Propagative Problem-Solving Architecture for Expanded Task Management and Cyberwarfare

Allan Millar
allanlm05@gmail.com
Cyber-Physical Systems
Institute for Cyber-Security Education and Research
Fargo, ND, 58102

Abstract

This is the abstract of my paper. It must fit within the size allowed, which is about 3 inches, including section title, which is 11 point bold font. If you don't want the text in italics, simply remove the 'em' command and the curly braces which bound the abstract text. If you have em commands within an italicized abstract, the text will come out as normal (non-italicized) text.

1 Introduction

The global power struggle has changed. Where previously having more soldiers and more metal was enough, we now fight in a space where bullets don't work. Cyber Warfare has already begun to dominate the modern landscapes of power and influence. Unfortunately, America's abilities in these areas do not live up to their historical strength. Central to these issues is a lack of computational resources and specialists with the skills to utilize them. The hope of this research project is to create the foundation for a previously proposed tool that has the ability to address both of these lacks at the same time.

2 Background

2.1 Proposed System

The proposed system was first discussed by Professor Jeremy Straub in his poster presentation titled Blackboard-Based Electronic Warfare System [1]. In this presentation he discusses a theoretical distributed cyberwarfare system loosely based on the blackboard problem-solving architecture. This system would automate many cybersecurity and cyberwar techniques, ideally to such a degree that any user operating the system could do so successfully with only a minimal technical background. Isaac Burton later expands on the details of the system and attempts implementing basic automated hacking [2]. Mr. Burton's contributions largely center on more theoretical specificity, including discussions of obstacles that one would come across developing the system. The system may be described as a tool that can automatically locate, hack or otherwise gain access to, and integrate machines at the behest of the user. This system and its captured resources may then be used to process various tasks, being particularly suited to those that may be computationally intensive.

2.2 Potential Capabilities

There are many reasons to want a large amount of resources. The strongest is definitively data processing. In a world increasingly run on data, the capabilities for processing it are in high demand. Besides data, increasing the number of machines you have access to has the potential benefit of increasing the attack surface of any machines you may want to attack. In a much more direct way, having more machines means increased potential for output, with a specific example being the famous DDoS attack. Simply put, having more machines and properly utilizing them will generally mean DDoS attacks could be easier than they would otherwise be. Another meaningful benefit, though controversial in many areas, is in surveillance. Machines serve many corporations and states as their eyes and ears. When you are trying to prevent real-world attacks, or trying to capture criminals, having strong information gathering networks is incredibly important. Given all of this, it seems natural to look for more effective ways of gathering and handling resources. Automation and to some extent artificial intelligence seem to be the next step. Create programs that allow fewer people to handler larger projects.

2.3 Problem-Solving and Terminology

The blackboard architecture is a problem-solving model. This means that there is no specific implementation that is The Blackboard. While the model has very explicit origins in the HEARSAY-II Speech-Understanding System [3], the HEARSAY-II imple-

mentation does not directly translate to use in the our theoretical system. In fact, in doing further research I have come to believe that the blackboard model is unlikely to be the best problem-solving model for use in our system. The distributed nature of our system inherently changes how we implement these models. As one may notice, the system first proposed by Professor Straub is called "Blackboard-based", where Mr. Burton calls it an "Autonomous Distributed System of Systems. One may further notice that I alter this description to being a "Propagative Problem-Solving Architecture". The reason for this continued shift in title has to do with the difference between model and implementation. The original system described is largely defined by the blackboard model; however, this is not the case in terms of actually creating the software. In reality, the system implemented is so far undefined in terms of problem-solving methodology. Whether the Blackboard model should be used will be discussed further during Implementation. It is because of the ambiguity of the future of the system that I have titled it a "Problem-Solving Architecture". In a similar way, the system is inherently defined by its ability not only to utilize multiple machines, which defines distributed computing, but also by its ability to recruit or capture more machines. This is why I have titled it as "Propagative" instead of simply "Distributed" as it was before.

3 Related Works

Need to mention tims stuff

3.1 The Blackboard Model and the Contract Net Protocol

Both the blackboard problem-solving architecture and the contract net protocol are multi-agent behavioral design patterns, and they both seek to delegate work to agents as efficiently as possible, but utilize different methods to achieve this. Blackboard systems are defined by three components: the blackboard, knowledge sources, and a control component. These are best understood via the scenario from which the name of the model originates. In this scenario, there is a single blackboard, a staff member holding chalk, and a group of professors sitting and looking at the blackboard. All problems are written on the blackboard. The professors look over the blackboard at all of the pieces of the problem(s). Whenever they have something to contribute, meaning solving a problem or portion of a problem, they notify the staff member, who takes their input and adjusts the contents of the blackboard. The professors continue in this way until either the problem is solved or none are able to contribute. The professors are knowledge sources and the staff member is the control component, which can be likened to a monitor or lock in multithreading. The defining feature of this method is how the problem-solving components (the knowledge sources) take initiative to solve problems.

The contract net protocol is somewhat similar, save having only two components specified. These are the manager and the contractor. Again using the scenario from which the name is derived, imagine a construction agency. They are tasked with getting a garage built. This building project may be broken up into pieces, such as foundation, roofing, electrical, insulation, etc. The problem is the garage, except here the agency has sole access to the problem. They take these pieces of the problem and ask various contractors for plans or blue-prints on how they would solve the problem. If an insulation specialist is asked to do the roofing, they may reject the contract, otherwise if they feel they have a solution, they send back a proposal. Once all of the proposals are received the agency picks the best and gives the winner access. Here the problem-solvers may not autonomously solve the problem, and instead the decisions must be made by the control component (the agency). Even though implementations of these models can be relatively similar, they can also be vastly different. The methodology used by this system will be discussed in greater depth in Implementation.

4 Approach

4.1 Narrowing Scope

One of the first steps taken in this project was deciding on some limits and goals. At the time this project began there didn't seem to be any papers on similar software that met the criterion of the model that inspired it. Since there wasnt any examples to follow, there were also no explicit restrictions. In order to keep the software as simple as possible, and for reasons expanded upon in Concealment and Camouflage, all software is written in python, specifically using version 3.6.7, and no modules that require additional installation were used. Creating a robust system that can operate on any operating system within the given time period would also be impossible alone, so all software currently assumes it is running on Ubuntu. It may be compatible with similar Linux distributions, but it will not be compatible with Windows, Mac, and likely many other operating systems.

A robust system for capturing machines for use with the system has been left for future projects, and the system created by Mr. Burton [2] has not been implemented. Instead, virtual machines were connected to using ssh as a very simplified place-holder for the

capturing process. Another large part of the system left for future projects is distributed task management and more implicitly the required task-decomposition. These and other aspects of the theoretical system will be discussed both Implementation and Future Work.

4.2 Concealment and Camouflage

The end goal of model which inspired this software is a system that can covertly recruit or capture resources for use handling various tasks. While I am not extensively trained in techniques necessary to create truly covert software, wherever appropriate and possible, I have tried to make decisions to that end. The decision to use Python was based partially on its relative simplicity to write, allowing for faster prototyping and thereby developing, but it was also based on the fact that it comes pre-installed in ubuntu, as well as most Linux distributions. This has one specific benefit; that when on one of the machines with the relevant distributions is being controlled, the software doesn't need to be packaged or install a compiler or interpreter. This reduces the complexity, size, and footprint of the software. For this reason I have also given the delivered package abstract names.

Further thought may be necessary about managing connections, since they currently remain open for the entirety of the time they exist. It is important to consider that the benefit of not having an open connection be weighed against not being able to communicate as quickly, or constantly creating connections. In a system that only opens connections when actively communicating, frequency of opening and closing increases as tasks or messages propagate back up to the source machine. This is not an issue when all machines are continuously connected. The connected network will still have an increased volume of data sent as you move up to the source machine, but this may be more inconspicuous than the alternative. This and similar issues must take into consideration the intended size of the network, as issues that may be minor when considered on the level of a single connection are going to be compounded when the network grows, potentially causing serious issues at scale.

5 Implementation

5.1 Distributed Problem-Solving

Continue

5.2 The CapNet Model

The capture network, or CapNet consists of the source machine and all captured and integrated machines. Machines can be considered nodes like those in graph theory. Everything is run from a controller program which is the only program the user will interact with. This is located on the source node. Once a machine is captured and connected as discussed in Recruitment, the program which sets up a client to the capturer behaves as the controller, receiving commands from the machine that captured it, the parent using tree terminology, and processing them. The distributed controller is only controlled via commands sent from the capturer, and this client also handles returning processed tasks or any other relevant information. Any captured computer should have the ability to capture more computers. These connections will function similar to those before, with the capturer starting a server for those it captures, and each captured computer being a client relative to its capturer. When the user selects a command or function, a command or series of commands is sent to the server running on that computer, which then sends the relevant command(s) to all currently connected clients. See Robust Client Messaging in Future Work for more discussion on that process. The clients receive the command and depending on the command, pass it along to sub-nodes and respond to the command itself, or, in the case of those with no clients, simply respond. This process is reminiscent of post-order tree-traversals in that the children are given commands before the parent begins responding to the command themselves; however, it differs in that it is not necessary for the parent to wait for its children to complete their respective tasks before it can begin processing the task. If the "expand" command is sent, a node will pass on the command and then immediately begin attempting to capture more machines, because the task does not require input from child nodes. In contrast, if a data processing task were implemented and called, and a node broke up the task and distributed those sub-tasks, it would have to wait for it's children to finish their respective subtasks before it could fully respond to its parent node. It is in this part of the system that the problem-solving model would be implemented.

Though the network is not a definable graph in terms of nodes (machines) and edges (connections), it adheres to the same rules as any graph. It is assumed that any node will only have one parent, meaning it may be considered an n-tree. Having an unrestrained structure has the benefit of increased connectivity, but it has two important detriments. The first is the increased risk of being noticed with more connections. The second is that it the network arrangement isnt likely to compliment implementation of many problem-solving models. Creating distributed

problem-solving architectures confirms naturally to tree structures. While it would be just as possible to imbed these abstract models within an unrestricted graph, both options for incorporation lead to difficulty. If incorporated in a tree like structure and simply imbedded in the standard graph, no benefit is gained from having the not tree network. If any benefit is to be gained from not following the tree structure, the models must be adjusted to take advantage of the increased freedom. Perhaps having multiple parent nodes would allow for faster load balancing of tasks, but this requires further research. If possible, there may be potential to remove the need for tree balancing as discussed in Profile and Profile Balancing, but this is also uncertain. While software for any problem-solving model has been implemented, all discussion going forward will assume a tree structure, following as well the logical rules of such a structure.

As Seen in figure 1, there is no one network, as in many cases there are multiple machines a given machine may connect to. This is especially true of machines connected to the internet, which may potentially connect to upwards of hundreds of millions of machines regardless of location on the globe. These networks are determined organically as machines are more or less exploitable, or connected to more machines etc. This also means that running the program in a real-world scenario is nearly certain to never generate the same network twice. This becomes more true as the network expands.

5.3 Testing and Safety

Given the nature of the program connecting to many computers, many potentially vulnerable to attacks, all testing was done with virtual machines also running Ubuntu, and scanning for ports and machines at random, via a tool like nmap, was abandoned early on. All connecting has been hardcoded with the relevant virtual machines for simplicity. The system also only functions as a skeleton at the time of writing. While ideally the system would be tested via actual task distribution and management, time constraints have led to the decision to forgo implementation of task management and instead focus on successfully automating and connecting to remote machines.

5.4 Recruitment

Recruitment can be subdivided into three parts: infiltration, integration, and adjacently management. Infiltration consists of the process of using various techniques, in our case primarily technical exploitation, in order to gain access to and by extension control over machines. Integration immediately follows infiltration, and is concerned with connecting the machine to our network via software. Management is about the logic within the software that allows us to handle specific tasks or do specific things. Mr. Burton's work [2] has to do with infiltration, while Mr. Lei's work with secure transmission [5] would largely be used in management. This project can be considered as the foundation of the system as a whole, and as such it is concerned with both integration and management.

The specific method of integration can be summarized as package delivery followed by activation. The source machine has the entire system on it, most notably the controller. This system also contains everything necessary to run a captured machine, but everything that is not necessary to run a captured machine is kept separate to simplify delivery limit what is distributed. This is another important step in concealing activity and camouflaging the program. Ideally all sent software would be hidden in a relatively inconspicuous location below the home directories, and even would be obfuscated to further secure it.

When the source machine is given the command to expand, these necessary portions are sent to machines that have been successfully exploited. After delivery an initiation script is executed, which makes sure that the package is delivered successfully, and initiates the client script. The beginning of this process was done using a simple ssh connection, but this connection is no longer used once the client successfully connects, instead becoming a TCP/IP connection. Currently there is not system that guarantees delivery of the package, but this and systems like it would be necessary in a real world scenario.

5.5 Categorizing Machines

While the software does not currently scan networks for machines, basic categories for machines have been created. As machines are scanned and infiltrated or otherwise, they would be categorized to determine future action. The categories are integrated, vulnerable, secure, dangerous, and unknown. Integrated machines are those which are already within the network. Vulnerable machines are those which have a known vulnerability, meaning the network is able to capture them. Secure machines are machines that the network has no exploit for. Dangerous machines are currently only theoretical, but would be machines that pose a threat to the network either if captured or if attacked. These machines would likely be machines that are being monitored or are otherwise considered a risk to the network. The final category is a catch-all for any machines that do not fit in one of those previously stated.

The software uses these tags to decide various things.

5.6 Three States

In the original proposal which inspired this project, Professor Straub discussed the system having three modes; these being inert, network building, and active. In implementing the system, this logic would be maintained on the level of a given node, but I have arranged the options and user interactions slightly differently. All actions except for exiting the software itself have been delegated to one of three categories. The first category is network management, which is where options relating to expanding the network or profiling are located. The second category is titled activation as a placeholder. It is intended to eventually capture everything related to task management. The third category is the deactivation category. This menu has commands related to pausing tasks, extracting from machines, and general deactivation on the level of the entire network. The command system is still largely undeveloped, and remains easy to adjust or rearrange. Currently the system can only expand to predetermined machines and stop expanding, but as more functionality is added the three node states of inert, active, and network building may be considered and re-examined.

5.7 State of Developement

The system as a whole is still very undeveloped. Mr. Burtons work is limited to being a proof of concept, and is not in a state where it should be integrated into the main system. Mr. Leis work with messaging may be integrated if deemed suitable as the main mechanism for communication between machines now that it is in a state of relative completion. The program that makes up this project is a package that contains 4 folders within it and some groundwork for distribution. The first folder contains files for the source machine. The second is a local module containing methods used within the program where things must happen dynamically. The third contains json files with information for connections, but would eventually store data for processing and any other data necessary. The last folder contains scripts necessary to integrate and manage a distributed machine.

Functionality is currently limited to automatically connected to predetermined virtual machines, delivering the relevant portion of the program, and initiating a client server connection with connected machines. The server can handle a dynamic client load, receiving communications from clients and accepting inputs from the user concurrently. Expanding has not been robustly tested, and likely needs improvement/testing in terms of larger networks. It does have a fully functional if basic menu, also done with python, that should eventually be improved. The software is relatively documented, but this documentation should be removed if ever used for its intended purpose.

6 Future Work

6.1 Future Projects

The following is a non-exhaustive list of opportunities for future projects expanding on the system so far implemented.

- Automated network scanning and dynamic expansion.
  - As discussed in State of Developement, the current expansion system can successfully connect and integrate a limited number of machines, but rely on the information necessary being hard-coded to do so. A future project could design an automatic network scanning software which adds vulnerable computers to a list to be hacked, and once hacked they would be added to a list of computers to be hacked.
-
-
-
-
-
-

6.2 Profiling and Profile Balancing

The recommended next step in developing the system is improving expansion and adding profiling functionality. Improving expansion will include more thorough testing of expansion, ideally using a larger but still enclosed network. If unable to implement an actual tool that scans for the machines and adds them to the list of machines to be hacked dynamically, this process should ideally be simulated. Simulating networks without internet, relying on local area networks or otherwise is also important, though it requires adding functionality to the software, as it currently assumes internet connections, and is not equipped for other types of connections. Profiling consists of having machines examine themselves to create a profile and propagating these profiles up the network. These profiles may consist of things like available cores, operating system, locally connected computers, or anything else

that is relevant to handling the desired tasks. These profiles are stored locally, but in a heap-like fashion, the summation of these profiles are passed upwards and stored at every level for use in task distribution or tree adjusting. Profile balancing consists of using these profiles to attempt and balance the network tree.

6.3 Task Implementation

Acknowledgment

This work was supported by the U.S National Science Foundation (NSF award). Facilities and equipment used for this work were provided by the NDSU Institute for Cyber Security Education and Research. I would like to express my thanks to the faculty who offered their assistance and advice during my researching, and another special thanks to my peers for helping make the experience much more than a research paper. I would also like to thank my family and friends, for always pushing to do better, and supporting in everything I do. Finally, I would like to thank the reader for taking the time to read this paper I put so many hours into. Thank you everyone.

References

[1] J. Straub, "POSTER: Blackboard-Based Electronic Warfare System." in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 1681-1683.

[2] I. Burton, J. Straub "Autonomous Distributed Electronic Warfare System of Systems."