Heuristic Security-Testing Methods

John E. Kerivan, Ph.D.

Originally published in the Journal of Digital Forensic Practice, Volume 1, Issue 1, March 2006 – ISSN 1556-7281

Abstract – This is the first of two papers that deal with the development of running state requirements for functional testing of security software and hardware systems. It outlines the need to adopt paradigms that reflect typical usage patterns, prevalent infection methods and proper security tool use and configurations that are grounded in real-world scenarios. This paper outlines a practical set of such test tools based on attack infection techniques designed to evaluate the efficacy and utility of signature as well as knowledge-based security systems, including those found in forensic toolkits. Signature-based testing of security solutions is complicated by the continuing increase in the number of attack signatures. Likewise, realistic behavioral testing methods for the same suffer from the increasing numbers of combinations and permutations for attack infection methods that quickly become outdated as new attack categories emerge. However, the usage patterns and base attack infection techniques have remained largely stable over the past four years. Thus, the heuristics associated with a recognizable set of security principles presents an opportunity and a challenge to construct forensic analysis test solutions based on the use of a security-pattern database (SPD) and the concept of adaptive event logging. I propose such a mechanism in this paper using three domains for the SPD and trigger requirements for ensuring that application, security, system and network logging are enabled for selected events. These domains represent the normal usage patterns of PCs, the basic attack infection method categories, and the security tool capabilities and their configurations necessary for optimum computer protection. This paper also shows a heuristic security checklist formed from the decomposition of 50 Trojans, worms and spyware and used as the basis of prevalent attack infection techniques currently in the wild. The purpose of this exercise is to assist digital forensic practitioners with a decision support tool during evidence gathering and analysis phases of an investigation. Recommendations are provided that show effective signature and behavioral heuristics for further refining sub-problems in the three domains. The most effective security test techniques are also shown to provide a common set of principles for the SPD.

INTRODUCTION

A heuristic is traditionally defined as the art and science of discovery and invention. It is characterized by simple and efficient rules of thumb based on how judgments are found or problems are solved with incomplete information. After a cybercrime has been committed, incomplete rules for gathering admissible evidence have regularly challenged first responders who work in the field of digital forensics (DF). This factor has caused the DF field to focus on those elements of an investigation that can pass legal muster. Government agencies including the Department of Justice (DOJ), the Federal Bureau of Investigation (FBI), the Department of Homeland Security (DHS), the National Institute of Standards and Technology (NIST) and others have quantified important and necessary guidelines for the conduct of a DF examination. These guidelines have naturally focused on the chain of custody as well as on the preservation of file and drive evidence found in the discovery process. However, the preponderance of DF tools and techniques have focused on file and hard drive analysis almost to the exclusion of other critical network and system related information about typical machine usage and attack infection techniques. For example, current DF evidence gathering practices focus on the following procedures:

- Documenting the machine components including hard drives, NICs to the MAC level, etc.
- Determining the existence of potential evidence; e.g., identifying the machine, finding available event logs
- Determining the existence of additional information; e.g., system logs remaining on the machine
- Determining the need for potential preservation orders to ISPs
- Assessing the skill level of the users of an associated computer.

Once it has been determined that a computer of interest has been involved in the commission of a crime, the discovery process is followed by an extraction process where file carving, keyword searches, etc. are performed on copies of affected disk drives. This extraction process is then followed by a reasoned approach to determine if sufficient evidence exists to satisfy the legal hurdles of a trial while providing inculpatory and exculpatory evidence that can be shown as tamper-proof and can show a valid chain of custody for the supporting evidence.

It is the last step that separates routine security administration and management from DF examinations. It is chain of custody requirements that have created confusion in the minds of some network and system administrators about acceptable practices for the support of potential legal proceedings in the event of a cybercrime.

Recent publications by the DOJ have been issued to help clarify the requirements for early responders in the event of the commission of a cybercrime.[1,2] However, there are no earlier responders than those who administer the systems and networks. They need guidelines on re-creating and preserving a cybercrime scene, especially how to protect correlated log data from network routers, switches, disk transfer and process table

information on affected machines. Such dynamic information can quickly steer a DF investigation into the most probable conditions to find and prosecute cybercrime. Unfortunately, application, security, system and network logging are seldom captured and correlated with significant security events; this information is thus rarely available to DF practitioners. Even the newer security information management (SIM) tools use store-and-forward mechanisms to move SNMP (*Simple Network Management Protocol*) or *syslog* data to central management platforms that can be an order of magnitude slower than what is needed to determine culpability in a cybercrime investigation.

There is also resistance to change. It is argued by system and network administrators that the overhead associated with capturing all system and network activity is cost prohibitive, especially from CPU and storage requirement perspectives. Such arguments disregard the dramatic decreases in price of disk storage, memory, and processing power in recent decades.

DF investigators need to capture real-time log data like a snapshot of an event. When a security event involves one machine this is a simple problem. An agent is alerted to dump the activity logs to a secure file, capture other running-state information such as process tables, registry activity, and disk transfer activity *before* the logs are lost or overrun and *before* a perpetrator or process initiator leaves a cybercrime scene. When an event involves a breach originating outside a machine, the same adaptive event logging must be triggered on intermediary network devices and the data must immediately be saved to avoid lost or overrun information.

In this and the next paper I will show that adaptive event logging provides an acceptable solution to the problem of cost-effective data capture by capturing only event-significant information in such a way that it can quickly assist early responders in dealing with security breaches. Scheduling and other administrative tools currently exist on most if not all platforms that can trigger adaptive event logging and provide acceptable and secure file transfer to central management stations without adding significant overhead to systems and network devices. What is missing is a decision support mechanism to alert target systems when a security event is imminent or in progress.

The DF field is also challenged by many tools in use by security product vendors that are derivatives of early UNIX system management programs such as *grep* and *dd* that have been repackaged with additional integration capabilities to better address the diagnostic needs of a DF investigator. There are so many tools that NIST has recently called for the establishment of standard methods for the evaluation of DF tools[3]. Again, little is mentioned about PC usage guidelines and attack infection methods that can also assist an early responder in determining the root cause of a significant security event. Attack infections methods comprise the procedure(s) that are followed to create a security breech. The procedures can be thought of as invariant within a family of Trojans, Worms or Spyware. They are not used to form the basis of a signature and may rely on programmatic or other mechanisms to invade a system. Note that the determination of whether sufficient evidence exists to prosecute individuals is usually made at a later stage

of an investigation, but it would be nice to have a snapshot of the scene while a breach is in progress to help make that decision.

Due to the emphasis placed on acquiring file and drive information, some critical data acquisition and analysis tasks are often missing in preserving a cybercrime scene prior to the arrival onsite of DF practitioners: specifically, the dynamic environment in which a crime has been committed too often remains undocumented. The dynamic aspects involve the state of the active network components, accurate representations of processes and threads running at the time a cybercrime has been committed and the type of infection method used to perpetrate the crime. They include the typical PC usage patterns (e.g., where users surf the Web, what applications are in use at the time of a security event) as well as the degree of protection for PC security tools afforded on the target machine. They also include the resistance of security tools to improper suspension or termination – factors which contribute to the compromise of a system and potential loss of critical log evidence. Under compromised conditions, these dynamic components are not contained on alternative media, especially when an attack infection originates outside a target machine. In some cases, there are vendor solutions capable of approximating the running state of a machine involved in a cybercrime. Unfortunately, they must make general assumptions about operational system and network profile states. They also appear to be missing information about some of the methods used to attack a system or perpetrate a cybercrime.

To make matters worse, the hacker community has been increasing the number of Trojans that can turn off the programs currently found in forensic toolkits. As of October 2005, there were over 160 Trojans, worms and blended threats in the wild that disable security, administrative and forensic tool components. In previous work[4], I reported on their techniques, which include suspending and killing security and system utility software, performing their desired activities, cleaning up (some have embedded multi-pass disk and file shredders) and then leaving the compromised system with little or no evidence that a security event has occurred or that a crime has been committed. It is therefore critical that the running state of machines be captured and correlated with network and host process data when evaluating security events for potential cybercrime evidence. It is also critical that more robust protection components be built into forensic toolkits and other security software that will be capable of defending themselves against such attacks and thus will preserve the integrity of a cybercrime scene.

This paper is the first of two that will describe a real-world test environment. The environment

- is bounded by the use of prevalent attack infection methods
- is based on the operational design of current Trojans, worms and spyware,
- includes the normal usage patterns of PCs,
- takes into account security-tool capabilities and
- manages event-logging configurations resident on a target PC at the time of a security event.

This information has been used to develop a heuristic security checklist (HSC) for the population of field-level data of a proposed security-pattern database (SPD). The publication of the schema for the SPD will also be shown in this paper. The next paper will show results from tests of a decision support system built upon the SPD and which can assist DF practitioners in the early stages of an investigation.

METHODS

As mentioned in the Introduction, an isolated lab was configured to provide a representative set of PC usage conditions under which the security and forensic applications could be exercised against malware (Trojans, worms and spyware). The annotated list of malware with descriptions can be found on the Web[5].

The lab was needed for two purposes. The first was to provide a secure facility in which the Trojans, worms and spyware could be exercised and their major functions catalogued. For example, in the spyware category, most malware is designed to set hooks into application or system message queues. The second was that the utility of selected security tools could be evaluated in light of the discovered malware characteristics. These tasks provided an opportunity to enumerate heuristics that were common within a malware category, thus leading to the development of an SPD knowledge base that can provide guidance to DF practitioners during an investigation.

Lab Machine Baselines

The lab included standard *Windows 2000* (Win 2K) PCs and their common applications (e.g., Microsoft Office, Internet Explorer) as well as other applications that could also be evaluated for baseline interoperability. Nine test systems were profiled prior to the start of testing to develop pre-test baselines. These were our ghost images with no interactive users but connected to our security-test local area network (LAN).

All Windows systems were monitored with Microsoft performance monitors as shown in the next section as well as another tool available from an independent third party vendor (www.sysinternals.com) called the *Process Explorer v 8.35*. The Process Explorer was an important tool to use for this type of testing as it could show the real-time process and thread execution of the malware emulators. The CPU cycles consumed by each process tree were documented and are shown in the Results section. The Process Explorer showed the amount of private bytes of memory consumed by each running program. It also provided insights into the processes, threads and modules used by the security applications and their instantiations as they responded to the top 50 attacks. It should be noted that the sysinternals.com Web site currently offers capable freeware that can assist DF practitioners in the evaluation of a cybercrime involving Windows platforms.

Performance Monitor Parameters

Performance monitors were activated on all test systems to track various usage parameters. The monitor used on Win 2K machines was the *Microsoft Management Console 1.2 version 5* (*Perfmon*) and was set to capture the following nine plus other parameters. Note that each of these parameters is a required field in the SPD and corresponds to guidelines for proper system performance published by Microsoft [6].

1. Cache - % Data Map Hits: shows the % of data that were resolved without retrieving a page. This is used to check on whether RAM is sufficient on the system in question;
2. Logical Disk – Average Disk Queue Length: shows an inverse relationship between an increase in queue and a drop in hard drive performance (delays);
3. Memory – Pages per second: shows an inverse relationship between increase in paging and lack of sufficient main memory;
4. Network Interface – Total Bytes per second: shows TCP/IP network usage at the local system interface. This is correlated between all systems and network analyzer data;
5. Objects – Processes: shows the number of running processes during the measurement intervals associated with each scenario;
6. Physical Disk – Average Disk Queue Length: shows queue length combined with available physical memory;
7. Processor - % Processor Time: shows CPU usage by percent of available time. If this reaches 65% the processor needs to be upgraded to a higher speed;
8. System - % Total Processor Time: shows the total time usage over all processors (if available);
9. Server – Total Bytes per second: shows how busy the server is while processing system level I/O.

The nine usage parameters identified above were used to provide potential application tuning information and to show bottlenecks as a function of virtual user activities captured by the Process Explorer during the execution of all tests.

At the same time a series of malware emulation tests were constructed to test the strength of the various system and security products. The tests were constructed based on standard malware categories and included the following;

- Destructive Trojans – designed to erase hard drives, disable system BIOS, etc.
- Remote administrative Trojans – designed to take control of victim hosts via network connections
- Keylogging Trojans and spyware – designed to capture user logon credentials
- Information-stealing spyware – designed to capture hard drive contents and ship via Internet connections to a third party site
- Multipartite worms – designed to infiltrate a site, disable running security software and capture user data and logon credentials

- Polymorphic malware – designed to be combinations of the above categories and capable of changing component identities, signatures and methods from one instantiation to the next

Attack Infection Methods

Four basic malware injection methods were used to bracket known infection techniques. They included the following:

1. Local physical access as an interactive user; e.g. from a local disk or malicious user;
2. Infection by email as well as an attachment; e.g. an embedded link vs. a script;
3. Clicking on an infected Web link using IE and permitting file downloads;
4. Use of a Browser Helper Object in IE to permit direct memory R/W capabilities, similar to comload[7] or warez (illegally-copied software) attacks.

When a top 50 program used standard silent installations to infect a victim system, emulators used the same packing, obfuscation and unpacking methods as the originals. MSI (Microsoft Installer) as well as INF-based (Information File) installations were attempted for top 50 programs using such methods.

Security Tool Configuration

The default security policies were used as the starting point for the combined anti-virus, anti-malware, system tools that are found in forensic toolkits and IDS/IPS (intrusion detection system / intrusion prevention system) tests. In these cases, the vendors provide a set of mechanisms to perform policy-based configuration for their respective toolset. The common architecture of this model is readily available on many vendor Web sites. In these cases, the configuration software is typically implemented with a security-policy server. This policy server and other third-party policy servers evaluate end-point security credentials relayed from a network-access device and determine the appropriate access policy to be applied (permit, deny, quarantine, or restrict).

Some vendors have policies defined that permit selected products to interoperate with theirs. For those security vendors who do not participate in an interoperability program, false alarms were exceedingly high. However, it should be noted that with the increased sophistication of Trojans and spyware, most security software is stopped and unloaded at the very onset of an attack. See the next section for additional data on this topic.

RESULTS

Typical PC Usage Patterns

Several studies have described how a PC is used during a normal business day. There are very few studies that have been published on non-commercial PC usage patterns. The *PC Power* studies commissioned by the US Department of Energy[8] serve as a reasonable estimate of the amount of time a PC is running and whether it is being actively used or not.

- For a normal workday, the PC is in the ON state but idle 35% of the time.
- A user is actively running and using programs roughly four hours per day.
- The types of active programs vary by job function and usually include standard office applications.
- The types of programs running in idle mode but not being actively used are normally the shell of the operating system (e.g., Windows Explorer), numerous housekeeping utilities and one or two user programs that are open but unattended.

In a previous study that characterized the self-defending capabilities of security software[4] it was shown that some programs are much more likely than others to cause performance or security problems. This latter weakness is troubling if the application in question has a high attack risk and has been left running in a system idle state, such as Internet Explorer.

As a follow-up to that paper we have assigned attack-risk values in this report to 21 standard programs or suites based on the number of times they have been implicated in a security attack and subsequent breach. This information was gathered from anti-viral, anti-Trojan, and anti-spyware sites and represents the average number of incidents that involve the application from 2003 to 2005. The values are naturally higher for applications that use network resources to provide functionality, such as email, browsers, or SMTP mail. However, attack-risk values are also affected by the security posture of a machine, including the patch level, degree of system hardening and exposure level due to the type of user running (with or without administrator rights). The applications shown in Table 1 are for the current release levels on Win 2K platforms. When the memory used by the application is leaking it is shown as a range, as in the case of MS-Word. Memory leaks are programming errors where resources are allocated but not properly freed resulting in the loss of system performance. In some instances, such a problem can consume considerable machine resources that would otherwise be available to defend a PC against an external attack. Also, when the total CPU time for a given application was higher than 65% utilization the PC was considered to be unfavorable for interactive use.

Table 1: Shows the average number of security vulnerabilities for standard office applications from the period 2003-2005.

Application	**Memory Used**	**Tot CPU**	**Vulnerabilities**	**Attack risk**
Iexplorer	12MB	6%	7880	3
MS-Word	8MB – 122MB	3% - 9%	1777	3
Windows Explorer	12.5 MB	0-1%	1360	3
Outlook	16 MB	3%	1063	3
SMTP	.5 – 2MB	1%	870	3
Imaging	5MB	1%	718	3
Notepad	.6MB	< 1%	639	3
Outlook Express	7MB	1-2%	271	2
Messenger	7MB	1-2%	249	2
Excel	5MB	2%	195	2
Anti-Virals	32 MB	28–95%	160	2
Anti-spyware	13 MB	5-85%	66	1
Acrobat Rdr	6MB	0-1%	27	1
WordPad	4MB	1%	25	1
Powerpoint	3MB	1%	23	1
Calculator	308KB	0-1%	21	1
MS Access	4MB	2-3%	12	1
Visio	19MB	3-4%	10	1
IDS/IPS	3-16MB	5-30%	1	1
TextPad	1MB	0-1%	1	1
MS Paint	2MB	1%	1	1

PC Usage Patterns and Heuristics

The following list was formed after analysis of the security vendor tracking data that showed PC applications with the greatest number of security incidents in the last couple of years. Note that the construction of heuristic attributes to manage application attack risk can also be used to trace surreptitious activity that may not show up in unsaved system logs. We consider this list to be a minimum set of requirements and have populated our SPD with the elements for further refinement as we expand the scope of this effort to cover other machine platforms and applications. Naturally as more of the heuristic requirements are missing from a given PC the threat level of that machine increases.

1. Anti-virus programs are enabled, up-to-date and running

2. Anti-malware programs are enabled, up-to-date and running
3. Host IDS/IPS programs are enabled, up-to-date, running and blocking on machines with high attack risk ratings
4. The system and IE are at the most recent patch level
5. Email attachment scanning is in use
6. Website surfing controls are in place
7. OS event logs are monitored for logon/logoff, audit failures, application errors and software configuration management (SCM) errors
8. Centralized storage for system, security and application events are used with periodic updates no greater than 5 minutes and real-time alerting enabled
9. Strong password checking is in use
10. Downloading and installing programs on PCs are blocked on the local system
11. Limited user profiles are in effect while surfing (no administrative rights permitted)
12. Default logins and unnecessary services have been removed or disabled
13. PC local time is auto-synchronized daily
14. All user programs are closed when not in use
15. All OS utilities are closed when not in use
16. All applications that run on a machine have been assigned an attack-risk rating
17. All users are automatically logged off after 10 minutes of machine idle time
18. Acceptable-use guidelines are in effect and the user is aware of them
19. Remote system management services are enabled and running on a PC.

Malware Catalogue and Heuristics

The following list was formed after analysis of the malware test data and construction of a realizable set of heuristic goals that should be utilized to provide proper security protection under three levels of application attack risk as shown in the previous table. Although not currently available in most Forensic Toolkits, we strongly urge the adoption of functional components by tool providers, that meet or exceed the self-protection goals found in the checklist. We consider this list to be a minimum set of requirements.

1. Security tools must provide OS, utility and self-protection for malware terminator functions at malware instantiation.
2. Security tools must provide OS, utility and self-protection for malware suspension functions at malware instantiation.
3. Security tools must have the ability to evaluate any CreateProcess or CreateThread function call whether invoked directly on a system or remotely via an intermediary service, such as a remote procedure call (RPC) using Runonce or Rundll32 or shell32 functions.
4. Provided self-protection functions must have minimum CPU cycle consumption, (< 5%).
5. Any non-OS related process that may set hooks (in any form) to the OS or any legitimate child processes or threads connected to the OS parent must be trapped and blocked.

6. Any programs issuing combinations of messages or API function calls such as *NtDeleteKey(), CloseHandle(), OpenProcess(), TerminateProcess(), JournalPlaybackProc(), DdeInitialize(), DdeImpersonateClient(), SetWindowsHookEx(), GetProcAddress(), DebugProc(), SysMsgProc(), WH_Keyboard(), SendMessage()* from non parent process functions, or other DLL injection methods into an OS process or thread via USER32.DLLs *LoadLibrary(), DdeConnect(), ImpersonateNamedPipeClient(), OpenProcessToken(), OpenThreadToken()* or via *CreateRemoteThread().* API function calls must be trapped and evaluated for potential malware.
7. Any attempt to delete, rename, change permissions on data files open and in-use by OS, utility or approved application processes as part of the company ghost image must be blocked.
8. Any attempt to delete, rename, change permissions, or otherwise prevent normal run-time usage of OS-related control data (registry or .ini files) and OS component files as part of the company ghost image must be blocked.
9. Selected self-protection components must be undetectable by normal service and process table viewing.
10. Security software must check every program upon instantiation for existence of known malware mutually exclusive identifiers and block their startup.
11. Network IDS tools must periodically sweep for SNMP trap listeners and promiscuous network analyzers that are not part of the company approved security system.
12. Security tools must distinguish between normal debuggers and potential hacking processes. For example, *DDESpy* or *Spy++ vs ProcKill* or *PWSHooker.*
13. All host and network logging tools must fail normally closed, i.e., close the user interfaces to avoid the perception that they are still protecting the system and issue security alerts in the event of a failure. This is typically managed by application exception handlers that trap various error conditions upon exit in each program.
14. Host based IDS/IPS tools must alert on unhandled OS exceptions.
15. Security tools must provide for in and out-of-band email notifications when triggered.
16. Security tools must securely log alerts on a host computer in the event that a network is down and cannot be sent to an upstream event collector.
17. Security tools must be able to securely send events to a multi-tiered set of collectors and must support multiple collectors per sensor.
18. Security tools must be able to securely send events to an enterprise-level database.
19. Security tools must be able to support at least three severity levels with different permissible actions at any severity level.
20. System activity to be monitored must include the following:
    a. Failed logon attempts due to incorrect password
    b. Failed logon attempts due to incorrect *user-id*
    c. Failed attempts to add/modify/change/delete a resource
    d. Successful modifications (add/modify/change/delete) of a resource
    e. Event altering on admin or other sensitive user-id access
    f. Event monitoring activity on a certain user-id

g. Monitoring password changes to any user-id
h. Changes to system values (i.e. Registry)
i. Any changes to user profiles
j. Any changes to critical system files

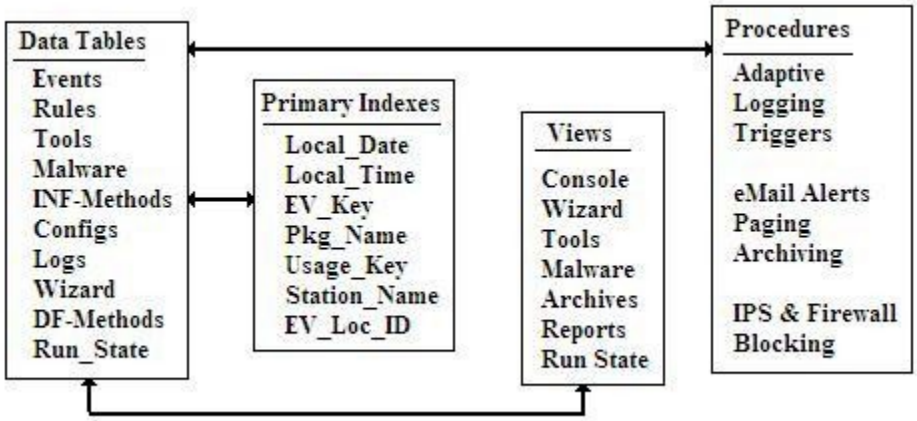
21. Security activity to be monitored must include the following
    a. Starting or stopping auditing/logging
    b. Deleting activity from logs
    c. Detecting modifications to logs
22. Application activity to monitor must include the following
    a. Ability to detect attacks targeted at any application, e.g. invalid URL's, FTP, *telnet* attacks
    b. Ability to detect rootkit installations
    c. Monitor capabilities for sensitive and specific directories
    d. Monitor capabilities for specific files
    e. Monitor capabilities for keyword content analysis
    f. Use of certain applications such as *Kazaa, Gnuetta, Gator*
    g. Network activity to monitor must include the following
    h. Network attacks directed to host (Network-type IDS signature, e.g. *SNORT* which detect attacks directed at host including DOS – denial-of-service – attacks and backdoors)
    i. Ability to log sessions and connections
    j. Analysis performed after decryption – i.e., works with SSL, IPSec, SKIP, SSL servers and IPSec in the DMZ (demilitarized zone)
23. Application activity to block must include the following
    a. Prevent the following behaviors by individual application (including OS functions like RPC)
    b. Permit preconfigured behavior policies for typical servers and desktops/laptops that protect all of the following items
        i. Unauthorized file additions
        ii. Unauthorized file modifications
        iii. Unauthorized file deletions
        iv. Unauthorized registry key additions
        v. Unauthorized registry key modifications
        vi. Unauthorized registry key deletions
        vii. Unauthorized network listening
        viii. Unauthorized network connections
        ix. Unauthorized system call activity
        x. Buffer overflows
        xi. Installation and launch of known unauthorized executable files (spyware, adware, Trojans, etc.)
        xii. Programmatic modification of existing executable files
        xiii. Launch of executable files from unauthorized process
        xiv. User privilege escalation
24. Security tools must have the ability to profile applications and auto-generate new behavior controls

25. Security tools must be able to monitor and restrict software installation and un-installation
26. Security tools must authenticate applications based on
    a. Fully-qualified path name
    b. Command line arguments that specify a machine name (e.g., Service Host)
    c. Entire lineage of process since boot
    d. User (if interactive)
    e. Combinations of the options above
27. Security tools must provide device installation/usage restrictions (e.g. prohibit wireless device connection such as Bluetooth, or restrict writing to A:, etc.)
28. Security tools must permit user-defined application controls and policies
29. Security tool agent functions that can be performed without connectivity to Management Server must have
    a. Prevention enforcement
    b. Queue logs until connectivity re-established
    c. Policy assignment
    d. Management server assignment
    e. SSL certificate management
    f. Log rollover/consolidation option settings.

Security-Pattern Database

An important goal of this project is to provide a repository for storing action based rules that can guide the decision making process for DF field investigations. As such the SPD was constructed as a lightweight database using *MySQL* with a minimum number of schema objects as shown below in Figure 1. Note that the relational structure has primary and secondary indexes that are based on many-to-many relationships in the tables. Due to the size restrictions for this report we are not able to publish the data dictionaries or constraint objects. However, they are available on the Web[9].

Figure 1: Shows the schema objects in the Security Pattern Database.



The relationship of table objects to views is as one would expect. Views such as the Wizard provide a front-end for decision support guidelines linked through events rules and other table elements as required. This is the main interface for the expert system that can be used at all stages of a DF investigation. The console view provides a user interface for linking disparate log files and tool outputs across package categories. The malware view provides category level relationships that cut across security events and alerting mechanisms as a function of event types and the likelihood of a Wizard match for recommended DF next step. The run-state view preserves as much of the across category data as was observable from network and system perspectives at the time of the triggering security event.

I will explain these relationships more fully in the next paper. At that time, I will also present results from simulated tests that show the advantages of having such a portable expert system available in the early stages of a DF investigation.

REFERENCES

1. “Electronic Crime Scene Investigation: A Guide for First Responders,” July 2001, Institute of Justice Web site at http://www.ojp.usdoj.gov/nij/pubs-sum/187736.htm
2. “Forensic Examination of Digital Evidence: A Guide for Law Enforcement”, NCJ 199408, April 2004, Special Report, National Institute of Justice, at http://www.ncjrs.gov/txtfiles1/nij/199408.pdf
3. Jim Lyle, “Computer Forensics Tool Testing at NIST”, July 2004, at http://www.cftt.nist.gov/documents/Amalfi-04.ppt
4. John Kerivan, Kenneth Brothers, “Self-Defending security software”, October, 2005, Milcom 2005 paper, at http://www.ngran.com/pages/819510/index.htm

5. John Kerivan, “Top 50 Malware Emulators,” Available from: http://www.ngran.com/pages/819506/index.htm
6. “PerfMon Sample: Demonstrates How to Monitor System Performance Using Performance Counters”, Microsoft MSDN Web site at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cssample/html/vcsamperfmonsamplecsharp.asp
7. “Comload is a rogue ActiveX control” See: http://www.spywareguide.com/product_show.php?id=470
8. Bruce Nordman, et. al., “User Guide to Power Management for PCs and Monitors”, January 1997, Energy Analysis Program, Ernest Orlando Lawrence Berkeley National Laboratory, University of California Available from: http://eetf.lbl.gov/EA/Reports/39466
9. John Kerivan, “SPD Data Dictionary,” Available from: http://www.ngran.com/pages/819510/index.htm