Automatic Attack Signature Generation Systems: A Review

Sanmeet Kaur and Maninder Singh | Thapar University Patiala, India

The latest developments in intrusion detection systems, including Honeycyber, Hancock, Arbor, Auto-Sign, Argos, Hamsa, F-Sign, and a hybrid honeyfarm–based defense system, can be compared on the basis of their capability to detect novel attacks, signature generation method, suitability for multiple instances of worms, type of signature generated, attacks and worms covered, false alarm rates, and relative strengths and weaknesses.

Security is a big issue for all networks in today's enterprise environment. Hackers and intruders have successfully brought down enterprise networks and Web services. One method to secure network resources and communication over the Internet is intrusion detection. Intrusion detection systems (IDSs) monitor the state of a system or network and recognize and report any malicious activity or improper behavior.

IDSs can use either anomaly- or signature-based techniques to detect intrusions. In anomaly-based techniques, any deviation from the system's normal behavior profile is recognized and reported. In signature-based techniques, the signatures or patterns of known attacks are stored in the database's signature repository and compared to incoming packets. When the system finds a match, it can take actions such as logging, alerting, and dropping packets.

The problem with signature-based techniques is that they can't detect novel attacks whose signatures aren't available in the repository (0-day attacks). Novel attacks are difficult for both proactive and reactive security approaches to detect. Anomaly-based detection techniques can report about these attacks but have a high number of false alarms. 0-day discoveries require real-time attack handling and response, but this isn't feasible for manually interfaced systems. Systems need automated defense mechanisms to prevent such attacks.

Researchers have proposed many automatic attack detection and signature generation techniques to detect network intrusion. Automated signature generation systems can be classified into two broad categories—signature generation without attack detection and signature generation with attack detection. Signature generation without attack detection doesn't apply any attack detection mechanisms prior to signature generation. Honeycomb, Polygraph, Nemean, Earlybird, Hancock, Auto-Sign, and F-Sign fall under this category.

Systems with attack detection first detect an attack, then generate attack signatures. Autograph, PAYL (Payload-Based Anomaly Detector), PADS (Position-Aware Distribution Signatures), TaintCheck, Vigilante, ARBOR (Adaptive Response to Buffer Overflows), Argos, Hamsa, ShieldGen, Honeycyber, and Eudaemon fall under this category. (For more information on signature generation mechanisms, see the sidebar.)

In this article, we present a comparative analysis of several of these systems based on a review of literature.

Review Methodology

We searched an appropriate set of electronic databases and other sources to increase the probability of finding highly relevant articles. We performed this review by identifying primary studies, applying inclusion and exclusion criteria, and synthesizing the results.

We compare eight systems on the basis of important parameters, including 0-day attack detection, signature generation method, suitability for multiple instances of worms, types of signatures generated, attacks and worms detected, false alarm rates, and relative strengths and weaknesses.

Related Work in Signature Generation Mechanisms

Rashid Waraich reviewed 12 automated signature generation mechanisms—Nemean, Dynamic Taint Analysis, Honeycomb, IBM-94, Autograph, Paid, PADS (Position-Aware Distribution Signatures), PAYL (Payload-Based Anomaly Detector), Polygraph, StonyBrook, Dalhousie, and PISA—based on parameters such as system location, input, signature output format, worm detection mechanism, number of attack instances required as input, use of honeypot technology, usefulness against polymorphism, and quality of generated signature.[1]

In 2006, the European Network of Affine Honeypots project reviewed 15 signature generation systems—Honeycomb, Polygraph, Earlybird, Nemean, Autograph, PADS, PAYL, COVERS, DIRA, DOME, Minos, Paid, TaintCheck, Vigilante, and HoneyStat—comparing parameters such as attack detection mechanism, detected attack types, attack detection input, type of attack detection system, expected attack detection delay, signature type, applicability to polymorphic attack payload, expected signature generation delay, performance, attack detection, signature quality, and collaboration among sensors.[2]

However, no work in recent literature presents a survey of current attack detection and signature generation systems.

References

1. R. Waraich, *Automated Attack Signature Generation: A Survey*, tech. report, Swiss Federal Inst. Technology, Computer Engineering and Networks Laboratory, 2005.
2. *D1.2: Attack Detection and Signature Generation*, tech. report, Research Infrastructures Action, NoAH Project, May 2006; www.fp6-noah.org/publications/deliverables/D1.2.pdf.

Automated Signature Generation Systems

We analyzed Honeycyber, Hancock, ARBOR, AutoSign, Argos, Hamsa, F-Sign, and a hybrid honeyfarm–based system. Here, we describe these automated signature generation systems.

Honeycyber

Mohssen Mohammed and his colleagues designed Honeycyber, an automated signature generation system for 0-day polymorphic worms.[1] As Figure 1 shows, Honeycyber has a double honeynet system. The packets containing worms go to the first honeynet and make outbound connections. The internal translator directs them to the second honeynet. The worms make outbound connections from the second honeynet and are redirected to the first honeynet. Legitimate packets don't make outbound connections, so the packets that make outbound connections are considered malicious.

This system can also detect *polymorphic worms*, which vary their payloads on every infection attempt. Signature generation is based on the longest common substring method applied on multiple invariant substrings. Honeycyber can automatically detect new worms and isolate attack traffic from innocuous traffic. It generates both Snort-[2] and Bro-based[3] signatures and can generate signatures to match most polymorphic worm instances with low false positives and low false negatives.

Mohammed and his colleagues extended this architecture using principal component analysis to determine the most significant substrings shared between polymorphic worm instances for use as signatures.[4]

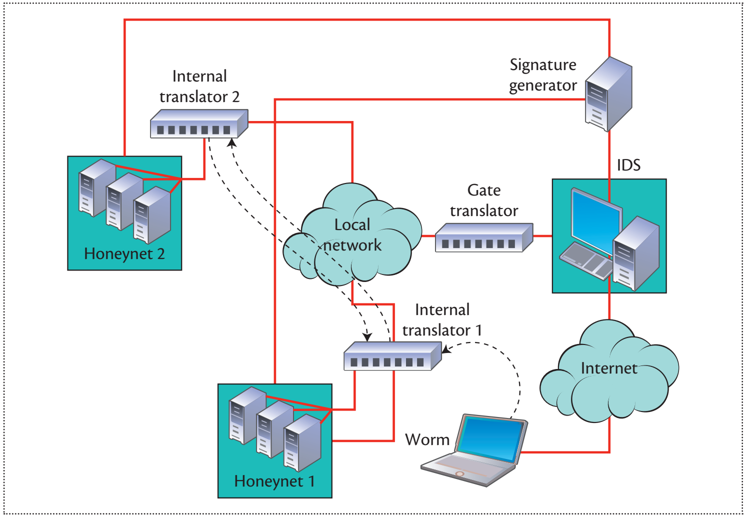

Figure 1. Honeycyber architecture.[1] Honeycyber's double honeynet system (with internal translators and a signature generator) is capable of generating Snort- and Bro-based intrusion detection system (IDS) signatures.

Hancock

Kent Griffin and his colleagues proposed Hancock, a system that automatically generates string signatures for malware detection.[5] Hancock was the first automatic string signature–generation system developed in Symantec Research Labs to automatically generate high-quality string signatures with minimal false positives and maximal malware coverage.

Each of Hancock's string signatures is a contiguous byte sequence that can match many variants of a malware family. The probability that a Hancock-generated string signature appears in any goodware program is very low. Each Hancock-generated string signature identifies as many malware programs as possible using

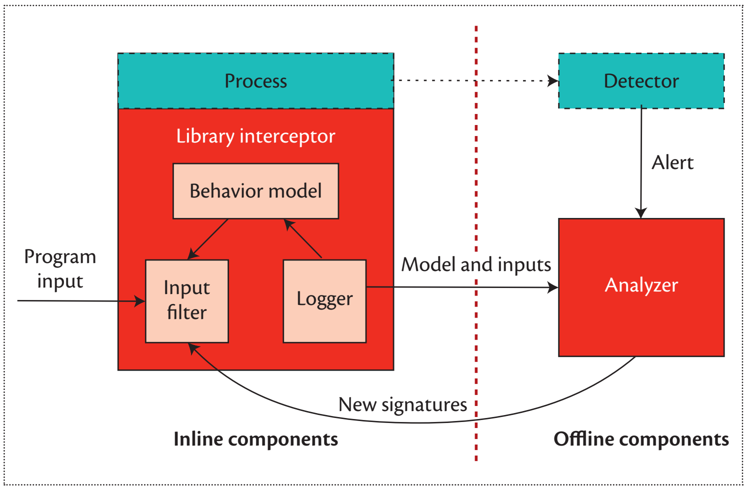

Figure 2. ARBOR architecture.[6] ARBOR contains inline components for input filtering and logging, whereas offline components are responsible for detection, analysis, and signature generation of buffer overflow attacks.

three types of heuristics to test a candidate signature's false positive rate—probability, disassembly, and diversity based. Probability- and disassembly-based heuristics filter candidate signatures extracted from malware files, and diversity-based heuristics select good signatures from among these candidates.

Hancock recursively unpacks malware files using Symantec's unpacking engine and rejects files that can't be unpacked. It examines every 48-byte code sequence in unpacked malware files and finds candidate signatures using probability- and disassembly-based heuristics. It then filters out byte sequences whose estimated occurrence probability in goodware programs is above a certain threshold (according to a precomputed goodware model). These byte sequences can be a part of standard library functions like I/O or graphics functions, which can be used by both goodware and malware authors. Hancock disassembles a set of malware files using IDA Pro.

Hancock then applies selection rules, based on the diversity principle, to the candidate signatures that passed the initial filtering step. Per the diversity principle, if the malware samples containing a candidate signature are similar to one another, the system will generate fewer false positives. Finally, Hancock generates string signatures consisting of multiple disjoint byte sequences rather than only one contiguous byte sequence.

To minimize the false positive rate, Hancock estimates the occurrence probability of arbitrary byte sequences in goodware programs using a fixed-order 5-gram Markov chain, a set of library code identification techniques, and diversity-based heuristics to find the similarities of the contexts in which a signature is used in various malware files. Combining these techniques, Hancock can automatically generate string signatures with a false positive rate below 0.1 percent.[5]

ARBOR

Zhenkai Liang and R. Sekar created ARBOR, a system that automates buffer overflow attack signatures.[6] This approach is based on the program behavior model. The authors argued that most existing buffer overflow detection techniques lead to repeated restarts of the victim application, interrupting service availability. ARBOR filters out attacks before they compromise the server's integrity, thereby allowing the server to continue to run without interruption. This significantly increases the availability of servers subjected to repeated attacks.

Figure 2 shows ARBOR's architecture. It's implemented using inline and offline components. Inline components are responsible for input filtering and logging whereas offline components perform tasks such as detection, analysis, and signature generation. The inline components hook into the execution environment of the protected process using library interception. The input filter intercepts all of the protected process's input actions. The inputs returned by these actions are then compared to the list of signatures currently deployed in the filter.

The system discards inputs matching any of these signatures and returns an error code to the protected process. If the input is associated with a TCP connection, the input filter breaks the connection to preserve the TCP protocol's semantics. The behavior model is a central component of ARBOR, used to make automatic filtering decisions based on knowledge gathered from the program itself rather than doing this encoding manually. Using library interception, ARBOR learns a protected process's behavior model. The logger records inputs for offline analysis. The offline components include a detector and an analyzer. The detector is responsible for attack detection; it promptly notifies the analyzer, which begins the attack signature generation process. The generated signature is then deployed in the input filter. This enables the system to drop future attacks before they compromise the protected process's integrity or availability.

ARBOR predicts attacks at the point of network input, resulting in reliable recovery. It also generates a generalized vulnerability-oriented signature from a single attack instance; this signature can be deployed at other sites to block attacks exploiting the same vulnerability. The system doesn't have any false positives but does have issues with false negatives, including attacks delivered through multiple packets, concurrent servers, message field overflows, denial-of-service attacks aimed at evading character distribution signatures, and addressing limitations.

Auto-Sign

Auto-Sign, designed by Gil Tahan and his colleagues, extracts unique signatures of malware executables to be used by high-speed malware-filtering devices based on deep packet inspection, and operates in real time.[7] It enables analysis at the binary level and doesn't require a semantic interpretation of code, making this technology generic—unaffected by CPU or platform changes.

Large executables comprise substantial amounts of code replicated across various instances of both benign and malware executables. To minimize the risk of false positive classification of benign executables as malware, Auto-Sign discards signature candidates that contain such replicated chunks of code. The authors claimed that this can control the number of false positives more effectively than increasing signature length. Auto-Sign focuses on various requirements pertaining to the signatures to be generated, such as low probability of the candidate signature appearing in the benign file, short length of the signature to cope with various IDS devices' storage limitations, and compliance with limitations of high-speed deep packet inspection devices to detect attacks in real time. These requirements should be well-defined to enable fully automatic generation.

Auto-Sign has two phases, namely, setup and signature generation. In the setup phase, two data structures—common function library (CFL) and common threat library (CTL)—are created. In the signature generation phase, signatures are generated, trimmed, and ranked, and the final signature is chosen on the basis of entropy, with a 3-gram representation. This method's main benefit is that it enables analysis at the binary level and thus doesn't require a semantic interpretation of code into function blocks. Auto-Sign needs to follow a more exhaustive and systematic methodology for building CFL repositories when generating signatures for high-throughput network security appliances.

Argos

Georgios Portokalidis and his colleagues' Argos is an emulator for fingerprinting 0-day attacks.[8] Argos is a containment environment that can handle worms as well as human-launched attacks. Argos is built on a fast x86 emulator that tracks network data throughout execution to identify invalid jump targets, function addresses, and instructions.

Figure 3 depicts Argos's architecture. Incoming traffic is logged in a trace database and fed to the unmodified application running on the emulator (Figure 3, step 1). In the emulator, a dynamic taint analysis detects when vulnerabilities are exploited to alter an application's control flow (step 2).

Argos considers data originating from an unsafe source in the network and data copied to memory as tainted. The new location is also considered tainted whenever it's used. Argos traces physical addresses rather than virtual addresses, hence reducing memory-mapping problems. When it detects a violation, it sounds an alarm that leads to a signature generation phase. To aid signature generation, Argos first dumps all tainted blocks and some additional information to file, with markers specifying the address that triggered the violation, the memory area it was pointing to, and so forth. To obtain additional information about the application, such as process identifier, executable name, open files, sockets, and so forth, the system injects its own shellcode to perform forensics (step 3). Argos was the first system to employ the means of attack (shellcode) for defensive purposes.

Argos then uses the dump of the memory blocks and the additional information obtained by system's shellcode to correlate with the network traces in the trace database (step 4). It submits the signature to a subsystem called SweetBait, which correlates signatures from different sites and refines signatures based on similarity (step 5). The final step is the automated use of the refined signature. Snort is attached to SweetBait to provide traffic signatures (step 6). Argos has used the Aho-Corasick pattern-matching algorithm to match network signatures.

The European Network of Affine Honeypots' testbed project recently used Argos as its virtual machine emulator.[9]

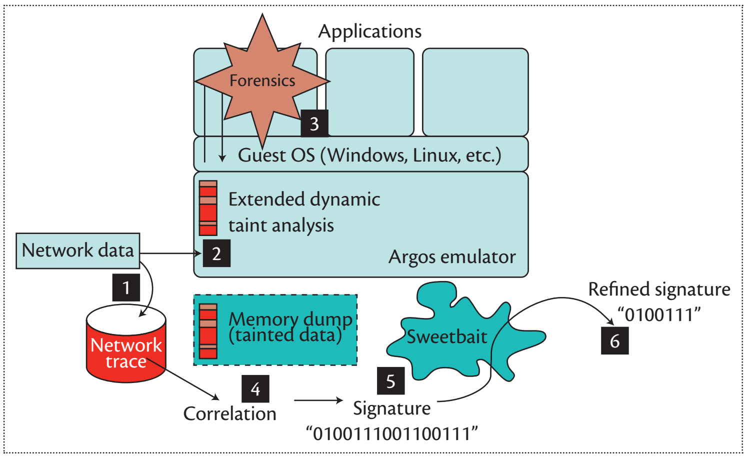

Figure 3. Argos architecture.[8] Argos is based on an x86 emulator that uses dynamic taint analysis on network traces and memory dumps to generate attack signatures.

Hamsa

Zhichun Li and his colleagues proposed Hamsa, a network-based automated signature generation system for polymorphic worms.[10] Hamsa is fast, noise tolerant,

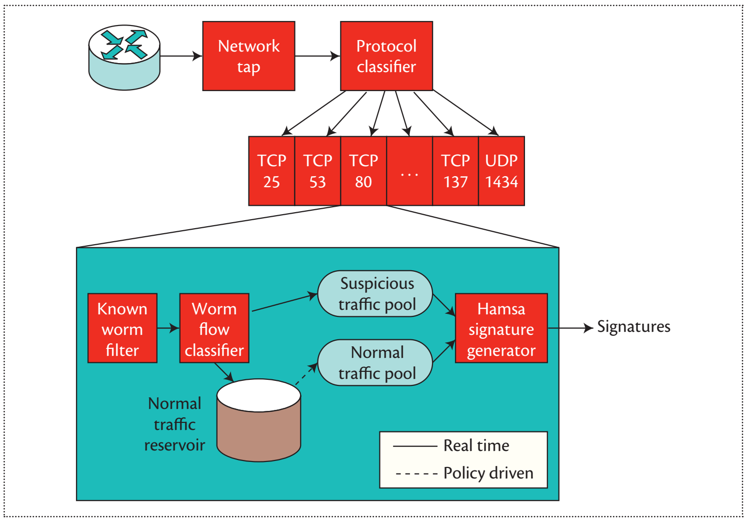

Figure 4. Hamsa architecture.[10] Network tap and protocol classifiers sniff and classify packets based on the port and protocol before separating the traffic into suspicious and normal flow pools. The signature generator then generates a content-based signature compatible with Snort and Bro.

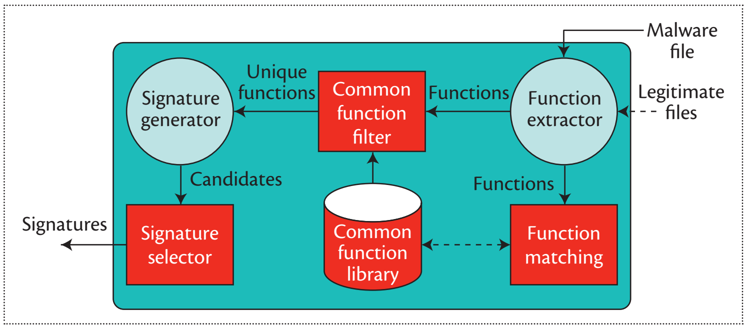

Figure 5. F-Sign architecture.[11] The function extractor helps in common function library creation, and the signature generator is responsible for generating signatures.

and attack resilient, analyzing polymorphic worms' invariant content.

Hamsa's architecture is similar to Autograph and Polygraph (see Figure 4). It sniffs traffic from networks, assembles packets to flows, and classifies flows based on protocols such as TCP, User Datagram Protocol, Internet Control Message Protocol, and port numbers. Then, for each protocol and port pair, Hamsa filters out the known worm samples and separates the flows into a suspicious pool or normal traffic using a worm flow classifier. Based on a normal traffic selection policy, some part of the normal traffic reservoir is selected to be the normal traffic pool. The signature generator then generates signatures using the suspicious and normal traffic pool. Hamsa focuses on content-based signatures because they treat the worms as strings of bytes and don't rely on protocol or server information.

Hamsa-generated signatures can be deployed easily in Snort or Bro IDSs. Hamsa significantly outperforms Polygraph in terms of efficiency, accuracy, and attack resilience.

F-Sign

Asaf Shabtai and his colleagues proposed F-Sign, an automatic and function-based signature generation systems for malware files.[11] F-Sign is designed to generate simple byte-string signatures that network-based IDSs can use to filter malware in real time. F-Sign employs an exhaustive and structured technique. It first extracts the malware's unique code from other segments of common and usually benign code, such as shared libraries, then generates signatures from malware such as worms, spyware, Trojan horses, and viruses. A human expert or an automated detection tool classifies the suspected files as benign or malicious.

F-Sign is a payload-based automated signature generation technique, capable of generating sensitive and specific signatures for malware of any size and type while minimizing false positives by analyzing the malware at the functional level and taking into account large common-code segments. It first creates a CFL that contains a representation of functions from standard libraries used by higher-level languages. After this, the signatures are generated for the entire malware corpus. Figure 5 shows the CFL creation and signature generation process.

The CFL can either be created by extracting functions using IDA Pro file disassembly or by extracting functions using a specialized state machine. The signature generation process for a malware file begins with identifying the internal functions, then each function is compared to the database of existing functions stored as CFL. Functions found in the CFL are marked as common. The remaining functions are candidates for generating a unique malware signature. The best candidate is chosen by entropy based on the length of candidate in bytes and frequency of appearances of a specific byte in the candidate function. The candidate with the highest entropy is selected as signature.

eDare—an early detection, alert, and response framework that provides malware-filtering services to network service providers, Internet service providers, and large enterprises—used F-Sign as its automatic signature generation module. Signatures generated by F-Sign comprise simple byte-strings, which can be used by high-speed, network-based malware-filtering devices. The false positive rate is calculated by counting signatures detected in the benign control group files. This system has low false positives for longer signature candidates and larger CFLs. F-Sign has been evaluated in conjunction with DefensePro, an IDS.

Honeyfarm-Based Defense against Internet Worms

Pragya Jain and Anjali Sardana proposed a hybrid approach that integrates anomaly and signature detection with honeypots.[12] This system makes use of all the approaches' advantages. Figure 6 shows this system's architecture. Signature-based detection is the first-level filter to detect known worm attacks. At the second level, an anomaly detector finds deviations from normal behavior. At the last level, honeypots are deployed to help detect 0-day attacks. The controller is responsible for traffic redirection among various honeypots deployed in the honeyfarm. In this approach, the longest common subsequence algorithm is used to generate signatures.

The detection rate of the hybrid honeyfarm–based approach is 81 percent, with a false alarm rate of 4 percent. The detection rate is remarkable compared to signature-based (32 percent), anomaly-based (34 percent), signature- and anomaly-based (61 percent), signature- and honeypot-based (46 percent), and anomaly- and honeypot-based (52 percent) techniques. There is an overall increase of 32.78 percent in the detection rate and a reduction of 33.3 percent in the false alarm rate compared to signature- and anomaly-based approaches.

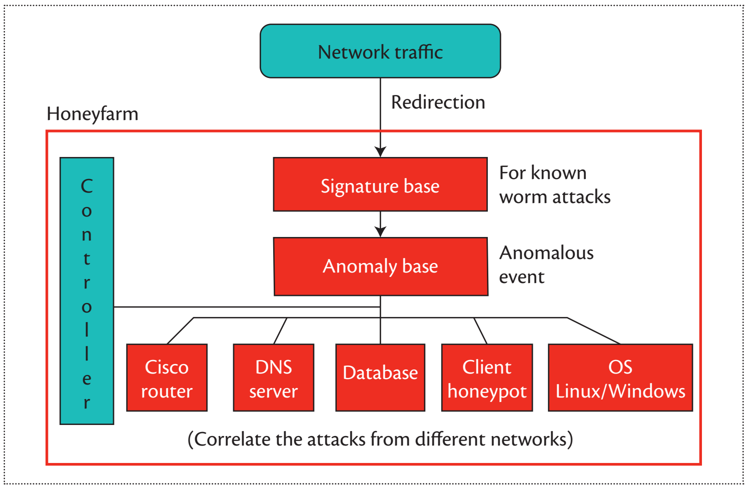

Figure 6. Architecture of the hybrid honeyfarm–based approach.[12] Signature-based detection is the first-level filter to detect known worm attacks. At the second level, an anomaly detector finds deviations from normal behavior. At the last level, honeypots are deployed to help detect 0-day attacks.

Comparative Analysis

Table 1 compares several aspects of these systems, including whether the system detects novel attacks prior to signature generation, the signature generation method, suitability for multiple instances of worms, type of signature generated, attacks and worms covered, false alarms rates, and relative strengths and weaknesses. Our findings are based on facts reported by authors of corresponding systems.

No single technique can detect all type of worms and attacks. The results we provide in this article are based solely on each of these tools' characteristics as described in literature. This method is a limitation of our study, and future work will involve a consistent and independent evaluation across all tools. Such an approach would make the results more comparable across the systems. ■

References

1. M.M.Z.E. Mohammed, H.A. Chan, and N. Ventura, "Honeycyber: Automated Signature Generation for Zero-Day Polymorphic Worms," *Proc. IEEE Military Communications Conference* (MILCOM 08), IEEE, 2008, pp. 1–6.
2. M. Roesch, "Snort—Lightweight Intrusion Detection for Networks," *Proc. 13th Conf. Systems Administration*, Usenix, 1999, pp. 229–238.
3. V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, nos. 23–24, 1999, pp. 2435–2463.
4. M.M.Z.E. Mohammed et al., "Detection of Zero-Day Polymorphic Worms Using Principal Component Analysis," *Proc. 6th Int'l Conf. Networking and Services*, IEEE CS, 2010, pp. 277–281.
5. K. Griffin et al., "Automatic Generation of String Signatures for Malware Detection," *Proc. 12th Int'l Symp. Recent Advances in Intrusion Detection*, Springer-Verlag, 2009, pp. 101–120.
6. Z. Liang and R. Sekar, "Automatic Generation of Buffer Overflow Attack Signatures: An Approach Based on Program Behavior Models," *Proc. 21st Ann. Computer Security Applications Conf.*, IEEE CS, 2005, pp. 215–224.
7. G. Tahan et al., "Auto-Sign: An Automatic Signature Generator for High-Speed Malware Filtering Devices," *J. Computer Virology*, vol. 6, no. 2, 2010, pp. 91–103.
8. G. Portokalidis, A. Slowinska, and H. Bos, "Argos: An Emulator for Fingerprinting Zero-Day Attack," *Proc. Int'l Conf. ACM SIGOPS EUROSYS*, ACM, 2006, pp. 15–28.
9. J. Kohlrausch, "Experiences with the NoAH Honeynet Testbed to Detect New Internet Worms," *Proc. 5th Int'l Conf. IT Security Incident Management and IT Forensics*, IEEE CS, 2009, pp. 13–26.
10. Z. Li et al., "Hamsa: Fast Signature Generation for Zero-Day Polymorphic Worms with Provable Attack Resilience," *Proc. IEEE Symp. Security and Privacy* (S&P 06), IEEE CS, 2006, pp. 32–47.
11. A. Shabtai, E. Menahem, and Y. Elovici, "F-Sign: Automatic, Function-Based Signature Generation for

Table 1. Comparative Analysis of the Systems under Study.

System	Ø-day attack detection	Signature generation method	Suitability for multiple instances of worms	Type of signatures generated
Honeycyber	Yes	Colored set size for string matching	Suitable for polymorphic worms	Bro- and Snort-based signatures
Hancock	Partial—can do so if signature covers many sample files in a malware family	Uses probability-, disassembly-, and diversity-based string signatures with contiguous byte sequence	Partial—can detect little variation in code; not suitable for high degree of polymorphism	Single-component and multicomponent signatures for antiviruses
ARBOR	Yes, restricted to buffer overflow attacks	Address space randomization	Suitable for polymorphic worms	Buffer overflow–related signatures
Auto-Sign	No	By ranking the candidate signature based on entropy, probability, and distance	No	Signatures compatible with NIDS/NIPS operating as malware-filtering devices
Argos	Yes	Dynamic taint analysis, longest common substring, and critical exploit string detection	Suitable for polymorphic worms	Snort-based signatures
Hamsa	Yes	Content-based token extraction	Suitable for polymorphic worms	Snort- and Bro-based signatures
F-Sign	No	Entropy-based selection after creating common function library (CFL)	Suitable for partially obfuscated malware having invariant codes but not for fully obfuscated malware	Compatible with eDare (early detection, alert, and response) framework
Hybrid honeyfarm–based approach	Yes	Longest common subsequence along with protocol-based packet header anomaly detection technique	No	Local signature detection and generation engine

Malware," *IEEE Trans. Systems, Man, and Cybernetics—Part C: Applications and Reviews*, vol. 41, no. 4, 2011, pp. 494–508.

12. P. Jain and A. Sardana, "Defending against Internet Worms Using Honeyfarm," *Proc. CUBE Int'l Information Technology Conference* (CUBE 12), 2012, ACM, pp. 795–800.

Sanmeet Kaur is an assistant professor in the School of Mathematics and Computer Applications at Thapar University Patiala, Punjab, India, where she's also pursuing a PhD in intrusion detection. Her research interests include network security, software testing, and software engineering. Kaur received an ME in software engineering from Thapar University Patiala. Contact her at sanmeet.bhatia@thapar.edu.

Maninder Singh is an associate professor in and head of the Computer Science and Engineering Department at Thapar University. His research interests include network security and grid computing, and he is a torchbearer for the open source community. Singh received a PhD in network security from Thapar University. Contact him at msingh@thapar.edu.

Attacks and worms covered	False alarm rate	Strengths	Weaknesses
Polymorphic worms	Low false positive (FP) rate	Double honeynet to detect polymorphic worms	Overhead is more; double honeynet takes more processing time
Malware detection	Sufficiently low FPs (below 0.1 percent)	Provides scalable goodware modeling technique; generates multicomponent signatures, which are more efficient than single-component signatures; low FP rates below 0.1 percent	Less coverage; not suitable for highly polymorphic malware
Ten real-world vulnerabilities including WU-FTPD, Apache SSL, ntpd, IRCd, Samba, and passlogd	No FPs; false negatives (FNs) possible with fragmented attacks, concurrent servers, and message field overflows	Effective for real-world buffer overflow attacks; ensures high availability of servers; low runtime overheads; can work with COTS without access to source code; lesser attack samples required for high-quality signature generation	Generates FN alarms in fragmented attack packets, concurrent servers, DoS attacks aimed at evading character distribution signatures, and message field overflows; stand-alone system doesn't communicate with other systems
Malware executables including worm emails, virus, Trojans, email flooder, denial-of-service (DoS) attack, exploits, worms, and P2P attacks	Low FP rates	Detects attacks from larger malware executable; reduces false positives by eliminating benign traffic signature candidates' platform-independent analysis as it works at binary level	Doesn't address the issue of generating composite signatures
Scalper, sadmind/IIS, Welchia, Poxdar, Sasser, Gaobot.ali, Zotob, Blaster, Mytob-CF, Dopbot-A	No false alarms	No FPs; cost effective	Doesn't generate self-certifying alerts
Code Red II, Apache-Knacker, ATPhttpd, CLET, Tapion	Low and bounded FPs and FNs	Faster than previous token-based techniques like Polygraph	
Malware detection	Low FP rates, provided CFL size is large (below 0.4 percent with 1,675 Mbytes CFL)	Suitable for high-speed malware-filtering devices; able to tackle allergy attacks against automated signature generation	Not suitable for fully obfuscated or polymorphic code
Metasploit-generated attack patterns	Four percent false alarm rate	High detection rate; hybrid approach includes advantages of anomaly- and signature-based techniques	High initial setup time; false alarm rate is substantial