Online surveillance of critical computer systems through advanced host-based detection

Harmonized Anomaly Detection Techniques Thread

Wahab Hamou-Lhadj

Software Behaviour Analysis (SBA) Research Lab

Host-Based Anomaly Detection Working Group

Concordia University

Montreal, QC, Canada

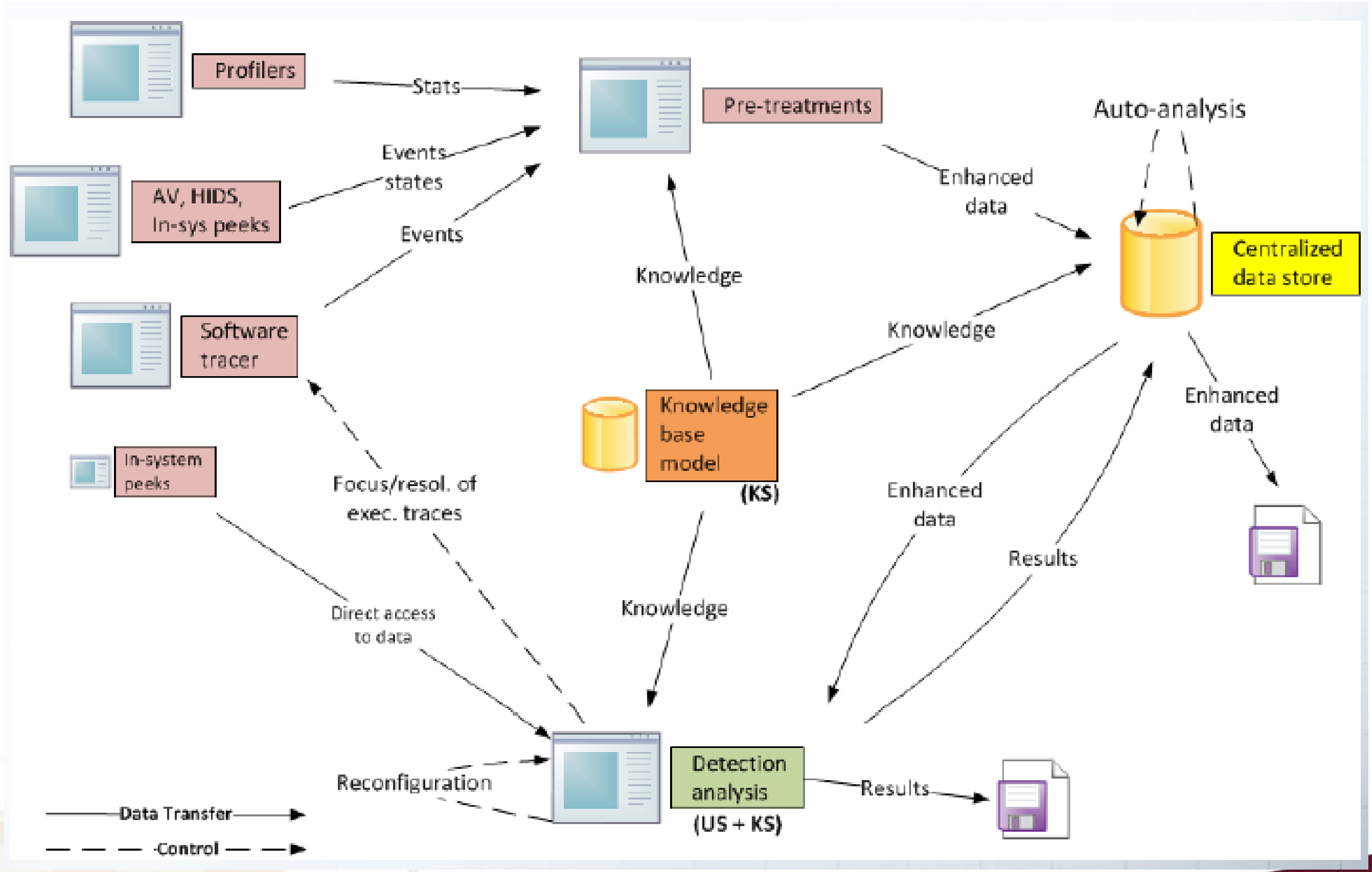
Feb. 06, 2013

DRDC, Valcartier, QC

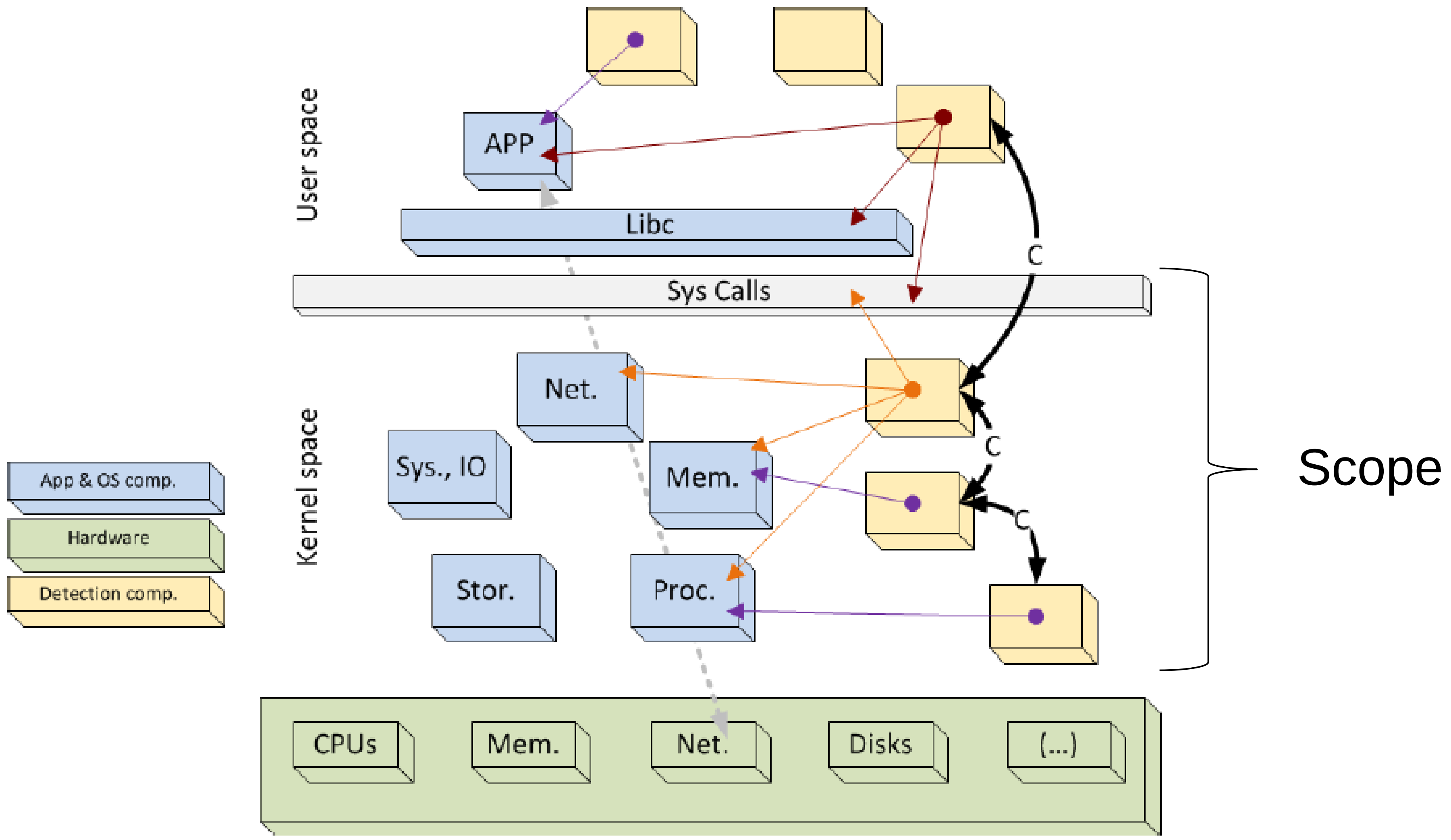
Goal

- Improve significantly the **online surveillance** of hosts
- Make use of a new paradigm: **health-based detection** of anomalies deep on the host
- Ensure good detection **accuracy**
- **Lower** the production of **false positive rates**
- Develop a **flexible, integrated,** and **configurable** tool
- Make recommendations for future research

Where we fit in the project



Detection layers



Thread Organization

- Three sub-threads
  - H1: Infrastructure and Integration (Shariyar Murtaza)
  - H2: Continuous monitoring (Shayan Eskandri, Amirreza Soudi, Amirhossein Zali)
  - H3: Trace-based models (Shariyar Murtaza, Afroza Sultana, another PhD?)

H1 – Infrastructure and Integration

H1.*	Milestone	Deadline
H1.1	Review existing work on anomaly detection techniques (include existing HIDS tools). Assess the applicability of existing techniques. Perform feasibility studies. Uncover technological and research gaps.	2012-04-01 2012-08-29
H1.2	Develop the multi-level analysis infrastructure, its scope, its structure, and rules of operation. Define the needed information for the infrastructure, the requirements and how to interoperate with the existing HIDS and the modules in Tracks 1, 2 and 4.	2012-09-01 2012-12-01
H1.3	Study the online activation and deactivation of probes and the knowledge required from Track 4 to guide it. Develop algorithms for the feedback-directed capability. Conduct experiments. Fine-tune and optimize. Disseminate results and findings.	2012-12-01 2013-09-01
H1.4	Develop algorithms for integrating the anomaly detection techniques. Conduct large-scale experiments. Examine the use of events generated by other HIDS. Evaluate effectiveness as compared to existing HIDS and AV. Disseminate results and findings.	2013-09-01 2014-04-01

H2 – Continuous Monitoring

H2.*	Milestone	Deadline
H2.1	Study the state of the art. Understand types of attacks. Survey and evaluate the different possible solutions	2012-04-29 2013-04-01
H2.2	Design and prototype new algorithms and mechanisms for detecting anomalies by correlating process execution and memory usage, uncovering authorized use of memory and hidden processes and other types of hidden malicious objects.	2013-04-01 2013-10-01
H2.3	Test under different conditions the proposed algorithms	2013-10-01 2014-01-01
H2.4	Disseminate results and findings.	2014-01-01 2014-03-01
H2.5	Improving the process-memory anomaly detection algorithm by examining the file system and network activities, correlate this information with process execution and memory usage	2014-03-01 2014-09-01
H2.6	Experimentation, validation, and optimization .	2014-09-01 2014-12-01
H2.7	Technology transfer and results dissemination.	2014-12-01 2015-04-01

H3 – Trace-Based Models

H3.*	Milestone	Deadline
H3.1	Review of trace-based host-based anomaly detection.	2012-04-01 2013-04-01
H3.2	Develop trace-based behavioural models of the OS. The model also considers system state, in-system peek information, and other information from existing HIDS.	2013-04-01 2013-10-01
H3.3	Evaluate the effectiveness of the new models on real-world systems . This task will require the monitors that will be developed in Tracks 1 and 2. Refine and fine tune.	2013-10-01 2014-01-01
H3.4	Disseminate results and findings.	2014-01-01 2014-03-01
H3.5	Develop file system based learning models. Use the new models with the process-based model to improve the detection capability.	2014-03-01 2014-09-01
H3.6	Conduct controlled experiments with real-world systems. Assess the effectiveness, accuracy, false positive rate and scalability.	2014-09-01 2015-01-01
H3.7	Disseminate results and findings.	2015-01-01 2015-04-01

System Call Sequence Modeling

- Models an application’s normal behavior from system calls sequences
- Detect deviations during operation of normal behaviour
- Techniques vary depending on:
    - Machine learning technique
    - Type of data (sys calls, arguments, both)
    - Offline vs. online techniques

Surveyed Approaches

- Sliding Windows (Forrest 1997, Warrender 1999)
- Rule Based (Tandon 2003, Petrussenko 2010)
- Neural Networks (Ghosh 1999)
- Hidden Markov Models (HMM) (Hoang 2003, Hu 2009, Khreich 2012)
- Finite State Automata (FSA) (Wagner 2001, Sekar 2001)
- Variable length N-gram (Wespi 1999, Jiang 2002)
- Statistical Techniques (Ye 2001, Burgess 2002)
- Call Stack Techniques (Feng 2003)
- Bag of System Call Technique (Kang 2005)
- Dataflow Based Models (Bhatkar 2006, Frossi 2009)
- Unsupervised Learning Based Models (Maggi 2010)
- Taint Enhanced Models (Cavallaro 2011)

Limitations

- Scalability and performance problems
- High false positive rates
- Lack of flexibility
- Work at the sys. call level only
- Lack of tools

Rootkit Detection

- Rootkit is a malware having several functionalities:
  - Stealth processing
  - Covert communication from system administrators.
  - Keystroke logging
  - Packet sniffing
  - Backdoor shell access
  - Remote attacking on networks

Rootkit Detection and Prevention Techniques

Host based techniques

Virtualization based techniques

External observer based techniques

Host Based Techniques

Techniques	Description
Kruegel et al. [2004]	Detect malicious LKMs using static analysis of LKM binaries
Kroah-Hartman [2004]	Load only RSA encrypted signed modules into memory
Secure boot [Parno et al., 2010;Jaeger et al.,2011].	Load a component if the hash is equal to a known-good value
Jestin et al. [2011a]	Cluster memory addresses to detect high memory addresses related to malicious system calls
AppArmor [Bauer, 2006] and SELinux [Smalley et al., 2002]	Limit access to the kernel by using policies
Strider Ghostbuster [Beck et al., 2005]	Identify hidden files and processes using normal views

Virtualization Based Techniques

Techniques	Description
[Garfinkel & Rosenblum, 2003]	Enforce HIDS policies from VMM, such as signature scan of memory, comparing commands, text comparison, etc.
[Petroni et al. 2007]	Use cryptographic hashes of code and the graph of function pointers to detect control flow (KOH) anomalies
[Wang et al. 2009]	Make a copy of hooks (pointers) to a write protected location, verify accesses and prevent KOH rootkits
[Seshadri et al., 2007] and [Riley et al., 2008]	Prevent kernel code from unauthorized modification and execution—targets KOH rootkits.
[Baliga et al. 2008]	Prevent KOH rootkits by using the policies based on process and file relationships
[Rhee et al. 2009]	Use policies for key data structure (e.g., modification through known functions) to detect DKOM rootkits
[Jiang et al. 2007]	A technique to run anti-malware programs from outside of an OS on a VM; e.g., antivirus

External Observer Based Techniques

- Copilot [Petroni et al., 2004], a PCI-card monitor, compares kernel text, LKM text and function pointers to detect KOH rootkits
- Gibraltar [Baliga et al., 2011] detect KOH and DKOM rootkits by using data structure invariants

Purpose	Invariant	Description
Detect hidden process	run-list $\subset$ all-tasks	run_list is a process list used by scheduler and all_task by others
Don’t let firewall disable	nf_hooks[2][1].next.hook == 0xc03295b0	To avoid redirection actual address is identified

Host Based Techniques: Tools Scanning Known Places

- Kstat—/dev/kmem vs. system.map
- Kern check—*system.map vs.* system call table
- Chkrootkit—logs and configs
- Rootkithunter—files, ports, processes
- Rkscan—Adore, Knark
- Knarkfinder—hidden processes
- Tripwire, Samhain and AIDE—checksum based integrity
- Sleuth Kit—File system forensics tool

Limitations

- Lack of integration
- Lack of mining capabilities
- No support of continuous learning
- Polling vs. inline monitoring
- High false positive rate
- Overhead caused by multiple configurations
- Scalability and performance problems

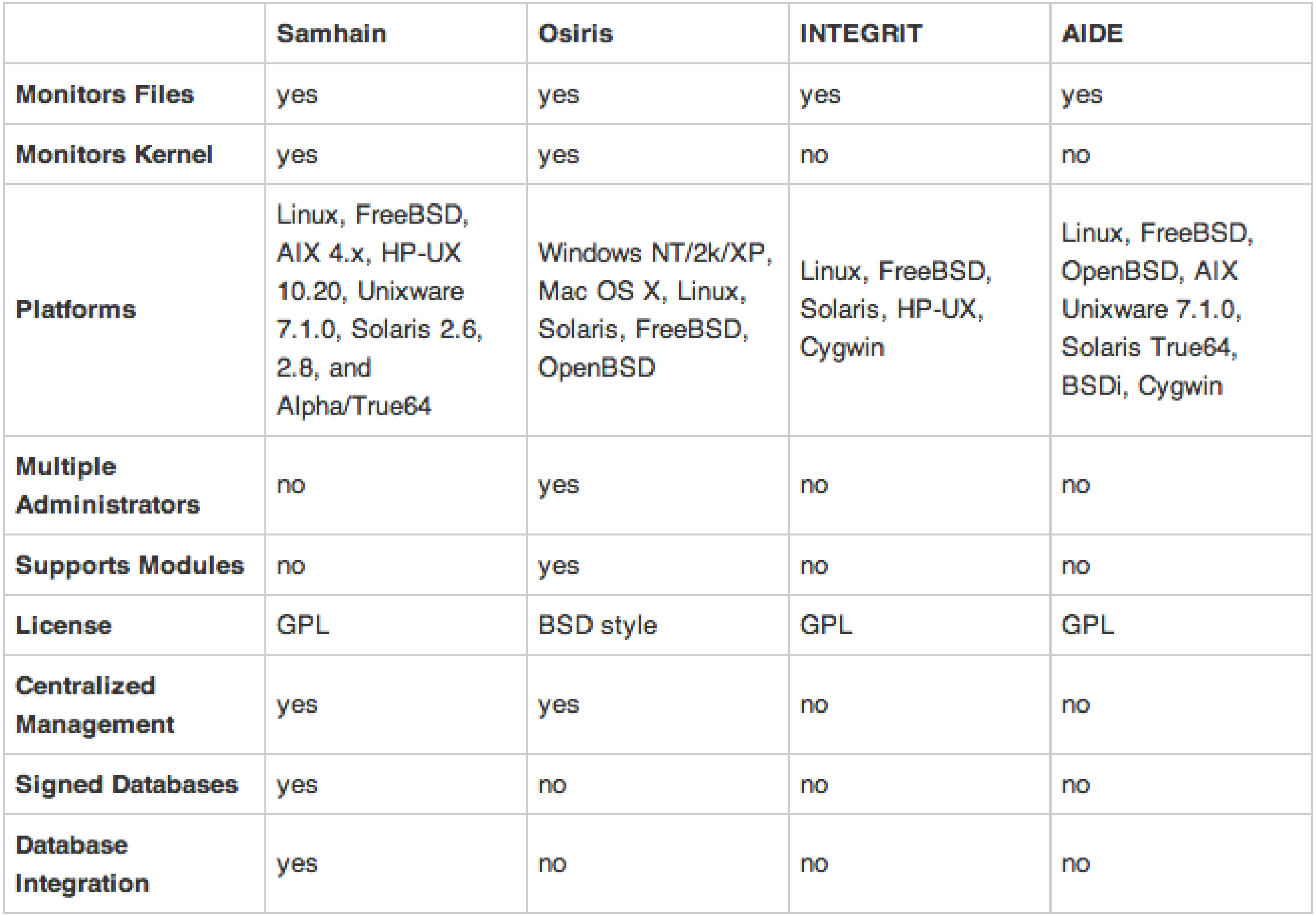
Promising Frameworks

- **Samhain:**
  - Developed in Germany
  - Open source
  - Client / Server architecture
  - Centralized management
  - Uses polling agents

Samhain capabilities

- Kernel integrity
- Open ports
- Process check
- Logfile monitoring/analysis
- SUID/SGID files
- Anti-tampering strategies

Comparison of Host Integrity Checkers



	Samhain	Osiris	INTEGRIT	AIDE
Monitors Files	yes	yes	yes	yes
Monitors Kernel	yes	yes	no	no
Platforms	Linux, FreeBSD, AIX 4.x, HP-UX 10.20, Unixware 7.1.0, Solaris 2.6, 2.8, and Alpha/True64	Windows NT/2k/XP, Mac OS X, Linux, Solaris, FreeBSD, OpenBSD	Linux, FreeBSD, Solaris, HP-UX, Cygwin	Linux, FreeBSD, OpenBSD, AIX Unixware 7.1.0, Solaris True64, BSDi, Cygwin
Multiple Administrators	no	yes	no	no
Supports Modules	no	yes	no	no
License	GPL	BSD style	GPL	GPL
Centralized Management	yes	yes	no	no
Signed Databases	yes	no	no	no
Database Integration	yes	no	no	no

ttp://www.symantec.com/connect/articles/host-integrity-monitoring-best-practices-deployment

Volatility framework

- Open collection of tools
- Works on Windows, Linux, and Mac
- Processing memory dumps
- Extensible API - plugin architecture
- Comprehensive coverage of file formats
- Fast and efficient algorithms
- Support large number of memory dumping tools
- Large community support
- Forensics/IR/malware focus

Volatility Framework Services

- Image information (date, time, CPU count)
- Running processes
- Process SIDs and environment variables
- Open network sockets
- Open network connections
- DLLs loaded for each process
- Open handles to all kernel/executive objects (files, keys, mutexes)
- OS kernel modules
- System call tables
- API hooks in user and kernel-mode
- Explore cached registry hives

Volatility Plugins for Malware Detection

- IDT - Prints the Interrupt Descriptor Table (IDT) addresses for one processor
- DriverIRP - Prints driver IRP (*I/O request packet)* function addresses
- kernel_hooks - Detects IAT (Import Address Table), EAT (Export Address Table) , and in-line hooks in kernel drivers instead of usermode modules
- malfind2 - Automates the process of finding and extracting (usually malicious) code injected into another process
- orphan_threads - Detects hidden system/kernel threads
- usermode_hooks2 - Detect IAT/EAT/Inline rootkit hooks in usermode processes

Future Directions

- In collaboration with the other groups, we intend to:
  - Continue to work on integrating the tools
  - Investigate the use of machine learning techniques
  - Investigate the use of kernel tracing techniques
  - Develop techniques based on control flow integrity and integrate them with other tools
  - Study the scalability and performance problems