Augmenting Cyber Assessment through Dynamic Malware Analyzer

Ambrose Kam
Lockheed Martin Rotary Mission Systems (RMS)
Moorestown, NJ 08057
ambrose.kam@lmco.com

Michael Nance
Lockheed Martin Rotary Mission Systems (RMS)
Collegeville, PA
michael.h.nance@lmco.com

Wenke Lee
Georgia Institute of Technology
Atlanta, GA
wenke@cc.gatech.edu

Kyuhong Park
Georgia Institute of Technology
Atlanta, GA
kpark 302@gatech.edu

Burak Sahin
Georgia Institute of Technology
Atlanta, GA
buraksahin@gatech.edu

Carter Yagemann
Georgia Institute of Technology
Atlanta, GA
yagemann@gatech.edu

ABSTRACT

Malware is one of the major threats in the cyber space today. It is estimated that over 400,000 new samples[1] are being introduced to the world every day. To defend against these evolving advanced persistent threats (APTs), there is a need for an automatic, scalable, and secure analysis capability. Lockheed Martin is leveraging Georgia Tech's technical expertise in malware behavior analysis capability, and coupling it with cyber simulation to support cyber resiliency analysis and risk assessment.

This paper will describe an approach on how a dynamic malware analysis capability can help design a system with better cyber resiliency against existing and emerging advanced persistent threats (APT). In general, malware will remain dormant until it is triggered by specific computing or network conditions or events. Hence, malware analysis framework, like Georgia Tech's, needs to offer a flexible environment to evaluate a plethora of malware. Most current analysis frameworks cannot segregate the malware traffic in the test network environment; the analysts must decide to isolate, or to let the malware traffic pass through the security controls while letting the malware exhibit their behavior. The dynamic malware analyzer solution from Georgia Tech leverages both static and dynamic code analyses, symbolic execution techniques to examine the interactions between the malware and its command & control (C2) server so that malware can exhibit its intended behaviors in the native environment. By coupling this capability with a cyber simulation, a company like Lockheed Martin's cyber team can develop specific remedies more quickly against the threats given the detailed nature of the malware analysis. Additionally, with a simulation environment, the cyber team can also test and evaluate their "what-if" cyber defensive postures relative to zero-day threats that have not been launched in the real world. Ultimately, this will enhance the system survivability and resiliency.

ABOUT THE AUTHORS

Ambrose Kam is a Fellow in cyber in Lockheed Martin; with over 25 years of experience in the Department of Defense (DoD) industry, he is currently leading teams of engineers and contractors to support cyber threat analysis and resiliency assessment through modeling and simulation. He regularly gives cyber lectures at public forums and conferences including, MIT, Georgia Tech and Military Operations Research Society (MORS). Ambrose received

[1] https://www.av-test.org/en/statistics/malware/

his Asian American Engineer of the Year (AAEOY) award in 2017 for his technical contributions, leadership and community services.

Charles Johnson-Bey is the Director of Cyber Innovations in Lockheed Martin Rotary & Mission Systems. He is the recipient of the 2018 Black Engineer of the Year Award (BEYA). He has spent his entire career solving complex engineering and technology problems. He has been a professor and a researcher, earned a doctorate in electrical engineering, worked at Motorola research labs and Corning's Research and Development Center, and is in a 15-year tenure at Lockheed Martin.

Wenke Lee is a Professor and John P. Imlay Jr. Chair in the School of Computer Science, College of Computing, the Georgia Institute of Technology. He is also the Co-Director of the Institute for Information Security & Privacy at Georgia Tech. His research interests include systems and network security, applied cryptography, and machine learning. Most recently, he has focused on botnet detection and malware analysis, security of mobile systems and apps, detection and mitigation of information manipulation on the Internet, and adversarial machine learning.

Carter Yagemann is Ph.D. student in College of Computing at Georgia Tech; he is a computer scientist and cybersecurity researcher. His interests include hacking, system design, and software engineering.

Kyuhong Park is Ph.D. student in College of Computing at Georgia Tech. He studies Systems Security, Operating Systems, Programming Languages; he is also a graduate research assistant at Georgia Tech.

Burak Sahin is a Ph.D. student in School of College of Computing at Georgia Institute of Technology. His research interests are focused on malware analysis, software security and program analysis

Michael Nance is a Senior Fellow for Lockheed Martin. He oversees engineering teams in the classified design, implementation, and testing of trusted ground, airborne, and space-based systems. He has over three decades of experience in senior leadership roles, including as a CISO, focused in Cyber and RF based secure information technologies, and is an expert in computer network infrastructure and cyber space operations. Dr. Nance is a Certified Information System Security Professional (CISSP) and an Information Systems Security Management and Architecture Professional (ISSMP / ISSAP). He received his Doctorate of Science (DSc) in Information Assurance from Capitol Technical University (Capitol College). He is active in STEM and in the information technology community, including with the InfraGard organization and various related councils. Additionally, he is a Board member of several organizations and universities, including serving as an Adjunct Professor on weekends at the University of Maryland University College. You can find him on various social media and broadcast networks and systems as a public speaker for Lockheed Martin especially within the Wounded Warrior community

Augmenting Cyber Assessment through Dynamic Malware Analyzer

Ambrose Kam
Lockheed Martin Rotary Mission Systems (RMS)
Moorestown, NJ 08057
ambrose.kam@lmco.com

Michael Nance
Lockheed Martin Rotary Mission Systems (RMS)
Collegeville, PA
michael.h.nance@lmco.com

Wenke Lee
Georgia Institute of Technology
Atlanta, GA
wenke@cc.gatech.edu

Kyuhong Park
Georgia Institute of Technology
Atlanta, GA
kpark 302@gatech.edu

Burak Sahin
Georgia Institute of Technology
Atlanta, GA
buraksahin@gatech.edu

Carter Yagemann
Georgia Institute of Technology
Atlanta, GA
yagemann@gatech.edu

INTRODUCTION

Cyber defenders are always in a disadvantage position. There are more malware today than remedies to address them at the moment; and this is only to get worse as the rate in which malware is being released simply outpaces the rate of malware addressed. To understand why malware analysts are behind the curve, one needs to recognize a few challenges with respect to the malware analysis process itself. First, the effectiveness of existing malware analysis frameworks relies on specific conditions that need to be met during the analysis phase. That is, malware will only unfold their intended malicious behaviors only when the appropriate triggering conditions (computing or networking environments) are present. For instance, malware may rely on a command and control (C&C) server, which may be dead at the time of analysis; or an advanced persistent threat (APT) may target specific environments, such as industrial control systems (ICS) or Internet of things (IoT) devices, and will not trigger otherwise. In general, it is important to automatically collect and examine triggering conditions with cutting-edge analysis techniques to deal with the high volume of new malware. This automated analysis should assist human experts in precisely and efficiently understanding hidden malware behaviors. Given the current triggering conditions in modern malware, existing analysis frameworks struggle to analyze the hidden behaviors of malware.

The second challenge to malware analysis is the lack of methods that researchers could collaborate and share data. As the number and complexity of malware increase, the cost of manual analysis increases exponentially. Not only manual analysis is time consuming, but it also requires trained experts and the process itself is prone to human error. Although anti-virus (AV) companies release analysis and threat reports, the data they share is either artifacts (e.g., malware sample files) or open-ended reports that analysts cannot directly utilize for further analysis. Even when the reports are publicly released, analysts have to go through the daunting effort of reconstructing the analysis environment before delving further.

Third, existing frameworks suffer from inherent compromises between safety and data richness. For example, executing the malware in a real device without proper confinement (e.g., sandboxing) will fail to contain the attack, while monitoring or emulating the execution environment can trigger anti-analysis countermeasures (e.g., virtualization detection) in the sample. On the network side, the analysis framework cannot simply block the traffic from the malware because it might need network connectivity to unfold its malicious behavior. The analyst must decide whether to isolate or allow traffic to pass through the security perimeter to contain the attack while letting the malware exhibit its behaviors.

To overcome these limitations, a scalable, shareable and secure dynamic malware analysis framework (DMAF) was developed to support a collaborative experimentation. In the following sections, we describe the components and the properties of this framework:

- An automated system to examine and collect triggering conditions for malware using a cutting-edge analysis engine: a) Static and dynamic analysis to understand malware, and b) hybrid (symbolic and concolic execution) to automatically collect and examine triggering conditions.
- A reproducible experiment environment for malware analysis: a) Based on triggering conditions, generating masqueraded C&C server logic, and b) setting up network topology and file system requirements.
- Neutralization of the attacks in the malware using: a) binary re-writing to cut-off the core malicious logic and anti-analysis logic, b) automated and active containment of the network stack, and c) the careful isolation of the lab environment from the Internet.
- Shareable malware analysis environment: a) The entire analysis environment can be shared including not only pcap file, but also system call/API traces, and instruction traces for fine-grained analysis and b) The shared data allows analysts to efficiently resolve issues with others.
- A convenient and accessible interface between the framework and the malware analyst: a) empowering the malware analyst using automated tools, b) boosting the analysis with useful automation tools.

In this white paper, we explore the architecture of this DMAF framework, its modules and explain the benefits of this framework through a malware analysis use case.

ARCHITECTURE OVERVIEW

The DMAF framework provides an environment that transform the malware into an analyzable format so that researchers can concisely analyze the malware, and extract the result data of analysis in a shareable format. By leveraging cutting edge technologies, researchers can now investigate the core logic of the malware to find triggering conditions. With this crucial information, one can replicate a computing environment that makes the malware into thinking that it is communicating with a real Command & Control (C2) server. As a result, the malware's intended behaviors would be exhibited and studied. The Figure below shows how a malware being spoofed in a lab environment that matches the trigger conditions.

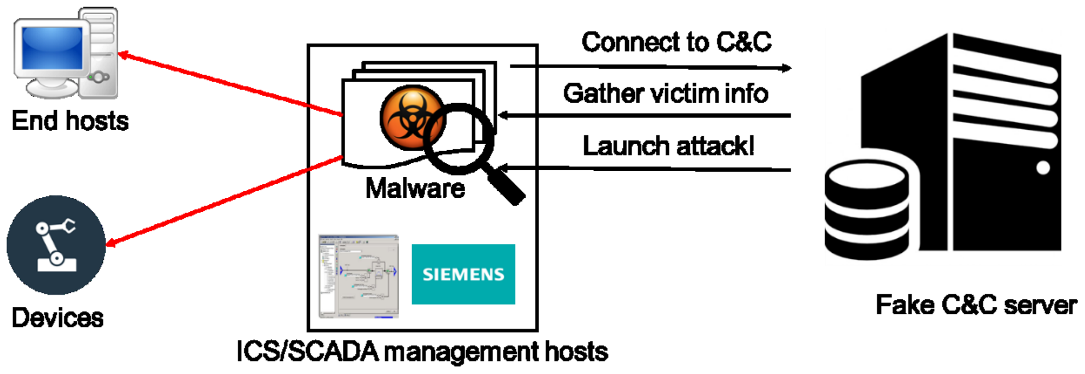


Fig. 1. A Malware Being Spoofed into Communicating with a Fake C&C Server

The DMAF consists of three main phases: preprocessing, analysis, and postprocessing. The preprocessing phase handles detection of features that thwart in-depth analysis of the malware and elimination of the obstacles to transform the given malware into an analyzable format. Also, the fuzzing feeder component automatically searches the network pcap files of the given malware in a malware analysis database, investigates the pcap files to extract C&C related packets in a best-effort manner. In the analysis phase, each component performs automated analysis to trigger conditions. As such, both static analysis and dynamic analysis generate required outputs to run the automatic symbolic execution with hybrid engines. Each component in this process iteratively interact with each other and examine possible triggering conditions. In the postprocessing phase, a secure runtime environment is constructed by leveraging the known triggering conditions from the previous steps. With this environment, malware researchers can perform repeatable, shareable experiment to concisely investigate malware behaviors, and generates data in network level, file I/O level, and even instruction-level that supported by the latest hardware feature. All data is generated in a shareable format, and the entire experiment process is repeatable—based on the verified trigger conditions. At the end of three phases, the analyst can utilize the outputs from DMAF to build reliable and highly effective protection mechanism for his/her infrastructure with intrusion detection system (IDS)/intrusion protection system (IPS) rules

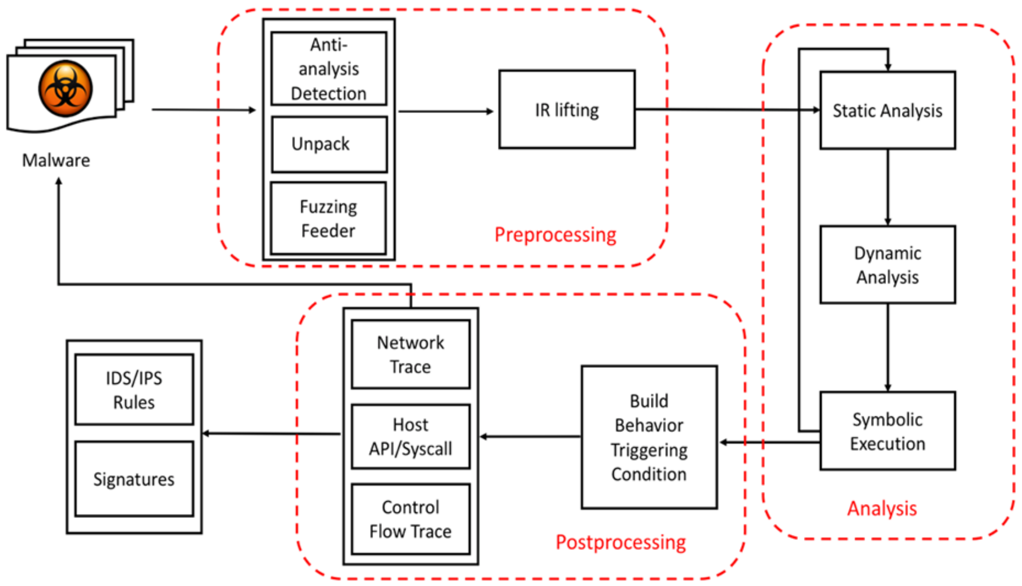


Fig. 2. Dynamic Malware Analysis Framework (DMAF)

Security is paramount in any given cyber lab environment; any possibility of contamination on the core infrastructure should be suppressed and eliminated. This proposed framework provides a compartmented environment to ensure safe handling of malware during an experiment. For example, the virtual hosts such as the virtual machine (VM) that runs malware, the virtual DNS server, and the virtual C&C servers are isolated as separate VMs. Additionally, the network is also isolated per virtual host. An autonomous firewall proxy monitors the network traffic among the virtual hosts and determines whether to allow the traffic to the Internet, block the traffic, or redirect the network traffic to internal virtual hosts to keep the malware live in the runtime environment.

PRE-PROCESSING

Over the past couple of decades, researchers have invented numerous forms of static analysis to identify malicious software. In response, malware developers obfuscate their code to make static analysis difficult and ineffective. Specifically, most malware samples rely on a technique known as *packing* to hide the malware's core functionality.

In the DMAF system, the team employs the industry best practices for "unpacking" malware samples so the code could be revealed. The team runs both specialized and generic "unpackers"; specialized unpackers are tailored made for handling commonly reused packers whereas generic packers tend to be slower and less accurate[2]. On the other hand, a generic "unpacker" typically works on all packers (although less accurate); it is also harder for malware to detect the presence of this type of "unpacker" because the monitoring and code extraction happens in a hypervisor running at a higher privilege level than the malware's sandbox.

[2] For example, if the packer never unpacks a portion of code, the generic unpacker will not recover it

The other important step in pre-processing is the packet capture (pcap) analysis. Here, the team leverages a large malware corpus to generate input for the fuzzing machine. The goal is for every malware family, to have a reference (which we refer to as "dictionary") about how the payloads of the network traffic that it receives and send, how the structure looks like, which are the stable parts (remains the same across binaries of the same family, and which are the variable parts (keep changing).

In this DMAF environment, pcaps are run on a daily basis and packets are pre-processed to extract the payloads via BRO and encrypted payload are filtered. The team would need to examine the encrypted payload for the randomness of the payload characters. For those payloads that are deemed valid, the team further processes them, to extract a "dictionary" like input to the Fuzzing Feeder.

The last step of Pre-Processing is Intermediate Representation (IR) lifting. IR is the data structure or code used internally by a compiler or virtual machine to represent source code. IR is appealing for architecture interoperability reasons; a given malware might be native to specific computing architectures, so translating the malware binary code to lower level virtual machine (LLVM) bitcode would be essential. The following figure below is the process of IR lifting for a malware sample.

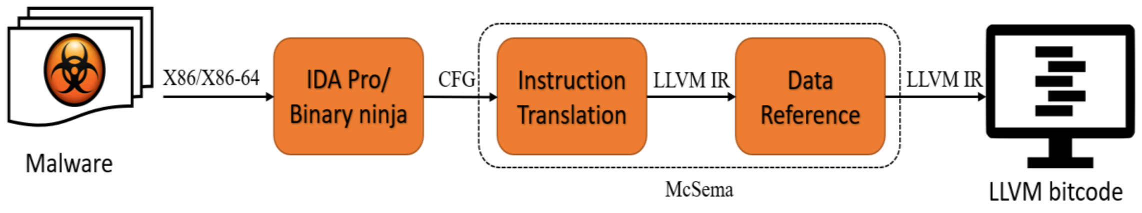


Fig. 3. Malware Code Translation via IR Process

ANALYSIS

Static analysis is a technique which gathers information about the binary without execution. Once binary code is lifted to an IR, one can use it on different architectures. In this phase, light-weight static analysis methods are applied, and all the complex analyses are handled during concolic and hybrid executions.

First, heuristics are applied to intraprocedural Control Flow Graph (CFG) to determine control flow information and data dependency information. Using that information, it is possible to create a set of pairs of possible start points and target malicious activities. For example, some candidate variables for the command inputs can be the network-related functions where the malware initiates the first dialog by reading an input from C&C server or sending a message to C&C server.

"Concolic" execution is, by definition, a combination of CONCrete and symbOLIC executions. Concrete execution generally means running a program on a specific set of inputs, and hence, it is insufficient to rely on concrete execution alone. Symbolic execution allows us to execute a program through all possible execution paths, thus achieving all possible path conditions[3]. The limitation of symbolic execution is scalability. For a simple program, symbolic execution is often preferred. However, with a large program, the number of possible paths grows exponentially.

In concrete execution, since the inputs are set to specific values, the behavior we observe is valid only for those particular values. Unlike concrete execution, concolic execution would yield a more generalized input space. Concolic execution can represent all the possible runs for a path. On the contrary, concrete execution can only produce exact values for that specific runs. While performing concolic execution, instructions are emulated, and their resulting effects reflect the symbolic environment to imitate concrete execution.

Hybrid analysis involves orchestrating static analysis, concolic execution, and dynamically instrumentation execution. In other words, the hybrid analysis method supports concolic execution while running the binaries in the real

[3] A path condition is the set of logical constraints that takes us to a specific point in the execution

environment. Concolic execution engines cannot model the entire system. Also, lifting binary into IR is not always possible. For example, for the packed binaries, we cannot use any static analysis or concolic execution techniques. Even if malware is not packed, the lifted IR may not represent the binary with 100% accuracy. The inaccuracy in lifting affects the accuracy of CFG, instruction, function boundary, etc. Moreover, malware has additional drawbacks on concolic execution. They can be packed, obfuscated or perform cryptographic computations. Modeling the environment and trying to emulate the concrete execution are not enough to overcome the drawbacks of aforementioned symbolic execution.

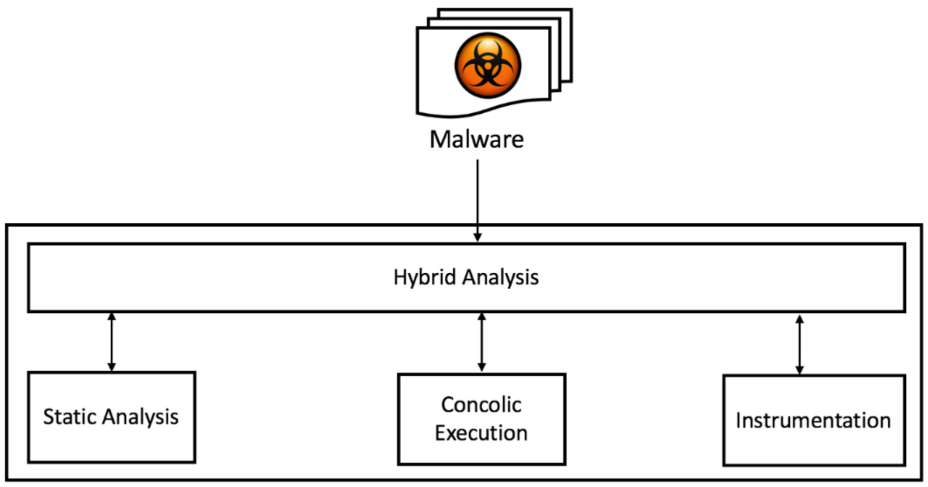


Fig. 4. DMAF Hybrid Analysis Architecture Diagram

While the malware binary is being executed, we should monitor, introspect and even change the information held by the application to manipulate the malware runtime behavior. To achieve this, some of industry Dynamic Binary Instrumentation (DBI) standard tools like DynamoRIO, Intel PIN or Frida are being used to dynamically execute malware and change their behavior. These tools provide the ability to trace instructions, memory, and CPU flags and registers which all have concrete values. As a result, we can get the information via DBI and feed these concrete values into the concolic execution engines, make only the point of interest as symbolic variables such as a network input which could carry the triggering value for a given behavior. This way, the number of the symbolic variables in the concolic execution engine can be reduced. This may reduce the path explosion and constraint solving problems. Another benefit is that we can always run heavy computational functions in concrete execution which will be faster than constraint solving. As shown in the Figure 1, the analysis module includes static analysis, dynamic analysis and symbolic execution iteratively feeding each other.

The above figure shows orchestrating analysis and execution methods. First, for any given malware, the DMAF framework tries to gather static information. Upon detecting the possible malicious paths or any pair of the start location and the target location, the information will be transferred into the concolic execution engine. In the presence of control flow information, the DMAF's concolic execution framework will use directed symbolic execution. Otherwise, the concolic execution engine will follow the path exploration technique.

Second, when the concolic execution techniques find feasible paths and cannot determine what the input would be due to lacking enough constraints, then hybrid analysis will run the malware using dynamic instrumentation tools powered by the possible set of inputs that should be fed to the running malware by the instrumentation tool.

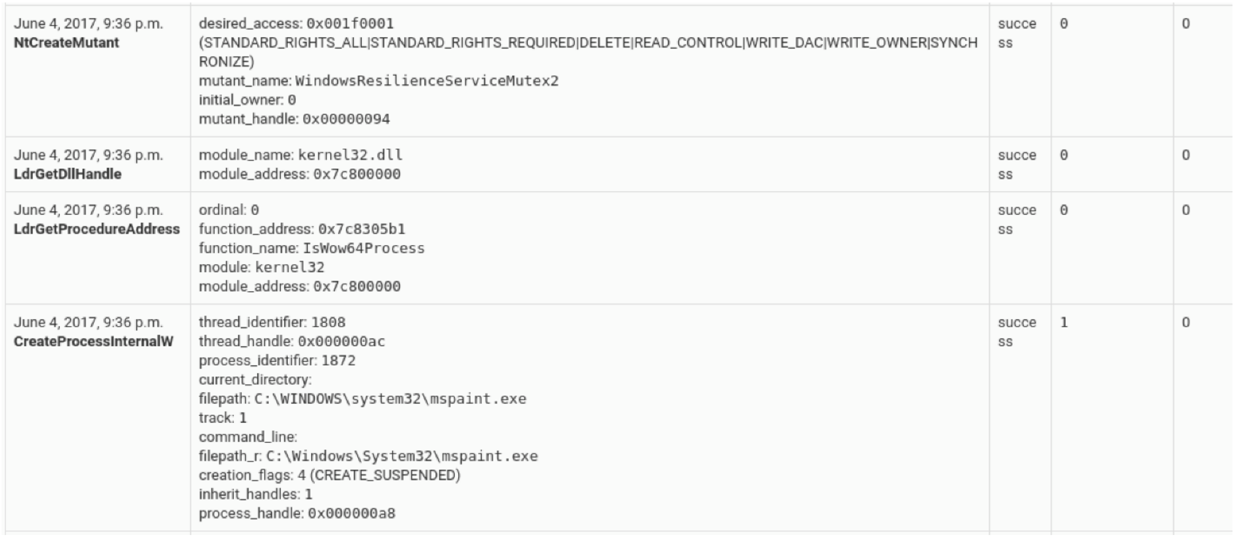
Third, an iterative execution approach is applied to the malware sample when the instrumentation tool cannot proceed further with the information gathered from concolic execution module. In this phase, the concretized memory and the context are transferred into the concrete execution environment to make symbolic space smaller. Hence, one can gain some accuracy and performance due to reduced symbolic space which makes constraint solving faster. Again, whenever the concolic execution module cannot determine a result, we follow the second and third steps iteratively. This iterative process will help both instrumentation and concolic execution. First, concrete execution doesn't have to be performed to increase code coverage. Second, concolic execution will suffer less because of reduced space of symbolic formulae and captured unsupported environment variables/functions during concrete execution

POST-PROCESSING

The primary objective of the postprocessing phase is to guarantee a secure, repeatable and shareable dynamic malware analysis while not only letting malware run live on the environment, but also feeding the malware sample with triggering conditions that are collected from the analysis phase. After the analyst performs deep dynamic malware analysis, DMAF generates execution traces in network-level and system-level in shareable format.

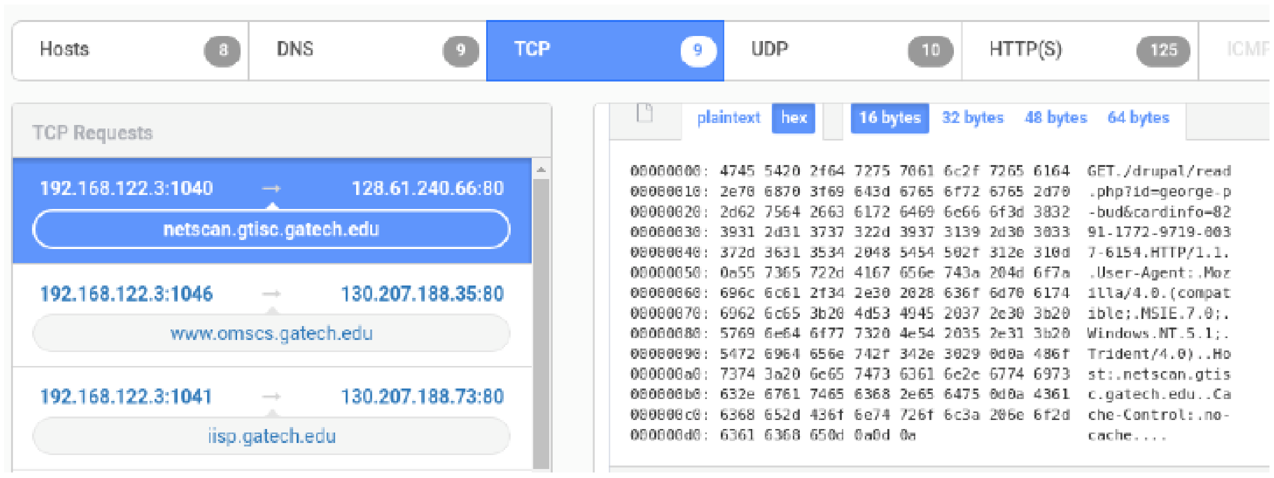
The postprocessing phase relies on the secure runtime environment using Docker, which is the latest operating-system-level virtualization and containerization technology. With triggering conditions, DMAF generates a fake C&C server, a virtual DNS server and a C&C client that is running the given malware sample. All instances are generated as virtual host by leveraging Docker image and Docker container. Docker system allows an efficient way to build virtual hosts and network topology and to ensure isolated network environment and an isolation layer on the host machine. With the Docker system, DMAF provides the secure and isolated environment aimed for repeatable malware analysis purposes.

The figures below show the results of the host and network level tracing of a given malware sample. The artifacts (including commands/API calls, host/network tracing) produced can be shared with other cyber analysts so the community can better understand a given malware and its behaviors within a computing environment. Armed with this information, cyber analysts can test different mitigation techniques to identify, suppress or eliminate the malware through experimentation. Figure 5 and Figure 6 clearly demonstrate the validity of this process and the value of DMAF.



June 4, 2017, 9:36 p.m. **NtCreateMutant**	desired_access: 0x001f0001 (STANDARD_RIGHTS_ALL|STANDARD_RIGHTS_REQUIRED|DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|SYNCHRONIZE) mutant_name: WindowsResilienceServiceMutex2 initial_owner: 0 mutant_handle: 0x00000094	success	0	0
June 4, 2017, 9:36 p.m. **LdrGetDllHandle**	module_name: kernel32.dll module_address: 0x7c800000	success	0	0
June 4, 2017, 9:36 p.m. **LdrGetProcedureAddress**	ordinal: 0 function_address: 0x7c8305b1 function_name: IsWow64Process module: kernel32 module_address: 0x7c800000	success	0	0
June 4, 2017, 9:36 p.m. **CreateProcessInternalW**	thread_identifier: 1808 thread_handle: 0x000000ac process_identifier: 1872 current_directory: filepath: C:\WINDOWS\system32\mspaint.exe track: 1 command_line: filepath_r: C:\Windows\System32\mspaint.exe creation_flags: 4 (CREATE_SUSPENDED) inherit_handles: 1 process_handle: 0x000000a8	success	1	0

Fig. 5. Host Tracing (Sample Results)



```
00000000: 4745 5420 2f64 7275 7061 6c2f 7265 6164  GET./drupal/read
00000010: 2e70 6870 3f69 643d 6765 6f72 6765 2d70  .php?id=george-p
00000020: 2d62 7564 2663 6172 6469 6e66 6f3d 3832  -bud&cardinfo=82
00000030: 3931 2d31 3737 322d 3937 3139 2d30 3033  91-1772-9719-003
00000040: 372d 3631 3534 2048 5454 502f 312e 310d  7-6154.HTTP/1.1.
00000050: 0a55 7365 722d 4167 656e 743a 204d 6f7a  .User-Agent:.Moz
00000060: 696c 6c61 2f34 2e30 2028 636f 6d70 6174  illa/4.0.(compat
00000070: 6962 6c65 3b20 4d53 4945 2037 2e30 3b20  ible;.MSIE.7.0;.
00000080: 5769 6e64 6f77 7320 4e54 2035 2e31 3b20  Windows.NT.5.1;.
00000090: 5472 6964 656e 742f 342e 3029 0d0a 486f  Trident/4.0)..Ho
000000a0: 7374 3a20 6e65 7473 6361 6e2e 6774 6973  st:.netscan.gtis
000000b0: 632e 6761 7465 6368 2e65 6475 0d0a 4361  c.gatech.edu..Ca
000000c0: 6368 652d 436f 6e74 726f 6c3a 206e 6f2d  che-Control:.no-
000000d0: 6361 6368 650d 0a0d 0a                   cache....
```

Fig. 6. Network Tracing (Sample Results)

CYBER SIMULATION

One of the main challenges in applying traditional simulation techniques to cyber is threat modeling. In general, it is not hard to model the information system network you are trying to protect. Network topology data and user/application traffic profiles should be available from the network administrators; the adversary and their chosen attack vectors might be a little more difficult to model. However, Red Teaming or Cyber Table Top (CTT) can typically provide some insights, and artifacts can then be fed into a cyber simulation environment. Threat modeling might be the longest pole in the tent since cyber threats, by nature, are very dynamic. This is especially the case for new malware. Zero Day threats are, by definition, new threats that have not been released to the wild, so there is no data on them. As a result, anti-virus and anti-malware solutions are not effective to identify and deter these Zero-Day threats as threat definition database are not populated with their information yet. Without having an analyzer framework, it is difficult to understand what the malware are doing to the information systems; hence, it would be almost impossible to model these new cyber threats and evaluate cyber defensive solutions against them.

The other challenge is security. As a policy, defense contractors are not allowed to introduce real malware to the tactical systems for fear of contamination. Hence, there is considerable difficulties to assess how tactical systems are going to perform in a contested and congested environment where cyber threats are prevalent in the operational environment. Predicting their cyber resiliency would also be near impossible if malware and their impacts are not clearly understood. This drives the need for cyber simulations and their capability to, at least, provide insights on how tactical system can perform in these challenging hostile environments. Cyber Attack Network Simulation (CANS) is an example of such tool. The objective of CANS is to enable the study of cyber events against a model of a planned or operational information system. For this malware analysis case study, CANS is used to simulate the execution and effect of cyber incidents in a modeled network environment, mimicking the behaviors of an intrusion (or an exploit) while demonstrating cyber impacts on the mission system, hardware, and software applications. The adversary effects on mission functionality are represented in the simulation environment, and remediation actions could be tested and evaluated by cyber defenders. Fig 7 below shows a high-level system architecture overview of CANS.

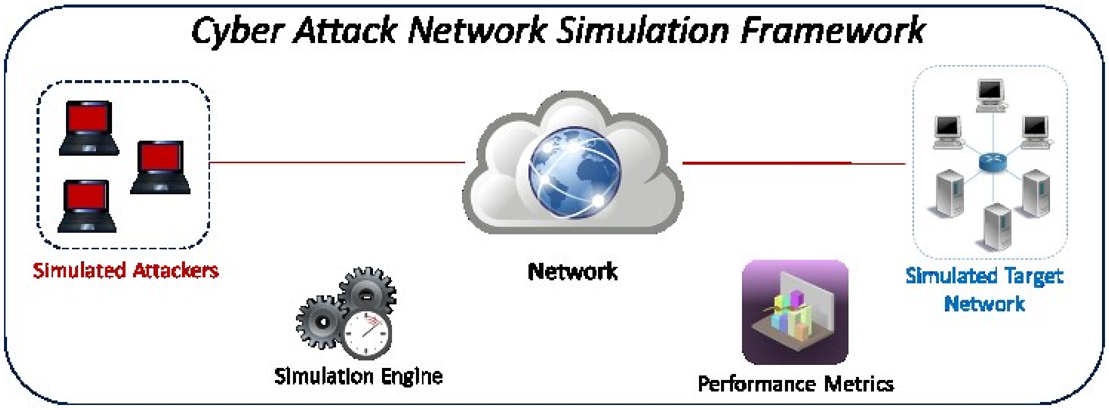


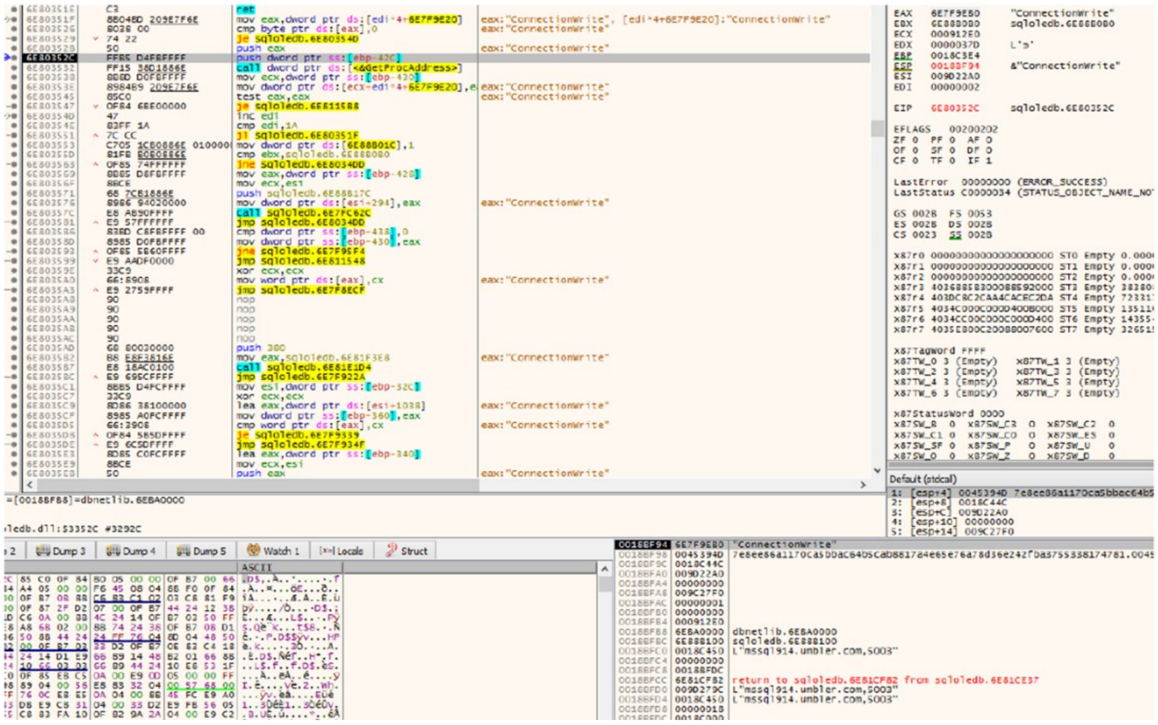
Fig. 7. CANS System Architecture Overview

To support playback and post-processing analysis, a cyber simulation needs to have attack scripting capability. An attack script might be a common vulnerability exposure (CVE) from a vulnerability database, or a malware (or malicious code) downloaded onto a network host node during a phishing attack launched by the cyber attacker. Once the CVEs or malware are defined in the attack script, the cyber simulation can be executed to produce the outcome. To support experimentation, end users can come up with the defensive strategies to mitigate and minimize the cyber risks associated with the chose attack patterns. Because each scenario run is tied to a random number, a specific iteration out of a set of Monte Carlo runs can be replicated at a later time if the run requires additional investigation.

While modeling a cyber threat through the use of CVE is useful to a cyber simulation like CANS at a high level, it is preferable to model a specific malware with great details. The aforementioned DMAF fills this void; not only a cyber simulation can now model a real-world malware without introducing live malicious source code to a tactical environment, it allows decision makers to assess cyber impacts at a mission level. Operators can be trained in realistic operational environment, so they know how to respond during a cyber event. Cyber defenders can also use this simulation environment to plan ahead, so the overall cyber resiliency level can be improved.

As a case study in 2018, the team leveraged DMAF capability and extracted a new zero-day threat so that cyber analysts could use CANS to come up with a better strategy to mitigate the effects of a malicious malware. At the time, there was a little-known malware called Mnubot[4]. The team obtained a sample of Mnubot from a threat database library and started to analyze its behaviors by using DMAF. The malware went through the unpacking and IR Lifting pre-processes outlined in Figure 2. The team then executed both static and dynamic analysis on the malware sample. Concolic execution provided insights on the Mnubot source code so that the cyber analysts can better understand the triggering conditions. For most malware, a C&C server is typically used to communicate with the malware and send specific commands to be performed. What is unique about Mnubot is that it does not interact with a C&C server directly. Instead, it relies on a Microsoft SQL server and uses it as a conduit. By wrapping malware commands in seemingly normal SQL traffic makes it hard to detect because anti-virus and anti-malware solutions simply cannot distinguish it from legitimate activities. Figure 8 & 9 below show snapshots of Mnubot and its SQL server communications captured by the aforementioned pcap and network/host tracing process. Like any other Remote Access Trojan (RAT), Mnubot needs to receive commands from the server. To do so, it has to query the Microsoft SQL database server for a new command periodically. For example, the highlighted blocks in Figure 8 shows various SQL function calls to be executed on the infected machine. As seen in Figure 9, Mnubot provides SQL server details—including server address, port, username and a password, all of which are hardcoded inside the sample. By examining the API calls and system commands, DMAF provides a detailed play-by-play description of the malware behaviors through control flow analyses. With this insight, the CAN team replicated the effects of these commands in a simulation environment by mapping them to the common vulnerability exposures (CVEs) and attack patterns. This was all done without the actual Mnubot malware being deployed to the tactical environment and risk contamination (or worse, violate any information system security policy).

[4] Discovered by IBM researchers in 2018, Mubot was one of the zero-day threats captured and identified.



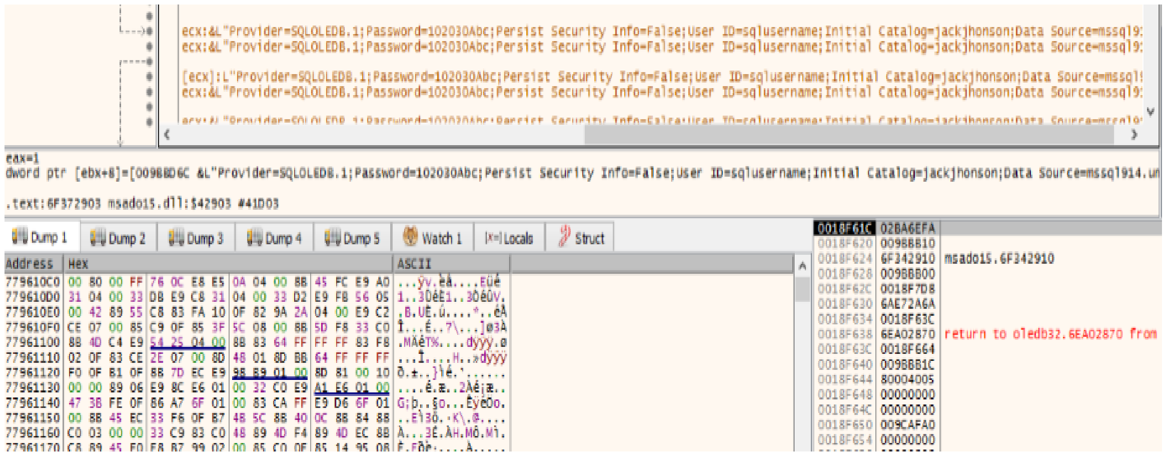
```
6E80351E   C3                  ret
6E80351F   8B04BD 209E7F6E     mov eax,dword ptr ds:[edi*4+6E7F9E20]      eax:"ConnectionWrite", [edi*4+6E7F9E20]:"ConnectionWrite"
6E803526   8038 00             cmp byte ptr ds:[eax],0                    eax:"ConnectionWrite"
6E803529   74 22               je sqloledb.6E80354D
6E80352B   50                  push eax                                   eax:"ConnectionWrite"
6E80352C   FFB5 D4FBFFFF       push dword ptr ss:[ebp-42C]
6E803532   FF15 38D1886E       call dword ptr ds:[<&GetProcAddress>]
6E803538   8B8D D0FBFFFF       mov ecx,dword ptr ss:[ebp-430]
6E80353E   898489 209E7F6E     mov dword ptr ds:[ecx+edi*4+6E7F9E20],e    eax:"ConnectionWrite"
6E803545   85C0                test eax,eax                               eax:"ConnectionWrite"
6E803547   0F84 6BE00000       je sqloledb.6E8115B8
6E80354D   47                  inc edi
6E80354E   83FF 1A             cmp edi,1A
6E803551   7C CC               jl sqloledb.6E80351F
6E803553   C705 1C80886E 010000 mov dword ptr ds:[6E88801C],1
6E80355D   81FB B0B0886E       cmp ebx,sqloledb.6E88B0B0
6E803563   0F85 74FFFFFF       jne sqloledb.6E8034DD
6E803569   8B85 D8FBFFFF       mov eax,dword ptr ss:[ebp-428]
6E80356F   8BCE                mov ecx,esi
6E803571   68 7CB1886E         push sqloledb.6E88B17C
6E803576   8986 94020000       mov dword ptr ds:[esi+294],eax             eax:"ConnectionWrite"
6E80357C   E8 AB90FFFF         call sqloledb.6E7FC62C
6E803581   E9 57FFFFFF         jmp sqloledb.6E8034DD
6E803586   83BD C8FBFFFF 00    cmp dword ptr ss:[ebp-438],0
6E80358D   8985 D0FBFFFF       mov dword ptr ss:[ebp-430],eax
6E803593   0F85 5B60FFFF       jne sqloledb.6E7F95F4
6E803599   E9 AADF0000         jmp sqloledb.6E811548
6E80359E   33C9                xor ecx,ecx
6E8035A0   66:8908             mov word ptr ds:[eax],cx                   eax:"ConnectionWrite"
6E8035A3   E9 2759FFFF         jmp sqloledb.6E7F8ECF
6E8035A8   90                  nop
6E8035A9   90                  nop
6E8035AA   90                  nop
6E8035AB   90                  nop
6E8035AC   90                  nop
6E8035AD   68 80030000         push 380
6E8035B2   B8 E8F3816E         mov eax,sqloledb.6E81F3E8                  eax:"ConnectionWrite"
6E8035B7   E8 18AC0100         call sqloledb.6E81E1D4
6E8035BC   E9 695CFFFF         jmp sqloledb.6E7F922A
6E8035C1   8BB5 D4FCFFFF       mov esi,dword ptr ss:[ebp-32C]
6E8035C7   33C9                xor ecx,ecx
6E8035C9   8D86 38100000       lea eax,dword ptr ds:[esi+1038]            eax:"ConnectionWrite"
6E8035CF   8985 A0FCFFFF       mov dword ptr ss:[ebp-360],eax
6E8035D5   66:3908             cmp word ptr ds:[eax],cx                   eax:"ConnectionWrite"
6E8035D8   0F84 5B5DFFFF       je sqloledb.6E7F9339
6E8035DE   E9 6C5DFFFF         jmp sqloledb.6E7F934F
6E8035E3   8D85 C0FCFFFF       lea eax,dword ptr ss:[ebp-340]
6E8035E9   8BCE                mov ecx,esi
6E8035EB   50                  push eax                                   eax:"ConnectionWrite"

=[0018BF88]=dbnetlib.6EBA0000

ledb.dll:$3352C #3292C
```

```
EAX    6E7F9EB0     "ConnectionWrite"
EBX    6E88B0B0     sqloledb.6E88B0B0
ECX    000912E0
EDX    0000037D     L'ɔ'
EBP    0018C3E4
ESP    0018BF94     &"ConnectionWrite"
ESI    009D22A0
EDI    00000002

EIP    6E80352C     sqloledb.6E80352C

EFLAGS    00200202
ZF 0  PF 0  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError   00000000 (ERROR_SUCCESS)
LastStatus  C0000034 (STATUS_OBJECT_NAME_NO

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

x87r0 00000000000000000000 ST0 Empty 0.000
x87r1 00000000000000000000 ST1 Empty 0.000
x87r2 00000000000000000000 ST2 Empty 0.000
x87r3 4036885B3000BB592000 ST3 Empty 38380
x87r4 403DCBC2CAA4CACEC2DA ST4 Empty 72331
x87r5 4034C000C000D4008000 ST5 Empty 13511
x87r6 4034CC00C000C000D400 ST6 Empty 14355
x87r7 4035E800C20088007600 ST7 Empty 32651

x87TagWord FFFF
x87TW_0 3 (Empty)    x87TW_1 3 (Empty)
x87TW_2 3 (Empty)    x87TW_3 3 (Empty)
x87TW_4 3 (Empty)    x87TW_5 3 (Empty)
x87TW_6 3 (Empty)    x87TW_7 3 (Empty)

x87StatusWord 0000
x87SW_B  0  x87SW_C3  0  x87SW_C2  0
x87SW_C1 0  x87SW_C0  0  x87SW_ES  0
x87SW_SF 0  x87SW_P   0  x87SW_U   0
x87SW_O  0  x87SW_Z   0  x87SW_D   0

Default (stdcall)
1: [esp+4]  0045394D 7e8ee86a1170ca5bbac64b5
2: [esp+8]  0018C44C
3: [esp+C]  009D22A0
4: [esp+10] 00000000
5: [esp+14] 009C27F0
```

```
ASCII
85 C0 0F 84 | 80 05 00 00 | 0F B7 00 66   .D$,.À..'.....f
A4 05 00 00 | F6 45 08 04 | 8B F0 0F 84   .À..¤...öE...ð..
0F B7 0B 8B | C6 83 C1 02 | 03 C8 81 F9   ïÀ.....Æ.Á..È.ù
0F 87 2F D2 | 07 00 0F B7 | 44 24 12 38   bÿ..../Ò....D$.;
C6 0A 00 8B | 4C 24 14 0F | B7 03 50 FF   Æ...L$...·.Pÿ
A8 68 02 00 | 8B 74 24 38 | 0F B7 08 D1   s.Qè K...t$8...Ñ
50 8B 44 24 | 24 FF 76 04 | 8D 04 48 50   é..P.D$$ÿv...HP
00 0F B7 D2 | 33 D2 0F B7 | 0E 83 C4 18   è.k.....3Ò....A.
24 14 D1 E9 | 66 89 14 48 | B2 01 66 8B   .È.D$.Ñéf..H².f.
10 66 03 D2 | 66 89 44 24 | 10 E8 53 1F   ..L$.f..f.D$.èS.
0F 85 EB C5 | 0A 00 E9 0D | 05 00 00 FF   ...À..eÀ..é....ÿ
89 04 00 56 | E8 83 32 04 | 00 57 68 00   I.é....Vè.2..Wh.
76 0C E8 E5 | 0A 04 00 8B | 45 FC E9 A0   ...ÿv.èå....Eüé
DB E9 C8 31 | 04 00 33 D2 | E9 FB 56 05   1..3ÒéÈ1..3ÒéûV.
C8 83 FA 10 | 0F 82 9A 2A | 04 00 E9 C2   .B.UÈ.ú....*..éÂ
```

```
0018BF94  6E7F9EB0  "ConnectionWrite"
0018BF98  0045394D  7e8ee86a1170ca5bbac64b5cab8817a4e65e76a78d36e242fba3755338174781.0045
0018BF9C  0018C44C
0018BFA0  009D22A0
0018BFA4  00000000
0018BFA8  009C27F0
0018BFAC  00000001
0018BFB0  00000000
0018BFB4  000912E0
0018BFB8  6EBA0000  dbnetlib.6EBA0000
0018BFBC  6E88B100  sqloledb.6E88B100
0018BFC0  0018C450  L"mssql914.umbler.com,5003"
0018BFC4  00000000
0018BFC8  0018BFDC
0018BFCC  6E81CF82  return to sqloledb.6E81CF82 from sqloledb.6E81CE37
0018BFD0  009D279C  L"mssql914.umbler.com,5003"
0018BFD4  0018C450  L"mssql914.umbler.com,5003"
0018BFD8  0000001B
0018BFDC  0018C000
```

Fig. 8. Mnubot Malware (Sample) SQL Commands



```
ecx:&L"Provider=SQLOLEDB.1;Password=102030Abc;Persist Security Info=False;User ID=sqlusername;Initial Catalog=jackjhonson;Data Source=mssql9
ecx:&L"Provider=SQLOLEDB.1;Password=102030Abc;Persist Security Info=False;User ID=sqlusername;Initial Catalog=jackjhonson;Data Source=mssql9

[ecx]:L"Provider=SQLOLEDB.1;Password=102030Abc;Persist Security Info=False;User ID=sqlusername;Initial Catalog=jackjhonson;Data Source=mssql9
ecx:&L"Provider=SQLOLEDB.1;Password=102030Abc;Persist Security Info=False;User ID=sqlusername;Initial Catalog=jackjhonson;Data Source=mssql9

ecx:&L"Provider=SQLOLEDB.1;Password=102030Abc;Persist Security Info=False;User ID=sqlusername;Initial Catalog=jackjhonson;Data Source=mssql9

eax=1
dword ptr [ebx+8]=[009BBD6C &L"Provider=SQLOLEDB.1;Password=102030Abc;Persist Security Info=False;User ID=sqlusername;Initial Catalog=jackjhonson;Data Source=mssql914.un

.text:6F372903 msado15.dll:$42903 #41D03
```

```
Address   Hex                                               ASCII
779610C0  00 80 00 FF 76 0C E8 E5 0A 04 00 8B 45 FC E9 A0   ...ÿv.èå....Eüé
779610D0  31 04 00 33 DB E9 C8 31 04 00 33 D2 E9 FB 56 05   1..3ÛéÈ1..3ÒéûV.
779610E0  00 42 89 55 C8 83 FA 10 0F 82 9A 2A 04 00 E9 C2   .B.UÈ.ú....*..éÂ
779610F0  CE 07 00 85 C9 0F 85 3F 5C 08 00 8B 5D F8 33 C0   Î...É..?\...]ø3À
77961100  8B 4D C4 E9 54 25 04 00 8B 83 64 FF FF FF 83 F8   .MÄéT%....dÿÿÿ.ø
77961110  02 0F 83 CE 2E 07 00 8D 48 01 8D BB 64 FF FF FF   ...Î....H..»dÿÿÿ
77961120  F0 0F B1 0F 8B 7D EC E9 98 B9 01 00 8D 81 00 10   ð.±..}ìé.¹......
77961130  00 00 89 06 E9 8C E6 01 00 32 C0 E9 A1 E6 01 00   ....é.æ..2Àé¡æ..
77961140  47 3B FE 0F 86 A7 6F 01 00 83 CA FF E9 D6 6F 01   G;þ..§o...ÊÿéÖo.
77961150  00 8B 45 EC 33 F6 0F B7 48 5C 8B 40 0C 8B 84 88   ..Eì3ö.·H\.@....
77961160  C0 03 00 00 33 C9 83 C0 48 89 4D F4 89 4D EC 8B   À...3É.ÀH.Mô.Mì.
77961170  C8 89 45 F0 E8 B7 99 02 00 85 C0 0F 85 14 95 08   È.Eðè·....À.....
```

```
0018F61C  02BA6EFA
0018F620  009BBB10
0018F624  6F342910  msado15.6F342910
0018F628  009BBB00
0018F62C  0018F7D8
0018F630  6AE72A6A
0018F634  0018F63C
0018F638  6EA02870  return to oledb32.6EA02870 from
0018F63C  0018F664
0018F640  009BBB1C
0018F644  80004005
0018F648  00000000
0018F64C  00000000
0018F650  009CAFA0
0018F654  00000000
```

Fig. 9. Commands to Access the Mubot C&C Server

Armed with this knowledge, the cyber team can now try different IDS/IPS techniques to mitigate the effects of the commands executed by this Mnubot malware while ensuring the original intended operations can still be performed by the tactical mission systems.

FUTURE WORK

This paper covers current progress in cyber modeling & simulation (M&S) to incorporate effects of the real malware threats. Our minimum viable prototype (MVP) demonstrates the value of this system by integrating and testing different cyber solutions on a potential tactical system environment. Thus far, this process is still very labor intensive as automation cannot be applied to expert system yet. The goal is to leverage Machine Learning (ML) and Deep

Learning (DL) techniques to quickly find the most optimum cyber solutions without this lengthy operations analysis process. Lockheed Martin is currently partnering with Massachusetts Institute of Technology (MIT) and experimenting the use of convolutional neural network (CNN) based Deep Learning techniques. MIT's expertise in ML and DL helps apply these cutting-edge technologies to cyber defenses. Collaboratively, the team has studied Google's TensorFlow and Facebook's PyTorch and determined a set of features that should be used to drive such optimum solution(s) in response to a given set of threats.

SUMMARY

This paper demonstrated how DMAF framework has been utilized to perform malware analysis on a real-world zero-day malware; by leveraging various cutting-edge techniques, the team has unveiled intricate malware behaviors by efficiently analyzing the malware (through static and dynamic code analysis) and its C&C server (through a SQL server). Ultimately, these behaviors are captured and eventually modeled in a cyber simulation environment for cyber risk assessment and system resiliency analysis. The progress described in this report indicates the approach is effective and robust for understanding malware. By coupling this malware analysis capability with a cyber simulation like CANS, cyber analysts can use this capability to assess cyber risks and boost cyber resiliency. Mission planning and cyber operator training activities are also getting benefits from this approach as CMF and CPT can now anticipate certain threat scenario(s) and continue their mission execution while a cyber event is taking place.

REFERENCES

[1] Barnum, S., *An Introduction to Attack Patterns as a Software Assurance Knowledge Resource,* OMG Software Assurance Workshop 2007

[2] Bodeau, D., Graubart, R.,*Characterizing Effects on the Cyber Adversary: A Vocabulary for Analysis and Assessment* MITRE Technical Report MTR130432, Bedford, MA Nov 2013

[3] Committee on Technology for Future Naval Forces, *Becoming 21st Century Force, Modeling and Simulation,* Volume 9, U.S. Naval Studies Board National Research Council, 1997

[4] Department of Defense Instruction (DoDI) 8500.2, *Information Assurance (IA) Implementation* Secretary of the Navy Instruction (SECNAVINST) 5239.3B, *Department of the Navy (DoN) Information Assurance (IA) Policy*

[5] Jaquith,A, *Security Metrics: Replacing Fear*, Uncertainty, and Doubt. ISBN 9780321349989

[6] Powell, S.,*Cyber Effects Prediction, Black Hat Briefing, DC, 2010*

[7] Ostermiller, Voula, *TI12 Network Modeling Report for BL9.C2*, Lockheed Martin MST, Nov, 2014

[8] Security Technical Implementation Guide - Application Security and Development, Defense Information Systems Agency (DISA), 23 January 2014

[9] Sergio Herrero-Lopez, S., Williams, J., Sanchez, A., *Large-Scale Simulator for Global Data Infrastructure Optimization*, 2011 IEEE International Conference on Cluster Computing, MIT, 2011

[10] Simmons, C., et. al., *AVOIDIT: A Cyber Attack Taxonomy,* Dept. of Computer Science, University of Memphis, IEEE Mag, Apr, 2010

[11] Uma, M., Padmavathi, G., *A Survey on Various Cyber Attacks and Their Classification,* International Journal of Network Security, Vol.15, No.5, PP.390-396, Sept. 2013