DOI 10.1007/s10559-019-00194-9
Cybernetics and Systems Analysis, Vol. 55, No. 5, September, 2019

SOFTWARE–HARDWARE SYSTEMS

AN OVERVIEW OF THE MODERN METHODS OF SECURITY AND PROTECTION OF SOFTWARE SYSTEMS

O. O. Letychevskyi,[1†] **V. S. Peschanenko,**[2] **Y. V. Hryniuk,**[1‡] **V. Yu. Radchenko,**[3] **and V. M. Yakovlev**[1††]

UDC 004.05, 004.4[2+9], 004.94, 519.7

Abstract. *Security and protection of software resources are one of the most important issues in the IT industry since attackers' actions become increasingly sophisticated and losses caused by cyberattacks are growing. Traditional methods of cyberattack prevention become inefficient; therefore, development of new methods and tools to secure software resources becomes of essential need. The studies that are based on formal methods with the use of modern algebraic theories are especially interesting and promising.*

Keywords: *algebraic modeling, behavior algebra, cybersecurity, insertion programming, formal methods, symbolic methods, symbolic modeling, vulnerability detection.*

Security and protection of software resources is one of the most important problems that are actively discussed at international conferences. Occurrence of ransomware network software indicates the unprecedented level of perfection of attack technologies. Malefactors are increasingly skillful in avoiding their detection and efficiently use security gaps. The importance of this issue is emphasized by the fact that according to [1], losses caused by cyberattacks made three trillion US dollars in 2015 and are estimated to make six trillion US dollars in 2021.

In the paper, we will review the main concepts and important problems in security and protection. We will pay special attention to the means of cybersecurity that utilize formal methods based on the modern algebraic theories.

MAIN CONCEPTS AND CLASSIFICATION OF CYBERATTACKS

A cyberattack or a hacker attack is a set of malefactors' (hackers') actions in order to intervene the operation of software system to capture data, obtain control or put resources of computer media out of order. An attack uses system's vulnerability caused by wrong architecture or a code error.

Software systems that have errors are vulnerable to attacks. For example, the database of Linux operating system has over 90,000 open errors. The issue related to possible use of these errors as vulnerabilities to attacks is more important than fixing a bug. The term "exploit" by definition is an attack that exploits some vulnerability.

An attack begins with penetration into a remote computer, which can be both a desktop and a component of a network or cloud environment. Respectively, attacks are classified by objects (a desktop, web attacks, cloud environment, blockchain). Attacks can use malware that is activated during penetration into user's computer. Table 1 shows the main types of malware according to the ways of penetration [2].

[1]V. M. Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine, Kyiv, Ukraine, [†]*oleksandr.letychevskyi@garuda.ai*; [‡]*yaroslav.hryniuk@garuda.ai*; [††]*victor.yakovlev@garuda.ai* and *victor.yakovlev@ukr.net*. [2]Kherson Sate University, *volodymyr.peschanenko@garuda.ai*. [3]Garuda AI, Kyiv, Ukraine, *viktor.radchenko@garuda.ai.*
Translated from Kibernetika i Sistemnyi Analiz, No. 5, September–October, 2019, pp. 156–169. Original article submitted May 20, 2019.

TABLE 1

Malware Type	Malware Description
Virus	A program that can join another program and be executed with its use in the "infected" environment
Worm	Unlike a virus, this program can be executed independently and replicate its duplicates on other computer resources
Trojan	A program that looks like useful one and performs a malicious attack at the end of its life cycle
Spyware	A type of malware that tracks user's actions in order to obtain important information such as password, account numbers, etc.
Ransomware	A program, which is secretly installed on user's computer and encrypts data in order to demand a ransom for correct decryption
Scareware	Harmful computer programs intended to enforce user to buy and load unnecessary and potentially harmful software, for example, fake virus protection
Bot	A program that simulates user's actions by means of appropriate interfaces according to an established schedule or an algorithm with the purpose of network attacks, fraud, distribution of unnecessary advertising and other overload of Internet channel. Ddos attack is an example of such program
Rootkits	A set of software that allow an unauthorized user to gain control over the computer system without detection in order to steal data

Attacks against web sites or web services, distributed and local networks, cloud environment consider and use the special features related to data transportation to network nodes. Security problems in this field are caused by different types of attacks that can be applied to violate normal operation of network systems (active attacks) or steal information transmitted by the network (passive attacks). According to [3], there are types of network attacks that violate normal operation of network systems in one way or another (deceleration and failures, losses and distortion of information, information capturing). Table 2 shows available types of web attacks.

TRADITIONAL MEANS OF NETWORK AND ENVIRONMENT PROTECTION

There are the following varieties of the modern traditional security systems.

- Intruder Detection Systems (IDS) — programs that watch system's behavior and fix deviation of its interior and exterior manifestations from the established security policy. They are usually located on network nodes, analyzing traffic and collecting information from subsystems.
- Sandboxes — systems that can run suspicious programs in a separate environment on the appropriate virtual machine. When a program is run in an emulation environment, it cannot damage user's system, and attack will be detected by the emulator.
- Proactive Systems (HIPS, Host-Based Intrusion Prevention System) — systems equipped with an open rule table. Based on this table, the system allows or prohibits certain actions to applications. The system is oriented to an interaction dialog with the user and impose high demands on their competence.

Many systems use the heuristic analysis technology, which allows detecting sections of a suspicious code that causes harmful activity.

Companies Check Point, CISCO, and McAffee are the main players in the market of traditional protection against cyberattacks.

Check Point company has been working in this field since 1993 and develops multi-purpose products on software system protection for both cloud environment and mobile devices. Yield Check Point Software Blade Architecture [4] (Fig. 1) is one of the recent products that provides complex configuring of necessary components of cybersecurity.

CISCO Company is over 20 years in the market. During this time, modern technologies, a huge number of devices and sensors considerably strengthened the set of protection facilities. The company is a famous developer of both hardware and software, from routers and data centers to behavioral analyzers.

TABLE 2

Attack Name	Attack Description
Spoofing	An outside program that uses false identification and forces the system being attacked to change web topology
Modification	Malicious system that changes routing in the network
Wormhole	System that attacks receives a data package and reroutes it to another place thus simulating the shortest path in the network for the system under attack
Fabrication	Generation of a false message by a malicious system for network routing, which impairs operation of routers
Denial-of-Service (Ddos)	Deceleration or violation of the operation of system under attack because of an overload of network traffic and an excessive amount of requests that simultaneously arrive in the system
Sinkhole	Generating obstacles that do not allow the system under attack to receive complete and correct information. The purpose is selective modification, re-routing and information loss
Sybil	An attack with the use of an infinite set of malicious network nodes, during which one such node transmits secret keys to another ones
Traffic analysis	Analysis of the amount of data transmitted between nodes being attacked
Eavesdropping	Attacks that are often carried out in mobile communication networks in order to abduct significant information, in particular secret keys
Monitoring	Interception of significant information in the network without its modification
Black hole	The system that attacks uses routing protocols to present itself as the best node for data routing for system being attacked
Rushing	The system that attacks replace packages transmitted between nodes that are attacked, then duplicates these packages again and again, making an impression of a highly loaded system
Replay	System that attacks can repeat network packages and/or delay them, and steal confidential information (such as passwords) during processing
Byzantine	Nodes of the system that attacks are embedded in between nodes that are attacked, by using non-optimal routes, creating cycles in routes, or selectively ”losing” packages
Location disclosure	Gathering information about nodes that are attacked and routes, monitoring the traffic with further application of other types of attacks

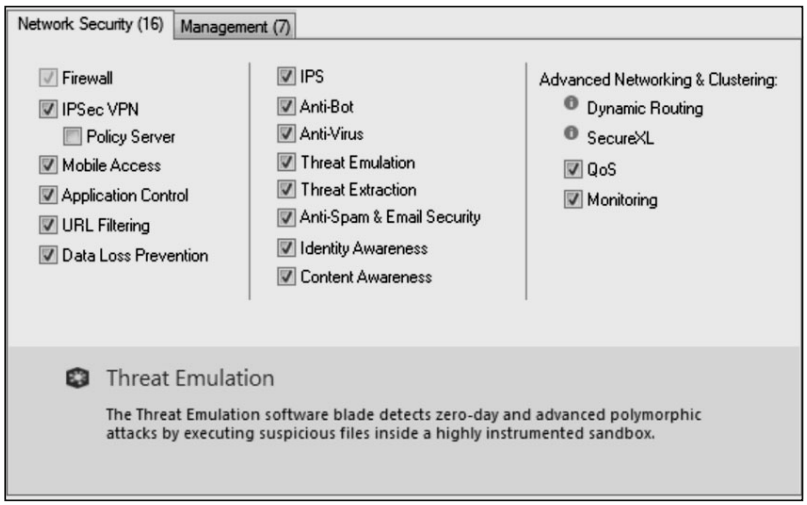

Fig. 1. Setup window of Check Point Software Blade Architecture system.

Mcafee Company, which mostly focuses on home computers, is famous due to its antiviruses, security of tablets and smartphones.

Despite the use of advanced technologies in software and hardware, modern security systems are not an ideal protection of software environment and have a number of shortcomings. First of all, this is a considerable quantity of wrong detections, especially in case of high-sensitive analyzer. This feature is mostly inherent in heuristic method. A delay in attack detection is caused by considerable duration of detection process. This can happen when attacker uses contrivances such as obfuscation (masking the behavior by introducing a "dead code"), polymorphism (a malicious file that changes itself during propagation, start) and file packing. As a result, a well-known attack may not be detected at all during file check. Isolated program run also has disadvantages, it needs quite a lot of time and computer resources of the user, which adversely affects high-speed performance in running daily operations. Moreover, modern malicious programs are capable to detect execution in a simulated environment and to stop the operation in it.

Throughout the last five years, formal methods have been considerably developed. They are a basis of cybersecurity software, due to the development of tools such as problem solvers and machine of automatic proof of theorems.

FORMAL METHODS APPLIED IN CYBERSECURITY METHODS

The following two aspect are considered to resolve the issue of security and protection or cybersecurity problem.

1. Detecting vulnerabilities in the available program presented in the form of binary or source code of the program.

2. Detecting an attack in the mode of operation of software resources in a certain computer environment such as a separate computer, computer network or cloud environment.

In 2016, DARPA conducted so-called Cyber Grand Challenge [5], a competition of a number of software systems for detection of vulnerabilities in the proposed programs. 40 commands with their systems competed in the ability to detect the maximum quantity of vulnerabilities in programs during a limited time interval. The main winners were as follows:

- Mayhem system [6] by Forallsecure command from Pittsburg that became the winner. The system applied symbolic performance of programs with the use of Microsoft Z3 solution system and symbolic memory model;
- Xandra System [7] by a team from New York, where their own vulnerability detector was used;
- Mechanical Phish System [8], where fuzzing method combined with symbolic execution was used.

A special feature of this competition was that each winner applied algebraic methods and deductive means (problem solvers and automatic proof of theorems).

Amazon is one of the leading companies that employ formal methods to protect cloud environment against exterior attacks. It has over 1500 services; therefore, the level of interference danger is rather high. The company very actively applies formal methods for precise detection of attacks, in particular, more than ten deductive tools.

The fuzzing method is the main tool to find unknown vulnerabilities (zero-day vulnerabilities). The idea of the method is that critical values of variables in the program that may cause vulnerable actions are generated during execution of different scenarios of program implementation. One of the most famous fuzzing machines is AFL (American Fuzzy Lop) created by Google [9].

Generally, formal methods can be used as follows.

Analysis of a Sequence of System Calls. System call is a program way of request for services at an operating system. Examples of system calls are functions for file reading/writing or networking. The majority of programs use system calls for a dialog with the network, saving user's settings, etc. An attack detection system intercepts and generates sequences of system calls for a specific process. There are also open arrays with sequences of system calls that correspond to specific vulnerabilities. With the use of available sequences, algorithms are formed that determine if these sequences are potential attacks.

To create vulnerability classification algorithms [10], different formal methods are used. One of them is based on the algorithm with the use of so-called windows that move along a sequence of system calls. This is a rather popular algorithm since window's fixed size allows high-speed performance control and use of different statistical methods. It is possible to apply hidden Markov models, recurrent neural networks; however, at the same time, choosing parameters for these methods is a challenge and a compromise since it is necessary to avoid over-training and to reduce the utilized

resources. The language approach provides generating simple n-grams like in text processing. However, dimension of the potential space of vectors becomes very high even for small n-grams.

Universality is an advantage of the analysis of sequence of system calls since system calls are a unified interface of operation between the program and operating system. It is also possible to detect zero-day attacks, i.e., attacks that have not been detected yet. This approach also has shortcomings because of small amount of data for training, and such attack detection systems have the big percent of errors of first kind and influence high-speed performance of the programs as a whole, since there are some indirect costs during reading-out of system calls.

Attack Detection Systems Based on Symbolic Computing. Symbolic execution consists of code interpretation where symbols that represent arbitrary values instead of specific entries, for example, strings or digits, are used during program execution. Computing in this case is the same as in specific execution, except for the fact that the values processed by the program can be a symbolic expression over input symbols. To this end, symbolic execution generates all possible paths of program execution. The set of input data that form such path of execution is deduced for each path. These data are useful for diagnostics of both low-level fails and high-level properties of the program.

There is often no access to the source code of the program being executed; therefore, the program in disassembled form is analyzed. Disassembling is the process of translation of the program from machine codes into assembler language (instructions of the processor). Thus, symbolic computing is carried out on the basis of assembly representation of the program.

One of the ways of using the results of symbolic computing is identifying sensitive places where unreliable tributes [11] are used. Unreliable data are data that are fed to program's input and are not subject to appropriate handling; handling can be different depending on the data and places of their use. Sensitive points are program's points that can cause anomalous behavior of the program, privileged access, etc. under some circumstances. For example, call of the function "system" launches the program that can be described by input argument. If the value that is transmitted to "system" is processed incorrectly, malefactor can launch any program, which is a serious vulnerability. Such approach has proved to be efficient in detecting vulnerabilities related to overflow of buffer, overflow of integer variables, remote run of the code.

However, in case of symbolic computing, the number of possible paths increases exponentially with regard to the number of instructions even in small programs. When elementary cases are disregarded, complete analysis may need great amount of time and computing capacities. Different techniques that optimize performance but thus decrease the accuracy of the method are applied to overcome this. Moreover, due to the use of symbolic computing, it is possible to generate data that confirm vulnerability by resolving conditions of symbolic path.

Thus, this approach is rather efficient for static analysis of files being executed. It allows us to generate paths of unreliable data and to detect vulnerabilities, provides a potential possibility of automatic acquisition of big arrays of test data for permanent testing of software system during its development.

Attack Detection Systems Based on Concrete and Symbolic Computing. These systems appeared in response to constraints of systems that are based on symbolic computing only and are called concolic computing. This computing is a hybrid of symbolic computing and concrete execution. Using symbolic computing allows generating all possible paths of program's execution, as well as input data that correspond to such paths. This data is used to test and find vulnerabilities in the program. However, such approach has shortcomings. For example, if there are cycles and recursive calls in the program, a great amount of time and resources may be necessary for symbolic computations. Concolic computing partially utilize concrete data rather than their symbolic representation in order to overcome issues of symbolic execution [12].

The process of concolic computing begins with adjustment of the environment for program execution. In modern operating systems, there is a possibility to launch the program in a special mode so that the program is executed step-by-step, command-by-command, and it is possible to read out the value of all registers and memory value at each step. Different techniques are then used. One of the possible techniques is creating the path of input data, for example, a new constraint is added each time when there is a jump instruction, which depends on input data [13]. Further, test data are generated on the basis of collected constraints, which can show possible vulnerability in systems on the basis of symbolic computing. This data can also be used to find more possible paths of program execution during specific execution.

Thus, the purpose of systems based on concolic computing is removal of the constraints available in the systems, which are based on purely symbolic computations. An example of such constraints is exponential growth of the number

of paths with respect to the number of instructions. These systems also allow processing the cases where there are cycles and recursive calls in the program. Examples of implementation of such systems proved their efficiency in finding vulnerabilities such as buffer overflow, various vulnerabilities related to integer arithmetic, division by zero, reference to zero pointer. At the same time, it is more complicated to develop and support such system as compared with a system that is based on symbolic computing only since it is necessary to generate an infrastructure to launch programs and execute them with specific data. To this end, it is necessary to choose software that would provide necessary testing environment. Specific features of the program may demand a special software. It means that vulnerability detection system needs to be adjusted for each program. Moreover, initial set of test data is developed manually and cannot be automated. This means that such systems are developed as an addition to available programs, which need to be tested and cannot be general-purpose systems.

Vulnerability Detection in a Reused Code. Software developers copy the code for reusing. At the same time, when copying a code, the programmer also copies its vulnerability, which respectively influences security of the software system as a whole [10]. For example, a quarter of the code of the kernel of open-source operating system Linux repeats; some segments even repeat up to eight times. 145 entries with vulnerability in the copied code have been detected and not corrected yet. Using the information about known vulnerabilities, malefactors can find all the points with similar vulnerable code and use them for their purposes. As a result, vulnerability detection systems that specialize on duplicated code have been proposed.

Different methods can be applied to implement such system. The simplest of them use lexical analyzers and generate a sequence of tokens for the code that contains vulnerability and can be potentially duplicated. Then a sequence of tokens is generated for the program that is tested for vulnerability. In the sequence of tokens of the program being tested, the system tries to find a subsequence that is similar to the sequence of tokens of vulnerability as such. Introducing a sequence similarity parameter makes it possible to express the fact that programmer changed the code during copying and adapted it to a specific situation. By varying this parameter, one can use the vulnerability data and find them in the changed code; however, accuracy of the method decreases since more error of first kind appear [14]. Such method is quite primitive since semantic properties of the program are disregarded.

More progressive methods consider semantic features of vulnerabilities. For example, one of the systems that is based on such approach generates a software graph that takes into account dependences between data and control flow of the program [15]. Such columns are generated both for the code that has vulnerability and for the program that is tested. In this case, detecting vulnerabilities reduces to finding an isomorphic subgraph. At the same time, finding an isomorphic subgraph is an Np-complete problem; therefore, contributors try to delete unnecessary edges when generating the graph or to add more semantic information. Such approach has shortcomings caused by fundamental constraint of subgraph search velocity; moreover, fragments of the code of a specific programming language are analyzed more often, i.e., new analyzer should be developed for each language.

FORMAL METHODS APPLIED TO DETECT WEB ATTACKS

The problem of security of network systems and systems that use web technologies are solved as a whole, beginning with the level of network transport to the level of applied protocol of the system and end user applications.

At the level of network transport, network security problems are mostly solved by means of routers and firewalls. Routers generate network traffic following certain rules defined by settings. Router can only pass through a certain portion of network traffic thus preserving the network behind it from undesirable or suspicious packages. Firewall is a program that only passes through a portion of network traffic, following specific rules. At the same time, issues related to interception of security keys, passwords, web sessions, etc. remain open. For such cases, formal methods are developed to verify the correctness of network protocols.

In particular, the paper [16] describes a formal model of security protocol and shows how it is possible to detect interception of security key by means of formal methods. To this end, the theory is used that includes description of language composed of sets of numerals (terms), constants, symbols of roles and functional symbols, set of formulas that specify equivalence of terms and sets of production rules. Protocols are specified as definitions of behaviors, each being described as a transitional system which, in turn, describes processes of creating, sending, receiving and handling of

messages. To describe transitional systems, message sequence charts (MSC) can be used, and the results of application of the rules can be represented as MSC routes.

The study [17] describes a set of tools that includes specification languages and code generation tools, which make it possible to formally describe and implement the necessary protocol that corresponds to the given specification.

The study [18] represents the method of formal determination of attacks related to variations in network topology, which is based on transformation of topological schemes. This study uses so-called hidden countermeasures, for example, hidden variables against attacks that use a side channel.

Methods related to modeling of network attacks followed by formal analysis of the model are described, in particular, in [19]. Attacks are presented by so-called attack graphs and model checking technique is used for the analysis of properties of these graphs. Moreover, the authors apply probability analysis by means of labeling of attack graphs with the values of probabilities and technique of Markov processes.

There are also finished tools such as Proverif (http://prosecco.gforge. inria.fr/personal/bblanche/proverif/) and Tamarin Prover (https://tamarin-prover.github.io/), intended for formal verification of security protocols, which use symbolic methods of verification as well. The study [20] represents Legacy Crypto tool, which is applied for vulnerability analysis of protocols that use cryptography.

Since the majority of currently available network systems are based on web technologies, security of such systems is also related with vulnerability of applied protocols. The study [21] describes a tool for modeling and analysis of web systems that allows determining system's vulnerability for two types of attack:

- Cross Site Scripting (XSS) Attack, when a malicious scenario is involved to the web page received from a "secure" server and makes it possible to execute transactions on behalf of the client and/or to steal client's personal data;
- Cross Site Request Forgery (CSRF) Attack when the user gets access to the page controlled by the malefactor and thus allows the latter to conduct transactions on behalf of the user.

A difference in these types of attacks is that in case of CSRF the server should be controlled by the malefactor while in case of XSS the server and appropriate services remain secure as a whole.

Other types of attacks in distributed systems that use the web technologies are related to the following:

- Interpretation of addresses where browser sends an incorrectly generated URL request to the server: in this case, incorrectly adjusted server can provide the malefactor with an access to the information stored on it or can execute any system commands;
- insufficient check of the submitted data: a malefactor client can send an information to the server that violates its normal operation; inadmissible symbols, type mismatch, violation of array boundaries, values out of admissible ranges, etc. can be used.
- direct interpretation of SQL requests: parameters of SQL requests are often transmitted directly in web form data or in URL lines; in these cases, simple substitution of parameters can provide the malefactor with illegal access to the data;
- buffer overflow issues: in case of overflow of input buffers, insufficient protection of the code in web applications causes violation of server operation or even make it possible for the malefactor to execute system commands on the server; a part of Ddos attacks also use these vulnerabilities;
- decompilation of Java code: since byte-code can be efficiently decompiled, a malefactor can obtain information from it, which allows unauthorized access to the server, for example, passwords (if this code saves it);
- interception of security keys: SSL keys can be disclosed and replaced by the malefactor who thus can execute transactions on behalf of the client; several potential vulnerabilities of protocols that use cryptosecurity are described in [20];
- impersonalization or interception of sessions: since HTTP protocol does not provide saving of states during interaction with the user, session identifiers which are transmitted between the client and the server as hidden fields are used to this end; capturing such information leads to obtaining information about the user.

In general, in view of the complexity of distributed systems, analysis in the field of their security is usually related to separate aspects. The study [22] overviews and classifies the formal methods used to solve the issues related to security of distributed systems. Some important sections are considered separately: security of Javascript scenarios, security of browsers, security of web applications and analysis of web protocols.

A methodology of generating formal methods of security analysis of systems that are based on web technologies is proposed in [23].

Formal descriptions of attacks related to injection, which include XSS/CSRF, direct interpretation of SQL requests, and attempts of executing system commands on the server are presented in [24].

The paper [25] describes formal methods of determining some types of web attacks and their implementation in the form of browser additions (i.e., on client's side only).

The study [26] presents formal models of attacks related to attempts to hack user's encrypted data in web applications such as password manager, cloud data storage, and systems of electronic voting. Proverif tool is used to model the applications.

Thus, because of the general complexity and multi-aspect nature of network systems, variety and evolution of applied technologies, the issues related to integral security of these systems (in all aspects of operation) remain important. Even in separate aspects, security problems for such systems have been solved only partially.

MACHINE TRAINING

Studies related to utilizing machine training for the analysis and detection of vulnerabilities are being conducted. The vulnerability search strategy is standard for all the algorithms that are based on machine training. First, a training sample is formed, which is then used to create a model in the form of a neural network that detects vulnerabilities in the code. The model is then tested against a control sample of available vulnerability examples. To detect vulnerabilities, samples such as sequences of system calls common to the known detected vulnerabilities, register access templates or packages sent by the network are analyzed. Systems that are based on machine training and artificial intelligence allow finding and generating brand new types of vulnerabilities [10–13].

INCERTION APPROACH

In 2017, studies in using the algebraic approach in vulnerability detection in the binary code began at the V. M. Glushkov Institute of Cybernetics. Semantics of the assembler of instructions languages x86 was formalized on the basis of the theory of incertion modeling developed by A. Letichevsky and D. Gilbert [27]. To this end, the mathematical apparatus of behavior algebra was used, which is a part of the method of incertion modeling.

The main principle of incertion approach [28] is interaction of agents in some environment, when each agent only knows information about the environment but can be the environment for agents of other level of abstraction. Behavior of each agent is described by the equations of behavior algebra. Atomic actions of the agents are objects of behavior algebra and are defined by pre- and post-conditions on attributes of agent environment.

Behavior algebra is a two-sort algebra, which defines relations and operations over behaviors and actions. Each behavior is a composition of actions and behaviors. Prefixing operation a.B defines that action a precedes B. Operation of non-deterministic choice of behaviors $u+v$ defines an alternative to the behavior scenario. The algebra has three terminal constants: successful termination Δ, deadlock state 0, and unknown behavior $\perp$. Approximation relation $\sqsubseteq$, which establishes partial order on the set of behaviors with minimum element $\perp$, is also defined on the set of behaviors. The behavior algebra also defines composition of behaviors: serial (;) and parallel ($||$). In other words, agent's behavior can be presented as an expression over behavior and recursive action. Actions are defined over some environments of attributes in which all the agents interact with each other. Each agent is defined by a set of attributes. Agent changes its state under some conditions generated by the values of attributes. Actions of each agent are defined by the pair $a =< P, S >$, where P is a pre-condition of the action represented as a formula in some basic logical language, S is a post-condition. As a basic logical language, we consider a set of formulas of first-order logic over polynomial arithmetic and some specialized theories, for example, enumeration types, bitwise operations, byte vectors. Generally, semantics of an action means that the agent could change its state if the pre-condition is true, and the state will vary according to post-condition, which is also a formula of first-order logic.

Agent's initial state can be presented by the initial formula. Beginning with the initial formula, it is possible to apply actions that correspond to the expression of behavior algebra. An action is applied if its pre-condition is satisfactory and complies with the current state. Beginning with the formula of initial state S0 and with initial behavior B0, we choose

an action and go to the next behavior. At the first stage, we check the conjunction S0 $\vee$ Pa1 for realizability if B0 = a1. Bl and Pa1 is a pre-condition of a1. The next state of the environment will be obtained by means of a predicate transformer, i.e., function PT over agent's current state, pre-condition, and post-condition: PT (Si, Pai, Qai) = Si + 1.

Applying the predicate transformer function to agent's different states, we can obtain sequences S0, Sl, … of formulas that express variation in agent's states from the initial state. We obtain a behavior scenario as a sequences of actions a1, a2, …. Each agent's state covers certain set of agent's specific values, and process of generating such scenarios is represented as symbolic modeling.

The set of binary code instructions determines interaction of agent's program with its environment: processor, operating system, and peripheral devices.

Formally, this is a set of the following attributes: registers of registers of general and special purposes defined as names of registers; physical memory, which can be considered as memory function; other specific attributes. Semantics of each instruction defines conditions of execution of the instructions, modification of the environment, and the next instruction.

The sequence of execution of a command determines a program control flow, which is determined after transfer of the binary code into the set of expressions of behavior algebra. For example, control flow for the code fragment

```
0000000000400390  <backtrace_and_maps>:
  400390:  ff cf                  dec     edi
  400392:  0f 8e 3a 01 00 00      jle     4004d2
<backtrace_and_maps+0x142>
  400398:  40 84 f6               test    sil,sil
  40039b:  0f 84 31 01 00 00      je      4004d2
<backtrace_and_maps+0x142>
  4003a1:  55                     push    rbp
  4003a2:  53                     push    rbx
```

corresponds to the following specifications of behavior algebra:

```
B400390  =  dec(37,edi).B400392,
B400392  =  jle(38,4195538).B4004d2  +  !jle(38).B400398,
B400398  =  test(39,sil,sil).B40039b,
B40039b  =  je(40,4195538).B4004d2  +  !je(40).B4003a1,
B4003a1  =  push(41,rbp).B4003a2,
B4003a2  =  push(42,rbx).B4003a3,
```

Semantics of instructions is translated to pre- and post-conditions, respectively. For example, instruction

cjne A B z,

is translated into the behavior specification

$$Bx1 = cjne.Bz + !cjne.Bx2,\ Bx2 = \ldots$$

and specifications of actions

$$cjne(n, A, B) = !(A == B) \rightarrow rip = rip + z + 3;\ FLAG\_C = (B > A);$$

$$!cjne(n, A, B) = (A == B) \rightarrow rip = rip + 3;$$

The vulnerability template shows program's behavior that leads to vulnerability state under some conditions (constraints over attributes of the environment). The vulnerability scheme is composed of the expressions of behavior and respective actions. The general form of the vulnerability template is expression of the behavior

Vulnerabilitypattern = Intruderinput; Programbehavior; Vulnerabilitypoint.

There are three main types of behavior in the generalized form. Intruderinput is behavior of the program that represents instructions that are input actions from exterior environment. It may include command line, reading of files, obtaining information from a socket or input from a keypad. Such behavior is mainly implemented as a system call of master code of the operating system. Programbehavior is an arbitrary behavior of the program that connects input point with vulnerability point and can contain other detailed behaviors. Vulnerabilitypoint represents actions that contain vulnerability.

Further, the process of finding vulnerability implies comparison of vulnerability templates to scenarios in the translated binary code in order to find vulnerabilities in case of coincidence. To this end, an efficient algorithm of two-level search can be used.

The first level of search is to find a behavior tree that corresponds to algebraic vulnerability templates. Behavior comparison differs from the traditional one used in anti-virus programs. Traditional models of such programs can contain specific values that are checked whereas the algebraic approach assumes solution of the behavior equations.

The second level of search is executed by symbolic modeling of the given behavior obtained in case of behavior compliance. In symbolic modeling, we apply actions for which we reveal accessibility of the expression Env && Prec, where Env is a symbolic environment of the model of binary code and Prec is a pre-condition of respective action. If it is satisfied, we fulfil operations of post-condition in the template environment and in the environment of the binary code model. If we have attained a vulnerability point in the template, then we have a scenario that conducts from introduction point. Symbolic modeling requires specific solvers to be used, which support operations of bitwise or byte vectors.

Formal methods considerably increase the efficiency of vulnerability search; at the same time, there still have been a number of issues.

One of the problems of formal methods and of the algebraic approach is that the attainability problem in symbolic performance is generally unresolvable. Exponential explosion of the state space and other typical problems of the model checking approach may occur. These problems can be solved with the use of alternative symbolic methods such as generation of invariants, approximation or inverse symbolic modeling. Different search options can reduce the state space; for example, we can provide a code covering criterion. However, this reduction can lead to a missed vulnerability.

One more problem is generalization of algebraic models. It is based on the number of examples and analysis of possible scenarios. However, there is no a universal technique and a guaranty that this model envelops all possible behaviors of vulnerability of the given type.

Other problems of modeling of binary code are as follows: modeling of the environment that contains the OS kernel, libraries, flows, indirect addressing, and other aspects specific to binary code. However, they are not critical but labor-consuming and can be resolved within the framework of this approach.

A substantial increase of activity in cybersecurity is expected for 2019–2020, in particular, implementation of formal methods into the procedure of detecting vulnerabilities and harmful actions.

As a consequence, to maximize the efficiency, model checking methods are integrated, in particular, problems of attainability and methods of automated proof of theorems. The focus will be on the solution of zero-day vulnerability problem by means of fuzzing and machine training methods, which will develop with the use of the newest formal methods with regard for specializations related to data domains.

REFERENCES

1. Cybercrime Magazine. URL: https://cybersecurityventures.com/.
2. Nwokedi Idika and Aditya P. Mathur, A Survey of Malware Detection Techniques, Department of Computer Science, Purdue University, West Lafayette, IN 47907 (2007).
3. Mohan V. Pawar and Anuradha J., “Network security and types of attacks in network,” ICCC-2015. URL: https://ac.els-cdn.com/S1877050915006353/1-s2.0-S1877050915006353-main.pdf?_tid=21866302-2e58-4f1e-88d0-7d3b9825e011&acdnat=1543497106_74f03131d7fc65a2469e9708a18cc54c.
4. Check Point Software Technologies Ltd., Software Blade Architecture. URL: https://ww.checkpoint.com/ downloads/ product-related/brochure/Software-Blades-Architecture.pdf.
5. DARPA, “Cyber Grand Challenge.” URL: https://www.cybergrandchallenge.com/.
6. S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, “Unleashing Mayhem on binary code,” Proc. IEEE Symp. on Security and Privacy (2012), pp. 380–394.
7. A. Nguyen-Tuong, D. Melski, J. W. Davidson, M. Co, W. Hawkins, J. D. Hiser, D. Morris, D. Nguen, and E. Rizzi, “Xandra: An autonomous cyber battle system for the cyber grand challenge,” IEEE Security & Privacy, Vol. 16, No. 2, 42–53 (2008).

8. Mechaphish Github Repository. URL: https://github.com/mechaphish/mecha-docs.
9. American Fuzzy Lop. URL: http://lcamtuf.coredump.cx/afl/.
10. B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, “Deep learning for classification of malware system call sequence,” AI 2016: Advances in Artificial Intelligence, Proc. 29th Australasian Joint Conference, Hobart, TAS, Australia, December 5–8 (2016), pp. 137–149.
11. M. Cova, V. Felmetsger, and G. Banks, “Static detection of vulnerabilities in x86 executables,” 22nd Annual Computer Security Applications Conference (ACSAC’06) (2006). https://doi.org/10.1109/ACSAC.2006.50.
12. M. Mouzarani, B. Sadeghiyan, and M. Zolfaghari, “Detecting injection vulnerabilities in executable codes with concolic execution,” Proc. 8th IEEE Intern. Conf. on Software Engineering and Service Science (ICSESS) (2017). https://doi.org/10.1109/ICSESS.2017.8342862.
13. S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, “Unleashing MAYHEM on binary code,” SP’12 Proc. IEEE Symp. on Security and Privacy (2012). https://doi.org/10.1109/SP.2012.31.
14. Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, and J. Hu, “Vulpecker: An automated vulnerability detection system based on code similarity analysis,” Proc. 32nd Annual Conf. on Computer Security Applications, ACSAC’16 (2016), pp. 201–213.
15. H. Flake, “Structural comparison of executable objects,” Proc. IEEE Conf. on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA) (2004), pp. 161–173.
16. G. Lee, “How to formally model features of network security protocols,” Intern. J. of Security and Its Applications, Vol. 8, No. 1, 423–432 (2014). URL: formal.hknu.ac.kr/Publi/ijsia.pdf.
17. J. Dodds, “Formal methods and the KRACK vulnerability,” Galois Inc. (2017). URL: https://galois.com/blog/2017/10/formal-methods-krack-vulnerability/.
18. J.-S. Coron, “Formal verification of side-channel countermeasures via elementary circuit transformations,” Proc. 16th Intern. Conf., ACNS 2018, Leuven, Belgium, July 2–4 (2018), pp. 65–82. URL: https://eprint.iacr.org/2017/879.pdf/.
19. S. Jha, O. Sheyner, and J. Wing, “Two formal analyses of attack graphs,” Proc. 15th IEEE Computer Security Foundations Workshop (2002). URL: https://ieeexplore.ieee.org/document/1021806.
20. K. Bhargavan et al., “Formal methods for analyzing crypto protocols: Using legacy crypto: From attacks to proofs,” URL: https://cyber.biu.ac.il/wp-content/uploads/2018/02/Biu-bhargavan-part1-slides.pdf.
21. V. Ferman, D. Hutter, and R. Monroy, “A model checker for the verification of browser based protocols,” Comp. y Sist. Vol. 21, No. 1 (2017). URL: http://www.scielo.org.mx/pdf/cys/v21n1/1405-5546-cys-21-01-00101.pdf.
22. M. Bugliesi, S. Calzavara, and R. Focardi, “Formal methods for web security,” Università Ca’ Foscari Venezia. URL: https://www.researchgate.net/publication/308004472_Formal_methods_for_Web_security.
23. L. Tobarra, D. Cazorla, F. Cuartero, and G. Diaz, “Application of formal methods to the analysis of web services security,” URL: https://www.semanticscholar.org/paper/Application-of-formal-methods-to-the-analysis-of-tobarra-cazorla/544d181da33da5439efcf49f31d50116355410d9.
24. D. Ray and J. Ligatti, “Defining injection attacks,” Technical Report #CSE-TR-081114, University of South Florida, Department of Computer Science and Engineering. URL: http://www.cse.usf.edu/~ligatti/papers/broniestr.pdf.
25. S. Calzavara, “Formal methods for web session security,” Università Ca’ Foscari Venezia, Dipartimento di Scienze Ambientali, Informatica e Statistica. URL: http://sysma.imtlucca.it/cina/lib/exe/fetch.php?media=calzavara.pdf.
26. C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, “Keys to the cloud: Formal analysis and concrete attacks on encrypted web storage,” URL: http://antoine.delignat-lavaud.fr/doc/post13.pdf. URL: https://hal.inria.fr/hal-00863375/file/keys-to-the-cloud-post13.pdf/.
27. D. Gilbert and A. Letichevsky, “Model for interaction of agents and environments,” in: Bert D., Choppy C. (Eds.). Recent Trends in Algebraic Development Technique, Wadt 1999. LNCS, Vol. 1827, Springer-Verlag, Berlin–Heidelberg (2000), pp. 311–328.
28. A. Letichevsky, O. Letychevskyi, and V. Peschanenko, “Insertion modeling and its applications,” Computer Sci. J. of Moldova, Vol. 24, Issue 3, 357–370 (2016).