouri S&T Intelligent Systems Center.

BIBLIOGRAPHY

[1] George Rush, Daniel R. Tauritz, and Alexander D. Kent. Coevolutionary Agent-based Network Defense Lightweight Event System (CANDLES). In *Proceedings of the Companion Publication of the 2015 Genetic and Evolutionary Computation Conference*, GECCO Companion '15, pages 859–866, New York, NY, USA, 2015. ACM.

[2] Andrew T Phillips. Now Hear This–The Asymmetric Nature of Cyber Warfare. In *US Naval Institute Proceedings Magazine*, volume 138/10/1,316, October 2012.

[3] Travis Service and Daniel Tauritz. Increasing Infrastructure Resilience Through Competitive Coevolution. *New Mathematics and Natural Computation*, 5(2):441–457, July 2009.

[4] Narain G Hingorani, Laszlo Gyugyi, and Mohamed El-Hawary. *Understanding FACTS: Concepts and Technology of Flexible AC Transmission Systems*. Wiley-IEEE Press, December 1999.

[5] Holly Arnold, David Masad, Giuliano Andrea Pagani, Johannes Schmidt, and Elena Stepanova. NetAttack: Co-Evolution of Network and Attacker. In *Proceedings of the Santa Fe Institute Complex Systems Summer School 2013*.

[6] Travis C. Service and Daniel R. Tauritz. Co-optimization algorithms. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 387–388, New York, NY, USA, 2008. ACM.

[7] Richard Colbaugh and Kristin Glass. Predictability-Oriented Defense Against Adaptive Adversaries. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2721–2727, 2012.

[8] Richard Colbaugh and Kristin Glass. Leveraging Sociological Models for Prediction I: Inferring Adversarial Relationships. In *2012 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 66–71. IEEE, 2012.

[9] Guanhua Yan, Ritchie Lee, Alex Kent, and David Wolpert. Towards a Bayesian Network Game Framework for Evaluating DDoS Attacks and Defense. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, pages 553–566, 2012.

[10] Justin Grana, David Wolpert, Joshua Neil, Dongping Xie, Tanmoy Bhattacharya, and Russell Bent. HMMs for Optimal Detection of Cybernet Attacks. Technical Report SFI-2014-06-022, Santa Fe Institute, June 2014.

[11] Edoardo Serra, Sushil Jajodia, Andrea Pugliese, Antonino Rullo, and V. S. Subrahmanian. Pareto-Optimal Adversarial Defense of Enterprise Systems. *ACM Trans. Inf. Syst. Secur.*, 17(3):11:1–11:39, March 2015.

[12] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L Rivest. FlipIt: The Game of "Stealthy Takeover". *Journal of Cryptology*, 26(4):655–713, 2013.

[13] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Qishi Wu. A Survey of Game Theory as Applied to Network Security. In *43rd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, Jan 2010.

[14] Terry Benzel, Bob Braden, Ted Faber, Jelena Mirkovic, Steve Schwab, Karen Sollins, and John Wroclawski. Current Developments in DETER Cybersecurity Testbed Technology. In *Proceedings of the Cybersecurity Applications & Technology Conference For Homeland Security (CATCH)*, pages 57–70. IEEE, 2009.

[15] Lori Pridmore, Patrick Lardieri, and Robert Hollister. National Cyber Range (NCR) Automated Test Tools: Implications and Application to Network-centric Support Tools. In *Proceedings of the 2010 IEEE Systems Readiness Technology Conference (AUTOTESTCON)*, pages 1–4, September 2010.

[16] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 2009.

[17] Dave Cliff and Geoffrey F Miller. Tracking the Red Queen: Measurements of Adaptive Progress in Co-Evolutionary Simulations. In *Advances In Artificial Life*, pages 200–218. Springer, 1995.

SECTION

2. CONCLUSIONS

In this thesis, two cyber security research frameworks are presented that apply computational intelligence techniques to solve complex problems in network defense scenarios. The first paper describes DCAFE, a framework that makes it possible to automate and control experiments in a distributed environment and allow for more realistic experiments to be carried out. DCAFE facilitated experiments that optimized Windows logging configurations to detect indicators of compromise, a form of automated system configuration. The second and third papers describe the ongoing development of CANDLES, a coevolutionary network defense simulation that explores possible attacker and defender strategies while abstracting away unnecessary details. This system gave us the capability to determine how adversaries on both sides would respond in various scenarios, creating a predictive capability.

Work like this will continue to make network security systems more autonomous and adaptive in the face of evolving adversaries. While it is impossible to know the future, predictive analysis will make it possible to change defensive strategy as necessary depending on expected attack vectors. Ever more sophisticated tools will appear for developing, testing, deploying, and maintaining both networks and the software infrastructure needed to protect them. Computational intelligence has a key role to play here since it can flexibly respond to unexpected changes in a way that traditional algorithms cannot. By placing more of the intelligence into the software itself, it enables non-experts to use tools that are otherwise unavailable to them, and experts can better utilize their time by focusing more

on the problem and less on configuring the tools. Over time, it may also give defenders the advantage they need to protect their networks against constantly changing adversaries.

VITA

George Rush was born on July 7th 1988 and grew up in southwest Missouri. He graduated from Fair Grove High School in Spring of 2006. From Summer of 2006 to Spring of 2010, he attended Missouri University of Science and Technology to earn a Bachelor of Science degree in Computer Science with a Minor in Applied Mathematics. He then worked in industry at AT&T Services, Inc. and Exegy, Inc. as a web developer before returning to Missouri S&T in Summer of 2013. While at the university, he performed research in collaboration with the Los Alamos National Laboratory that resulted in the three papers which are the foundation of this thesis. He earned his Master of Science degree in Computer Science from Missouri University of Science and Technology in December of 2015.