2011 3rd International Conference on Cyber Conflict
C. Czosseck, E. Tyugu, T. Wingfield (Eds.)
Tallinn, Estonia, 2011 © CCD COE Publications

Using a Novel Behavioral Stimuli-Response Framework to Defend against Adversarial Cyberspace Participants

Daniel Bilar
Department of Computer Science
University of New Orleans
New Orleans, LA 70148, USA
dbilar@uno.edu

Brendan Saltaformaggio
Department of Computer Science
University of New Orleans
New Orleans, LA 70148, USA
bsaltafo@uno.edu

Abstract- **Autonomous Baiting, Control and Deception of Adversarial Cyberspace Participants (ABCD-ACP) is an experimental defensive framework against potentially adversarial cyberspace participants, such as malicious software and subversive insiders. By deploying fake targets (called baits/stimuli) onto a virtualized environment, the framework seeks to probabilistically identify suspicious participants through aggregate suspicious behavior, subvert their decision structure and goad them into a position favorable to the defense. Baits include simulating insertion of readable and writable drives with weak or no password, marked doc/pdf/txt/exe/cad/xls/dat files, processes with popular target names and processes that detect thread injections.**

This approach bears some similarities to the concept of subverting an enemy's OODA (Observe, Orient, Decide, and Act) loop, an information warfare strategy which seeks to proactively influence and change enemy behavior. By controlling perception of the environment, this approach similarly seeks to influence adversarial participants' decision complexity, noise levels, effectiveness and ultimately their ability to fulfill their mission. This is a work in progress: The conceptual framework is described, and implemented baits and preliminary empirical results are presented.

The long term project end vision is an autonomic framework playing a repeated, dynamic, imperfect information, non-cooperative stimuli-response game which probabilistically identifies, then impedes, quarantines, subverts, possibly attributes and possibly inoculates against suspected adversarial cyberspace participants.

Keywords: virtualization, malware, dynamic game, stimuli, behavior

This research was funded in part by the US Department of Defense through a DoD IASP Grant (H98230-09-1-0369) which is administered by the US National Security Agency and in part by a grant from the University of New Orleans Office of Sponsored Research.

I. INTRODUCTION

It is written that a person's character may be recognized by how he handles alcohol, his conduct in financial matters and his anger. In other words, behavior shown in certain situations gives insight into character. A cyber-defensive approach in the form of a behavioral stimuli-response framework is presented in this paper. It should be noted that is work in progress.

Why is this needed? It is needed as an addition to Defense-in-Depth. The empirical performance of the first line of defense - anti-viral (AV) byte signature blacklisting - has been steadily declining. Independent laboratory test results over a period of a decade have shown a steady rise in false negative rates, i.e. failing to detect malicious code. The average miss rate of even previously submitted malicious code hovers in the double digits. In 2010, after failing to update static signatures *for just one week*, the best AV tested missed 37%, the worst between 60% and 90% (see TABLE I.).

TABLE I. DETECTION RATE RANGES OF SIXTEEN TO TWENTY POPULAR AV SCANNERS [1]

Report Date	AV Signature Update	Malicious Code Corpus Date	False Negatives (%)
2010/11	Aug. 16th	Aug. 17th-24th	[38-63]
2010/08	Aug. 16th	Aug. 6th	[0.2-19.1]
2010/05	Feb. 10th	Feb 11th-18th	[37-89]
2010/02	Feb. 10th	Feb. 3rd	[0.4-19.2]
2009/11	Aug. 10th	Aug. 11th-17th	[26-68]
2009/08	Aug. 10th	Aug. 10th	[0.2-15.2]
2009/05	Feb. 9th	Feb. 9th -16th	[31-86]
2009/02	Feb. 9th	Feb. 1st	[0.2-15.1]
2008/11	Aug. 4th	Aug. 4th -11th	[29-81]
2008/08	Aug. 4th	Aug. 1st	[0.4-13.5]
2008/05	Feb. 4th	Feb. 5th -12th	[26-94]
2008/02	Feb. 4th	Feb. 2nd	[0.2-12.3]

In addition, recent advances in formal computer virology show that detection of malicious code that poses the most problems (staged downloads and interactive) cannot be accomplished in linear time and enters the realm of exhaustive search space and undecidability. Reference [2] proved that detection of interactive malicious code is at least in complexity class $NP^{((NP^\wedge oracle)^\wedge(NP^\wedge oracle))}$.

This is no accident since the design and implementation of modern malware seeks to specifically undermine the information gain of static signature approaches, in effect presenting the defense with Halting-type problems. The reverse is *not* true: From the point of view of adversarial participants, cyber-targets are pathologically honest and do not systematically confuse adversarial participants with high entropy schemes.

This paper's view is that defenses have to *adopt similar comprehensive dissimulation and deception stances on cyber-targets and embedding environments*. By turning the tables on potentially Adversarial Cyberspace Participants (ACP), their code footprint, decision complexity, noise levels and uncertainty about the 'real' view of the cyber-environment are increased, thereby giving defenses more temporal and spatial leeway. This experimental framework is not meant to substitute for but rather complement traditional blacklisting byte signature based mitigation approaches whose limitations are well-known [3].

II. PRIOR WORK

This paper emphasizes the primacy of ACP control flow subversion through judicious manipulation of the environment's observables. In the realm of best-of-breed static structural signatures, [4] extracted malware family signatures using maximum graph homomorphism between the function callgraph of two executables. The flowgraph structure and its code 'neighborhood' were used to develop an opcode-sequence agnostic graph hash for fast approximate comparison. The intersection of known family members subsequently generated the family superstructure signature. Such a signature extracted automatically from 15 variants of the polymorphic malware family 'Swizzor' was able to recognize 900 additional variants (in a sample of 20,000 unsorted pieces of malware) with no false positives. However, this approach required pertinent structural information to be recovered; a proposition that does not hold with malware that purposely obfuscates its control flow structure.

This paper also posits ACPs (especially malicious software) to be sensitive to real or perceived operating environment changes. For evidence consistent with this assertion, the reader is referred to the 2005 analysis of the Slammer worm, in which complex dependencies between user/kernel processes and threading are described, as well as the 2008/2009 Conficker A worm, which exits upon detection of a Ukrainian keyboard locale [5] [6]. In a comprehensive 2008 empirical study, [7] investigated the environmental awareness of modern malware by measuring the deterrence value of imitating virtual machines and debuggers through light-weight registry key insertions, system call hooking (e.g. *CheckRemoteDebuggerPresent()* set to TRUE) and process generation (e.g. a process named OllyDbg, a popular debugger). Of the 6205 malware samples, about 25% reduced their malicious behavior through these light-weight techniques.

This paper's approach furthermore seeks to draw ACPs into a repeated stimuli-response game with the expectation that its dynamic behavior can be influenced quicker than non-malicious participants. In a comparative analysis of malicious and non-malicious software, [8] showed through statistical static structural analysis that malicious code tended to have a lower basic block count, implying a *simpler decision structure*: less interaction, fewer branches and limited goals compared to non-malicious software. This suggests that malicious code can be ‘outplayed’ by exploiting this simpler decision structure.

From an implementation point of view, honeypots and honeynets - simulated decoys that detract from 'real' networks, hosts and services – are well known examples of ‘morphing the network’, i.e. changing the perception of the network’s makeup. Reference [9] implemented a highly scalable, parsimonious hybridization of low- and high-interaction honeynets that doubled as a platform for malware collection. He suggested it to be used as part of an automated, next-generation system to stop botnets. Ad-hoc hot patching, as well as randomization techniques (randomized heap/stack/library positioning at compile, link and load times) are incorporated into modern operating systems like Windows Vista/7 [10].

Lastly, probabilistic identification and control of hitherto-unknown/unseen threats serves to enhance situational and behavioral awareness on a host, network and mission level. In this, this project complements other efforts in US military domains: DARPA’s Integrated Battle Command (BAA 05-14) gives decision aids for battle operations, DARPA’s Real-Time Adversarial Intelligence & Decision Making (BAA 04-16) tries to help battlefield commanders compute and counteract threat predictions in tactical operations. Lastly, Israel’s Virtual Battle Management AI - a defense system designed to handle situations that exceed the physiological limits of human command in case of a doomsday strike - mirrors most closely the project end vision [11].

III. DESIGN OF FRAMEWORK

The **Gameboard** consists of a virtualized operating environment (a Windows XP SP2 VM) which is ‘morphed’ by the Defender. Morphing means that from the point of view of Gameboard participants, the environment (or merely its perception by the participants) is altered via stimuli in order to provoke a reaction that could be used for identification. Stimuli (such as a .pst file, a simulated network drive, or a process named iexplore.exe) are introduced to induce potential adversarial participants (both humans and programs) to ‘show their colors’ (see Figure 1).

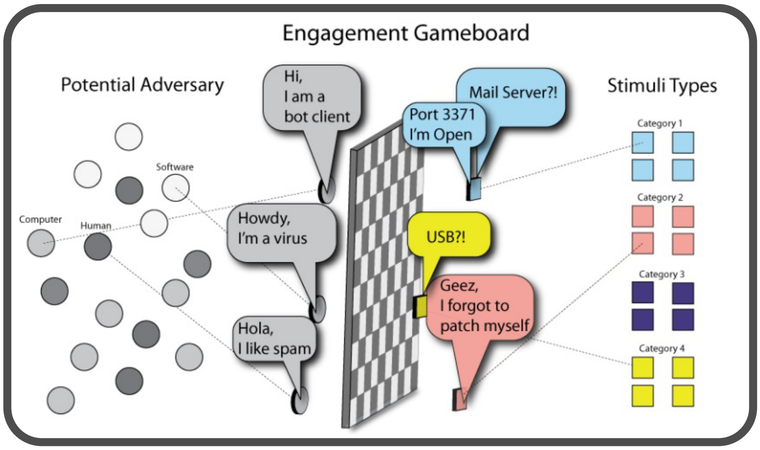

Figure 1: Notional Gameboard illustration. Stimuli (e.g. fake network drives, fake processes with names of popular applications, AutoCad files) are deployed and participants' responses to the baits evaluated.

Conceptually, a repeated, dynamic, imperfect information, non-cooperative stimuli-response game is played on the Gameboard. The players in the game are {Defender} versus {Participants}. All Participants (benign or malicious) are situated within the Gameboard (the VM). The Defender is situated *outside* the Gameboard to hide some of its footprint, but it has the ability to introduce baits/stimuli, change (real or perceived) macroscopic Gameboard parameters, gauge responses and initiate defensive moves.

The game's first goal is to judge whether after several rounds of the stimuli-response game the aggregate evidence warrants classifying a participant's observed behavior as adversarial. The concept of aggregate evidence borrows from Whewell's "Consilience of Induction", in which the convergence of several, ideally independent hypotheses serves to strengthen that conclusion [12]. Upon probabilistic identification, the game's second goal is to engage appropriate defensive measures to impede, quarantine, and subvert the ACP threat.

The working assumptions are as follows:

1. From observations of triggered stimuli and responses, uncertainty anent unknown intent can be reduced. In particular, potential adversarial participants can be probabilistically identified.

2. Defender can control the runtime behavior of ACPs by influencing what Participants perceive within the Gameboard.

A. Baiting Adversarial Cyberspace Participants

The repeated stimuli-response game can be conceptually decomposed as follows: A Defender conversation consists of a high level scenario which is either preemptively engaged, chosen by the user, or activated by other defensive systems (such as an NIDS). Conversation examples include "Worm", "Rootkit", “Bot", "Trojan", "Trusted Insider", "Hapless User" and more.

A Defender scenario informs one or more engagement types. Engagement type examples include “Offer spread vectors”, “Offer confidentiality vectors”, “Offer reconnaissance vectors”, “Present weakened defenses”, “Change system parameters” and more.

For each engagement type, the Defender autonomously chooses a dynamic engagement strategy. These engagement strategies consist of a game tree aggregate of baits/stimuli, participant responses and defensive responses. It is a dynamic game tree since moves are generated dynamically based on observed responses to previous stimuli.

B. Controlling and Deceiving Cyberspace Participants

Collberg's atomic primitives constitute *abstract categories of defenses* and are subsequently used as a blueprint for defensive responses upon probabilistic ACP identification [13]. These primitives are cover, duplicate, split/merge, reorder, map, indirect, mimic, advertise, detect/ response, and dynamic. The framework’s adaptation of some of these primitives is given below:

Quarantine [Indirect]: Defender moves ACP to an instrumented but isolated platform in order to learn more about its behavior.

(Self-)termination [Tamperproof]: Defender terminates ACP or induces its self-termination. In addition, the Defender may simulate termination of benign components as a strategic mimetic move (such as unlinking it from the process table).

Scarcity [Mimicry, Tamperproof]: Defender presents the Gameboard in a “critical” or “strained” state in an effort to violate ACP’s expected usage scenario (e.g. 99% memory utilization, heavy network congestion, no heap space left) [14].

Subversion [Tamperproof]: Data-taint/poison the input to ACPs in order to create an attribution trail (e.g. email bugs in .pst files). This is especially important for military defense systems, where attackers try to plausibly deny responsibility through one or more levels of indirection.

C. *Composition of Context-Sensitive Interactions*

It is an open research question whether engagement strategies can be derived from first principles (i.e. formal malware models [15]). Similarly, it is not clear a priori which set of defensive responses is best suited for which ACP classes. Empirical sandbox observation of 10,000s of malware samples (exhibiting a wide variety of behaviors) was scheduled, and samples were procured from a friendly malware repository, http://offensivecomputing.net. It turned out that lack of sample metadata (names were hashed) hindered the establishment of 'ground truth' (known identities of the control samples) anent the engagement strategies and the defensive responses. Hence, a systematic evaluation of the dynamic compositional question and concomitant quantitative measurements has not yet been undertaken.

D. *Views of the Gameboard*

Since they are situated within the Gameboard, all Participants have a view of the Gameboard, but not necessarily the same in terms of scope and fidelity. In particular, Participants' views and subsequent behavior are constructed by interacting with the Gameboard (checking if a certain process is running for instance).

Defender's view: All of the Participants' behavior unfolds over time. Some behavior on the Gameboard is benign, while some is potentially adversarial. Some behavior is seen by the Defender via baits that are triggered, while some behavior will not be seen. The Defender engages in conversations with Participants to figure out potential benevolence/malevolence.

Participant's view: The interactions between the Defender (through the Gameboard morphing) and the Participant influence the Participant's perception of the environment and, as posited, subsequent Participant behavior. This behavior may in turn influence the Defender's strategies, and so on, until identification decision thresholds are reached and defensive responses are engaged.

E. *Goals of The Defender*

Drawing from prior experience and input from stakeholders, a list of Defender goals was assembled, in descending order of importance. These goals influence both the nature of the baits/stimuli injected into the Gameboard, the timing of their deployment, as well as defensive responses.

Mission Continuity: Defender should not self-sabotage or sabotage the mission of benign Participants in the Gameboard. The primary goal of any defense is to sustain the mission. Mission continuity constraints include but are not limited to: sustaining mission availability, confidentiality, integrity, and command and control authenticity.

Actionable Information Gain: Defender's responses should be geared towards reducing uncertainty and learning more about potential ACPs. This is in part accomplished by the interactions in the dynamic game. In addition, freezing the Gameboard and migrating the ACP threads into a more highly instrumented environment is being explored.

Defender Stealth: Potentially adversarial participant should remain unaware of Defender's observation and manipulation of ACP's perception of the Gameboard. This is accomplished by positioning the Defender outside of the Gameboard and by randomizing design and implementation aspects of the baits.

Subversion: Defender responds in such a way as to repurpose the adversarial participant for the benefit of the Gameboard's mission. One possibility is supplying the ACPs with specially crafted random input, which has been shown to crash in other contexts between 25%-40% of given applications [17][18].

Participant Attribution: Defender responds in such a way that attribution of an adversarial behavior source is made more likely (e.g. smart watermarking/ poisoning of data).

Inoculation: Defender may be able to synthesize a general modus operandus over observed behavior for the purpose of inoculation: Through judiciously chosen baits the traversal of appropriate control flow paths in the ACP is induced. This is in keeping with the light-weight shutdown results of [7].

IV. Implementation of the Framework

The Defender needs to influence and control the Gameboard environment in a way that is transparent to the Participants. The VMWare platform was chosen due to its market share and proprietary design. Unlike Bochs or Qemu, VMWare's code is not normally available, forcing manipulation of the VM from the 'outside', with no detectable footprint in the Gameboard besides the baits. Since there are numerous robust VM detection approaches, it is reasonable to assume that Participants can ascertain whether they are running in a virtualized environment [19]. As virtualized execution environment are becoming more commonplace with the push towards large-scale virtualized commercial environments, this is a reasonable extension and benefits the scheme.

A. VMUtils library

VMware's VIX API is used to control the Virtual Machine [20]. Since VIX is still in flux, further modifications led to the development of the library *VMUtils*.

VMUtils wraps a number of VMWare's VIX library functions in order to simplify calls to the VIX API. An example is getting a handle for a VM which previously consisted of multiple lengthy and confusing VIX API calls, all of which had to be paired with additional error checks. The VMUtils library abstracts bait implementation away from the Defender's engagement strategies and allows for bait design through a mediate layer. This is also useful for a centralized VM administration approach like VMWare Server. Alternatively, a conventional network-based communication API could be substituted in place of VMUtils.

B. *Baits/Stimuli*

Baits/stimuli seek to alter the perception of the operating environment (i.e. 'morph the Gameboard') in order to induce tell-tale behavioral responses from potentially adversarial cyberspace participants. Some changes in the environment are lightweight, sometimes they are entirely simulated:

- Simulating insertion of readable and writable media
- Simulating creation of Network Drives with weak or no password
- Planting marked doc/pdf/txt/exe/cad/xls/dat files
- Planting bank cookies
- Creating fake processes with names of popular AV programs
- Creating processes to detect thread injection
- Navigating a browser to Microsoft Update/AV sites (to see whether access to these sites is blocked)
- Navigating a browser to a bank site to see if participants attempt a XSRF attack
- Navigating a browser to a social network site known to be vulnerable to XSS attacks
- Simulating a particular bot client
- Slowing down or speeding up Gameboard system time

A robust bait portfolio must give quantitative metrics on adversarial participant specificity and sensitivity: Low false positives are desired (i.e. does it flag benevolent participants as adversarial?), as well as low false negatives. This is ongoing empirical work and has not yet been addressed. The baits developed and deployed to date are described below.

1) Dummy Process

A dummy process execution and monitoring bait was implemented first. A well-known ACP tactic is, after infecting a machine, to turn off or uninstall AV software. This bait hence targets the self-defense trait of ACPs by executing a number of bait processes named after popular AV programs and monitors them for execution disruption. Alternatively, it is possible to implement a callback-model for the dummy process baits: the bait program creates new threads for each new bait AV process started within the Gameboard, then makes the thread wait for an exit code from the bait process. The later design was chosen.

A list of common AV process names (e.g. avguard.exe) was compiled into a *config* file, which is read by a baiting program. The baiting program then renamed the dummy process, copied it down to the Gameboard, and executed the bait AV process. By waiting for exit codes from the processes running in the VM, the Defender determined if any (and how many) baits were tripped – in other words, which bait AV programs were terminated.

There are very few legitimate reasons (Force Quit, for instance, being an exception) a non-malicious program would kill a running process of a common AV. Intuitively, this bait has high malicious code specificity. It may have low sensitivity depending on how many ACPs attempt to terminate the dummy processes.

2) *Network Shares*

Another common ACP tactic is *spreading* via network shares. A mechanism was implemented to mount and remove network shares and monitor them for access; the rationale being that spreading is common for malicious code with network shares representing tempting targets. A Defender directory was mapped to a network drive on the Gameboard. The directory was monitored for changes, immediately alerting the Denfender if an attempt was made to write to the network share.

Since USB keys were used by Conficker and the 2008 Central Command attack for spreading, attempts were undertaken in conjunction with the network shares. It turned out to be harder than anticipated, due to the way VMWare handles USB devices.

3) *Data and System Files*

A similar mechanism can be used for bait files. As the January 2010 Aurora attack showed, industrial espionage targets the confidentiality of intellectual property, such as AutoCad design files. By data-tainting a seemingly high value file, it is hoped that an attribution trail can be established. Steganographic means may be pursued, but a simpler mechanism was chosen for proof-of-concept.

A data file was created on the host machine containing a bogus .gov or .mil email address (or other attractive metadata), then copied into the Gameboard and monitored for activity. This bait aims to coax out malicious actions of potential ACPs: Some instances of malicious code will search a filesystem looking for anything that looks like email addresses, accounting spreadsheets, Outlook .pst files or other data files. Using the same monitoring program from the network share bait, this bait would be tripped on file access or, at a later point in time, by bogus email usage. Although this bait is straightforward to implement, it may have lower specificity due to installed indexers like Google Desktop. This mechanism may also be applied to sensitive system files, complementing Windows File Protection.

4) AV Sites

Editing the Windows Hosts file is a way that malicious code will attempt to block web access to AV websites. This is an example of an ACP's self-defense trait with high specificity.

A bait program was written to test connectivity to many known AV websites. The bait program read URLs from a *config* file and sent http requests to the web-server from within the Gameboard. The first request was sent to the URL; then, using an external DNS server, the same request was sent to the corresponding IP address. Return codes were then compared to determine if malicious code had tampered with web requests. If no determination could be made, a HTTP request was sent by the Defender from the outside and used as a control to compare the previous samples taken from within. Determining whether malicious code is interrupting connections to AV servers constitutes a highly specific indicator of malicious behavior. In 2010, a similar method was used for the Conficker Eye Chart test to test for Conficker infection.

5) User Activities

Another scheme is to simulate normal user behavior to coax out malicious ACP action. Any form of day-to-day user activity might constitute a trigger for malicious code. Such activities include, but are not limited to, checking email, program execution, online banking, or social networking. These activities are simulated and monitored for interruption or abnormal execution.

As a proof of concept, Visual Basic (VB) scripts were used because of the tight integration with Windows and the MSDN references [21][22]. These scripts were deployed onto the Gameboard. Although this needs to be verified empirically, the script's execution isn't likely to be detectable with an acceptable false positive rate by malicious software because MS Windows' handling of VB scripting through wscript.exe. The observable change the ACP sees is wscript.exe running, but there is no straightforward way to tell what it is doing or that it is a Defender's bait script. An example is given in Figure 2, a Yahoo login script controlling a Firefox browser.

```
Set oShell = WScript.CreateObject("WScript.Shell")         ' Create a Shell object.
Dim cmd
cmd = Chr(34) & "C:\Program Files\Mozilla Firefox\firefox.exe" & Chr(34)
oShell.run cmd,0,True  ' Open Firefox
WScript.Sleep 1000
oShell.sendKeys "%d"                  ' Firefox keyboard shortcut for Select Location Bar
oShell.sendKeys "www.mail.yahoo.com"
oShell.sendKeys "{ENTER}"
WScript.Sleep 10000
oShell.sendKeys "emailAccount"      'User name
oShell.sendKeys "{TAB}"
oShell.sendKeys "letmein"             'Password
oShell.sendKeys "{ENTER}"
WScript.Sleep 1000
oShell.sendKeys "^t"          'Firefox keyboard shortcut for New Tab
oShell.sendKeys "%d"          'Firefox keyboard shortcut for Select Location Bar
oShell.sendKeys "www.evil.com"
oShell.sendKeys "{ENTER}"
```

Figure 2: Navigating Firefox to Yahoo.com with wscript.exe

This example shows logging into a Yahoo mail account, opening a new tab, and navigating to a different website. Should an XSS injection be detected (in conjunction with an open source tool, XSSer), the Defender is notified. This feature is being validated further.

6) *Thread Injection*

Windows' *CreateRemoteThread(..)* serves as a prevalent exploit vector for malicious code. This function, exposed by Windows executables, enables the injection of an arbitrary thread inside the memory space of other processes [23].

Similar to the Dummy Process bait, a process is deployed to run within the Gameboard, continuously querying its number of threads. Once the bait process detects additional threads, it reports back to its monitor outside the Gameboard and terminates. Detection of remote thread injection was chosen under the assumption that it represents both a highly specific and sensitive trigger of malicious activity in non-debugging environments.

7) *Macro-Enviromental Triggers*

Under development are so-called *macro-environmental triggers*, such as controlling tick time within the Gameboard. By speeding up or slowing down tick time, time-dependent actions could be triggered, thus allowing for more inference anent ACPs' decision structure, patterns and/or movements. Macro-environmental triggers are by their very nature not highly specific: sudden loss of resources such as RAM/HDD shortages and network congestion have been shown in other contexts to crash programs in unexpected ways [14].

C. *Defender*

As noted, the Defender schedules, organizes, and monitors the baits, as well as coordinates defensive responses. It also keeps track of information about the Gameboard, such as logins and file paths.

The Defender loads information about the baits it intends to run from a *config* file. These baits are then deployed in an order, timing and frequency determined by a dynamic engagement strategy. The baits write information back to the controller via a pipe: millisecond timing analysis, bait trip counts, and errors are subsequently stored in a database. This aggregate evidence is used to weigh different hypotheses (using a Bayesian log likelihood model selection approach [16]) anent the observed behavior and formulate dynamic engagement strategies. This is still under development; in the proof-of-concept prototype, only static strategies have been implemented so far.

V. PRELIMINARY EMPIRICAL VALIDATION

Implemented baits are summarized in TABLE II. The rightmost column lists malicious code examples that informed the design of the baits.

TABLE II. IMPLEMENTED BAITS AND MALWARE TRIGGERS

Bait Name	**Bait Action**	**Malware example**
Dummy processes	Inject false antivirus programs into the OS process list and monitor for halt in execution	Conficker [24] (kills AV processes), Bugbear [25] (shuts down various AV processes), Vundo[26] (disables Norton AV)
Network Shares	Mounts and removes network shares on the client then monitors them on the server's side for activity	MyWife.d [27] (attempt to delete System files on shared network drives), Lovgate [28] (copies itself to all network drives on an infected computer), Conficker (infects all registered drives)
Files	Monitors system critical or bait files on the client for activity	Mydoom.b [29] (alters the host file to block web traffic), MyWife.d (deletes AV and system programs), Waledac.a [30] (scans local drives for email addresses)
User Action	Executes “normal” user behavior on the client system and monitors for unusual execution	Mydoom.b (diverts internet traffic, thus altering what is expected to appear), Vundo (consumes system resources and slows or impedes program execution)
Thread Injection	Continually queries its number of threads for any changes from the expected number	Poisonivy [31] (injects code into processes such as ‘explorer.exe’ or ‘msnmsgr.exe’), Pandex [32] (seeks ‘iexplore.exe’ program to inject its code)

The Win32 MyDoom.b email worm was used to generate the following time line points: t_{0a} (bait setup), t_{0b} (bait deployed and ready to be triggered), t_1 (malicious code is executed), t_2 (bait is triggered), and t_3 (bait is recalled/terminated) as described in Table III.

TABLE III. MYDOOM.B TIMING RESULTS (AVERAGE IN SECONDS)

Bait	t_{0a}	t_{0b}	t_1	t_2	t_3
Files (watching critical directories)	67	68	68	69	70
User action (checking AV websites)	70	71	68	73-103	103

These preliminary timing experiments are consistent with the second assumption stated in the beginning of Section III: ACP runtime behavior can be influenced by

Gameboard perception. Many of our samples actually failed to run within the Gameboard. Upon closer inspection, it seems that the virtualized environment provides a certain amount of protection in itself; malicious software often checks whether it is running in a debugging and/or virtualized environment and subsequently does not exhibit malicious behavior [33].

VI. FEASIBILITY AND FUTURE WORK

It should be clear from the exposition that this experimental framework is merely at an early proof-of-concept stage. Whatever research direction is charted, quantification of metrics and empirical validation are to be addressed since they represent methodological lacunae in the literature. A meta-survey of ninety security papers between 1981 and 2008 showed that quantified security was a weak hypothesis because of lack of validation and comparison against empirical data [34]. Bearing this in mind, future research must additionally tackle the following issues:

Behavior inferred by the stimuli-response framework needs to be modeled. Leveraging previous behavioral ontology work [35] and following Shannon's terminology for Markovian models, a mechanism was recently proposed to extract and characterize cyber-behavioral traits of humans for classification, prediction and change detection purpose. That framework introduced the notion of 0^{th} (atomic elements), 1^{st} (atomic + frequencies + context), and 2^{nd} order (probabilities of sequence of activities + context) behaviors. The approach has been evaluated in domains such as military targeting, stress monitoring, and insider threat detection with encouraging results [36].

From a game-theoretic perspective, the game may be played with *obscuring* participants. Obscuring participants may be able and willing to play sub-optimally (not take baits for example) to thwart behavioral estimates. In the context of cyber adversaries, maximum-entropy and hidden Markov model methods have been used to estimate subgame probabilities (i.e. the proportion of time spent in malicious and benign subgames). This approach may be extended to obfuscating adversaries, who attempt to hide their subgame probabilities [37].

Lastly, in order to transition the framework to production systems, the performance and stability challenges of scaling to 100,000s of virtualized hosts on infrastructure clouds will have to be kept in mind at design time [38].

As noted, the project end vision is an autonomic framework playing a repeated, dynamic, imperfect information, non-cooperative stimuli-response game which probabilistically identifies, then impedes, quarantines, subverts, possibly attributes and possibly inoculates against suspected adversarial cyberspace participants. Speculatively, an autonomous defense 'alter ego' for human decision makers is envisioned which, when coupled with physiological sensors, remains poised to take over when human judgment is deemed to be too affected by emotions and/or information overload. As far-fetched as this may sound in 2011, skeptical readers are invited to peruse the US Air Force Chief Scientist's vision for 2010-2030 [39].

ACKNOWLEDGEMENTS

We thank the anonymous reviewers at the National Security Agency and at the ICCC3 conference for their helpful suggestions and comments.

REFERENCES

[1] A. Clementi, “Anti-Virus Comparatives,” http://av-comparatives.org, Feb. 2008 - Dec. 2010.

[2] G. Jacob, and E. Filiol, “Malware as Interaction Machines,” J. Comp. Vir. 4:3, 2008, pp. 235-250

[3] M. Locasto, Y. Song, and S. Stolfo, “On the infeasibility of modelling polymorphic shellcode,” in ACM CCS, 2007, pp. 541–551

[4] T. Dullien, and E. Carrera, and S. Eppler, and S. Porst, “Automated Attacker Correlation for Malicious Code,” in Proceedings of the NATO IST Symposium (Tallinn, Estonia), November 2010, pp. 26.1-26.10

[5] J. Crandall, Z. Su, S. Wu, and F. Chong, “On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits,” in Proceedings of the 12th ACM CCS, pp.235-248, 2005.

[6] P. Porras, and H. Saidi and V. Yegneswaran,” An Analysis of Conficker”, SRI International Technical Report, March 2009.

[7] X. Chen, “Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware”, ICDSN Proceedings, pp. 177-186, 2008.

[8] D. Bilar, “On Callgraphs and Generative Mechanisms,” in J. Comp. Vir. 3:4, 2007, pp. 285-297

[9] D. Zamboni et al., “The Nepenthes Platform: An Efficient Approach to Collect Malware,” in LNCS 4219, Berlin: Springer, 2006, pp. 165-184.

[10] M. Conover, “Assessment of Windows Vista Kernel-Mode Security”, Symantec Advanced Threat Research, 2006.

[11] N. Shachtman, " Israel Eyes Thinking Machines to Fight ‘Doomsday’ Missile Strikes," Wired Danger Room, http://www.wired.com/dangerroom/2008/01/israel-thinking/, January 2008

[12] L. Snyder, “'The whole box of tools': William Whewell and the logic of induction,” in Handbook of the History of Logic - British Logic in the 19th Century, vol. 4, 2008, pp. 163-228

[13] C. Collberg. “Surreptitious Software: Models from Biology and History,” Computer Network Security Series, Berlin: Springer, 2007, pp. 1-21

[14] H. Thompson, J. Whittaker, and F. Mottay, “Software Security Vulnerability Testing in Hostile Environments.“ in Proceedings of the ACM Symposium on Applied Computing, 2002, pp. 260-264

[15] S. Kramer and J. Bradfield, “A General Definition of Malware,” J. Comp. Vir 6:2, 2010, pp. 105-114

[16] R. Kass and L. Wasserman, “A Reference Bayesian Test for Nested Hypotheses and its Relationship to the Schwarz Criterion,” in Journal of the American Statistical Association 90:341, 1995, pp.928-934

[17] B. P. Miller, G. Cooksey, and F. Moore, “An Empirical Study of the Robustness of MacOS Applications Using Random Testing,” in Proceedings of the 1st International Workshop on Random Testing, 2006, pp. 46-54

[18] B. P. Miller, L. Fredriksen, and B. So., “An Empirical Study of the Reliability of Unix Utilities,” in CACM 33:12, 1990, pp. 32-44

[19] P. Ferrie, “Attacks on Virtual Machine Emulators“, Symantec Advanced Threat Research, 2007

[20] VMWare, “VIX API 1.10.2 Documentation”, http://www.vmware.com/support/developer/vix-api/, October 2010

[21] M. Russinovich and D. Solomon, “Microsoft Windows Internals,” 5th ed., Microsoft Press, June 2009.

[22] Microsoft, “VBScript Language Reference”, http://msdn.microsoft.com/en-us/library/d1wf56tt%28v=VS.85%29.aspx, 2011

[23] B. Blunden, “The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System,” Plano, TX: Wordware Publishing, 2009, pp.245-265

[24] V. Tiu, "Virus Analysis - Confounded Conficker," Virus Bulletin, pp. 7-11, March 2009

[25] F-Secure, “Virus Encyclopedia – Worm:W/32Bugbear”, http://www.f-secure.com/v-descs/tanatos.shtml, 2009

[26] Symantec Corporation, “Security Response – Trojan.Vundo”, http://www.symantec.com/security_response/writeup.jsp?docid=2004-112111-3912-99, 2011

[27] McAfee Inc., “Virus Profile: W32/MyWife.d@MM!M24”, http://home.mcafee.com/VirusInfo/VirusProfile.aspx?key=138027, 2010.

[28] Sophos Ltd., “Sophos Security Analyses – W32/Lovgate-E Win32 worm”, http://www.sophos.com/security/analyses/viruses-and-spyware/w32lovgatee.html, 2011

[29] F-Secure, “Virus Encyclopedia – Email-Worm:W/32Mydoom.B”, http://www.f-secure.com/v-descs/mydoom_b.shtml, 2009

[30] F-Secure, “Virus Encyclopedia – Email-Worm:W/32Waledac.A”, http://www.f-secure.com/v-descs/email-worm_w32_waledac_a.shtml, 2009.

[31] D. Elser, "Metafile Art Class," in Virus Bulletin, June 2008, pp. 4-7

[32] C. Prakash and A. Thomas, "Malware Analysis – Pandex: The Botnet That Could," Virus Bulletin, pp. 4-8, March 2008.

[33] T. Raffetseder, C. Kruegel, and E. Kirda, “Detecting Systems Emulators,” in LNCS 4779, Berlin:Springer, 2007, pp.1-18

[34] V. Verendel, “Quantified security is a weak hypothesis”, in Proceedings of the NSPW, 2009, pp. 37-50

[35] N. Sandell, R. Savell, D. Twardowski, and G. Cybenko, “HBML: A Representation Language for Quantitative Behavioral Modeling in the Human Terrain,” in Social Computing and Behaviorial Modeling, New York: Springer, 2009, pp. 180-190

[36] D. Robinson, “Cyber-Based Behavioral Modeling”, PhD Thesis, Dartmouth College (Thayer School of Engineering), July 2010

[37] J. T. House and G. Cybenko, "Hypergame Theory applied to Cyber Attack and Defense," Proc. SPIE, vol. 7666, 2010

[38] E. Kotsovinos, “Virtualization: Blessing or Curse?, ” in CACM 54:1, pp. 61-65, January 2011

[39] W. Dahms, "Technology Horizons: A Vision for Air Force Science & Technology During 2010-2030," Technical Report, USAF Science and Technology, http://www.af.mil/information/technologyhorizons.asp, May 2010