Zero-Day Attack Path Identification using Probabilistic and Graph Approach based Back Propagation Neural Network in Cloud

Swathy Akshaya M.[1], Padmavathi G.[2]

[1]Research Scholar, Department of Computer Science, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, akshayakulandaivel@gmail.com

[2]Professor, Department of Computer Science, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, padmavathy.avinashilingam@gmail.com

Article Info
Page Number: 1091 – 1106
Publication Issue:
Vol. 71 No. 3s2 (2022)

Article History
Article Received: *28 April 2022*
Revised: *15 May 2022*
Accepted: *20 June 2022*
Publication: *21 July 2022*

Abstract:

In the current environment, Networks are generally installed and employed by fundamental security defense procedures like firewalls, Intrusion Detection Systems. It is generally not stress-free for adversaries' to break down the machine. Rather than targets, it usually depends on an attack events chain to flourish threats. A zero-day attack is defined as unknown threats in software for which either patch is not issued or developers are not aware of it. Among many other attacks, this attack is considered as most susceptible one. The number of these exploits discovered remains rising at an increasing rate in the current situation. When these exploits happen in an attack path, the path remains a zero-day attack. The proposed work is developed to identify the Zero-Day Attack path using Probabilistic and Graph Approach based Back Propagation-Neural Network. If specific attack actions avoid system calls, proposed instance graphs capture the complete zero-day attack paths. An approach based on Back Propagation Neural Network outperforms the existing Accuracy, Correctness, and Misclassification parameters. The experimental result shows the effectiveness of the proposed work Back Propagation-Neural Network for zero-day attack path identification, which achieves a better result than the existing work.

Index Terms: Zero-day Attack, Path Identification, Cloud Security, Infected Nodes, System Object Dependency Graph (SODG), Back Propagation Neural Network (BP-NN), Probability Inference.

1. Introduction

In the current scenario, technologies and IT environments are growing very fast. Therefore, threats of exploits are raised more than before. Many companies are ready to work on identifying the known threats using particular security implements like antivirus devices, anti-malware devices, vulnerability assessment tools.

A zero-day exploit may frequently impact resources such as the system or internal users. Placing a Source is infeasible work if not having the forensics capabilities to recognize related elements. Each attack chain is an exploit's order that becomes an attack path. An exploit is facilitated by an unseen vulnerability called a zero-day exploit. Whenever malicious activity is on this path, it converts to a zero-day attack path. In the current situation, number of these exploits discovered remains rising at an increasing rate. As per Symantec's Internet Security Report of 2014, zero-day vulnerabilities were identified in 2013, which is higher than the former year. “Identified twenty-three zero-day vulnerabilities indicates 61% rise over 2012 which is higher than combined two preceding years.

In 2017, these attacks grew from 8 in the preceding year. In 2016, Zero Day Initiative identified numerous threats, such as 50 in Apple products, 76 in Microsoft products, and 135 in Adobe products. At the Equifax breach, attackers expanded access to data from the primary consumer credit reporting agency in September 2017. In the Equifax database, the Personal information of more than 143 million people is stolen. The Famous WannaCry ransomware attack threatened most of the world in September 2017 because of a zero-day exploit.

Tactlessly, specialists forecast the occurrence of these threats, which is going to degrade with technology. In 2015, there was about one per week. Cyber security Ventures foresaw that there would be one new exploit that will occur every day in 2021. Identified attacks emerged from eight in 2011 to eighty-four in 2016 [1]. When it continues like this, a new zero-day attack will be identified every day of 2022. Each of these activities signifies a vulnerability that may consequence in a tremendously dangerous zero-day attack that has the capability of striking complete industries.

The objective of this research work is to identify the zero-day attack path. In this work, a probabilistic and graph approach based Back Propagation Neural Network is proposed for identifying the zero-day attack path. If specific attack actions evade system calls, proposed instance graphs capture the complete zero-day attack paths.

This work is organized as follows:

1. Section 1 briefly discussed the introduction, motivation, objective of the paper, and organization of the paper.
2. Section 2 describes previous approaches applied for zero-day detection.
3. Section 3 illustrates the proposed approach in detail.
4. Section 4 presents the experimental results by comparing the proposed method with the existing approach.
5. Section 5 accounts for the conclusion and future scope.

2. Recent Statistics of Zero-day Attack

Some of the recent impacts and statistics of zero-day attacks are represented in Table.1

Table 1. Recent Impact and Statistics of Zero-Day Attack

Year	Zero-Day Attacks	Infection
2021	30 Zero-Day Vulnerabilities Discovered	Multiple Vulnerabilities appeared in Apple macOS, Google Chrome, Microsoft Windows, and other industries.
2020	38 Zero-Day Vulnerabilities Discovered	Security Restrictions Bypass and Authentication Bypass.
2019	Discovered 28 Zero-Day Vulnerabilities	Remote code execution in organizations.

2018	Six Undisclosed Zero-Day Vulnerabilities	3 Manage Engine Products. The application includes log 360 and even log application manager.
2017	WannaCry Ransomware Attack	Hitting several orientations worldwide, including UK national health servers.
2017	Zero-Day Attack	The application attack surface is raised by 111 billion new lines of software code per year.
2016	Discover Zero-Day Exploits	Hitting entire 84 industries.
2016	Zero-Day Attack	135 vulnerabilities in Adobe and 76 vulnerabilities in Microsoft products.
2014	Zero-Day Attack	23 Zero-day vulnerabilities identified indicate a 61% growth over 2012.
2011	Discover Zero-Day Exploits	Hitting entire 8 industries.

Due to signatures are not generated, these exploits may not be identified by anti-malware or IDS/IPS devices [2]. Though zero-day attacks are formidable to discover and identify, numerous strategies have emerged. According to Ratinder Kaur et al., every attack follows the basic detection strategies given below in Figure. 1.

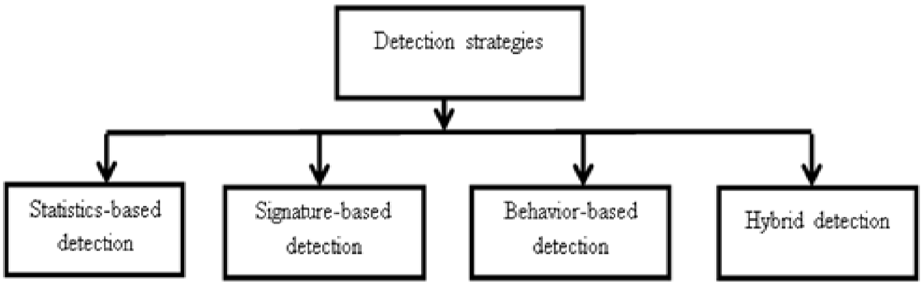


Figure.1 Detection Strategies for Zero-Day Attack

1. **Statistical Detection-** This method uses machine learning approaches to collect data from formerly identified exploits and produce a model for safe system behavior. These techniques provide limited effectiveness and false positives/negatives. Security administrators have trust in behavior-based detection approaches without particular detection capabilities.
2. **Signature Detection-** The threat scanning process practices previously used malware databases and their behavior as a reference in this method. After analyzing using the machine learning approach and generating signatures for previously available malware, necessary to utilize the signatures to identify the formerly unidentified attacks. This method comprises a search of bytes sequence within a malicious executable and files previously affected by this particular malware. This method gives better results only if threats are identified earlier. Only after an instance of this malware has affected the networks and systems can an expert specify

a signature for new malware executables. Due to malicious software unnoticed earlier, this method cannot manage the zero-day attacks [3].

3. **Behavior Detection-** This method identifies malware depending on its interactions with the target system. If it is the consequence of a malicious attack, it examines its interactions with present software to identify.
4. **Hybrid Detection-** This method integrates the methods mentioned earlier to use the benefit of its strengths.

3. Related Works

Avasarala et al. (2017) proposed a class matching approach with a procedure for recognizing the number of suspect objects that encompasses data regarding the network transactions or computer operations likely related to a security threat. Suspicious objects are transferred to an inspection service operating that implement on 1 or a few common-purpose computers. Digital data is transmitted to an analytical service operating that implement on 1 or a few common purpose computers. 1 or few scores are transmitted to a correlation facility that groups a scores plurality, supplementary data regarding each suspicious object together form a cumulative data that represents 1 or few cumulative features of suspicious objects in plurality form, producing an infection verification pack that contain routines, during execute on an end-point machine in the computer network setting, thus reducing the mistrusted security risk [4].

Nahid Hossain et al. (2017) presented tag-based methods to identify the attack and reconstruction that contain identification of source and analysis of influence. The new techniques are proposed to disclose the giant depiction of attacks through compact construction and visual graphs of attack phases. This model contributed to a red team assessment run by DARPA and detected and reconstructed the information of the red team's attacks on hosts, which run on Linux, FreeBSD and Windows successfully [7].

Shiqing Ma et al. (2017) proposed semantics aware program annotation and instrumentation method. This method splits work performance depending on application explicit high-level task structures, thus preventing training, producing execution partitions with rich semantic info, and offering numerous perceptions of the attack. A prototype is developed and integrated by three dissimilar provenance systems: ProTracer, Linux Audit system, and LPM-HiFi system. This method produces cleaner attack graphs having rich high-level semantics and gives low time overheads and space [8]. BEEP ProTrace and MPI seek to achieve higher precision than Backtracker, but they have inadequate scalability because they are not always automated instrumentation. SLEUTH provides more effective event storage and analysis.

Xiaoyan Sun et al. (2016) identified a zero-day attack path using the probabilistic method and used ZePro and Patrol's prototype system. Analyzing the system calls built a Bayesian network depending on the instance graph to disclose the zero-day attack paths. This method calculates the possibilities to get the object instances infected. The high probability instances are connected using dependency relations that create zero-day attacks [10]. This system revealed parts of the attack paths only.

Mishra and Gupta (2014) introduced a combined solution that employs the CSS and URI matching concepts to guard against a zero-day phishing attack. These attacks are viral and severe hazards on the Internet that are utilized to cheat users and snip their data through spoofed emails, fake websites, or both. A hybrid solution is proposed to defend against this attack. In this work, the matching concept is used for every URI with trusted domains using the Link Guard algorithm, and the concept of CSS matching is used from the Bait Alarm scheme. This approach is practical and provides security to various types of website phishing attacks, and produces a false-positive rate in less amount [5].

Wang et al. (2014) proposed certain demonstrative mechanisms on determining the zero-day attack to estimate the strength of networks modeled network diversity and then introduced two complementary diversity metrics. k-zero-day safety is a diversity metric to discourse this problem. This metric computes how many vulnerabilities are necessary for compromising network resources; a large amount infers high security due to the possibility of having more unidentified vulnerabilities at the identical time that can below. The complementary diversity metrics are proposed based on the least average attacking efforts [6].

Ratinder Kaur et al. (2014) proposed a Hybrid Technique for Detecting Zero-Day Polymorphic Worms by using Signature and Anomaly Detection. It has some difficulties detecting zero-day through signatures. Because signatures are hard to detect, a zero-day attack has new signatures for each new attack, and thus it becomes complex.

4. Literature Review

Some of the significant works of zero-day attacks provide a detailed view of literature are reviewed and presented in the following Table. II.

Table II. Review of Literature for Zero-Day Attack Identification

Author	Year	Title	Techniques	Observation
Avasarala et al.	2017	System and method for automated machine learning, zero-day malware detection [CNS].	Class matching approach.	True positive and False positive detection rate low. The accuracy of detection needs to be improved.
Xiaoyan Sun, et al.	2016	Towards Probabilistic Identification of Zero-day Attack Paths [IEEE].	Bayesian Networks.	If an attack evades System calls or attack span time exceeds the given period, the current system may not detect a Zero-day attack.
Chanchala Joshi, et.al.	2016	ZDAR System: Defending Against the Unknown [IJCSMC].	Supervised and Unsupervised Classification.	Signature generation of unknown activities is complex, the false alarming rate of anomalous behavior.
Ravinder Kaur et.al.	2014	Hybrid Technique for Detecting Zero-Day Polymorphic Worms [IEEE].	Signature and Anomaly Detection.	Signatures are hard to detect, and the zero-day attack has new signatures for each new attack.
Wang et al.	2014	k-Zero-Day Safety: A Network Security Metric for Measuring the Risk of Unknown Vulnerabilities [IEEE].	k-zero-day safety as shortest paths in a DAG.	Complexity of computing.

Observations due to Literature Study

Among this literature, the significant findings of zero-day attacks are listed below.

1. According to Xiaoyan Sun et al., if attack span time exceeds the given period, the system may not detect a Zero-day attack.
2. According to Ravinder Kaur et al., signatures are hard to detect, and a zero-day attack has new signatures for each new attack.

The limitations can be overcome by being capable of detecting the zero-day attack, and by finding the zero-day attack path identification, the threat can be handled and reduced to a certain extent.

5. Proposed Methodology

System call auditing is done in each host, gathered its traces, and transferred to the central investigation machine for offline instance graph grounded BN construction and attack path detection. The proposed flow diagram is given in figure 2.

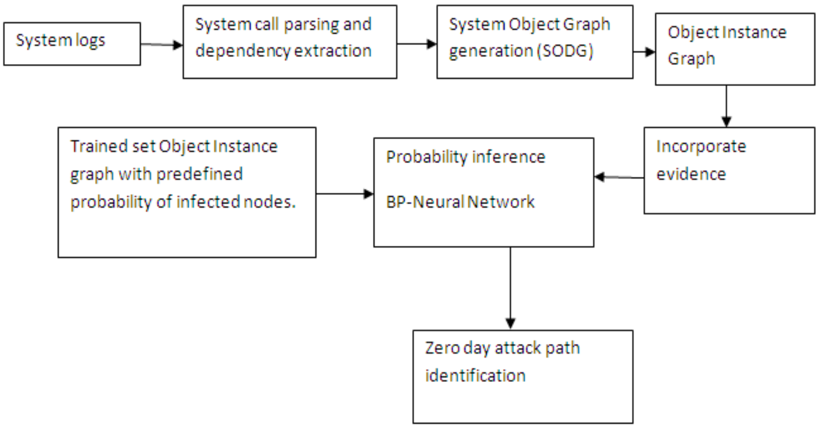

System logs
System call parsing and dependency extraction
System Object Graph generation (SODG)
Object Instance Graph
Incorporate evidence
Probability inference
BP-Neural Network
Trained set Object Instance graph with predefined probability of infected nodes.
Zero day attack path identification

Figure.2 Flow Diagram for Proposed Work

5.1 Proposed Methodology

a) Procedure

1. First, build a network-wide supergraph from system calls.
2. Identify the zero-day attack path (subgraph) hidden in the supergraph.

b) Steps

1. **Step 1: Logging-** The execution of the program and its communication are logged. The process involved and communication between two processes or files can be seen in the log file.
2. **Step 2: Parsing system call-** Each host file can be parsed into system objects, such as File, Process, and Socket.
3. **Step 3: System Object Dependency Graph**: It is built as a super graph to identify the intrusion propagation by investigating the system calls.

a. Dep $\in \{(src \leftarrow sink), (src \rightarrow sink), (src \leftrightarrow sink)\}$, src and sink indicates OS objects, then $V_x = V_x U\{src, sink\}$ and $E_x = E_x U\{dep\}$. dep gets start and end timestamps from syscall.

4. **Step 4: Parsing system object instances-** If system call trace $T\ [t_{\text{begin},}\ t_{\text{end}]}$ in a time window is indicated as $\sum_T$ and system objects set included in $\sum_T$ is indicated as O_T, object instance graph GT (V. E)

a. When syscall $\in \sum_T$ has parsed to 2 system object instances such as $\text{src}_x, \text{sink}_y$, x, y ≥ 1, and dependency relation dep_z: $\text{src}_x \rightarrow \text{sink}_y$ in which src_x is indicated as x^{th} instance of system object src εO_T and sink_y indicated as y^{th} instance of system object sink $\in O_T$, $V = V \cup \{\text{src}_x, \text{sink}_y\}$, $E = E \cup \{\text{dep}_z\}$. The timestamps for $\text{syscall}, \text{dep}_{c,}$ and sink_j are t_syscall, t_dep_z, t_src_x, and t_sink_y. The t_dep_z gets t_syscall from syscall.

5. **Step 5: Graph Pruning:** The complexity and speed of the processing are reduced by instance graph pruning. The duplicate entries or dependency connections between system objects are removed.

6. **Step 6: Zero-Day Attack Identification-** Infected nodes are predefined in the environment. These nodes combine probability values for infected nodes and aid in recognizing these attack paths.

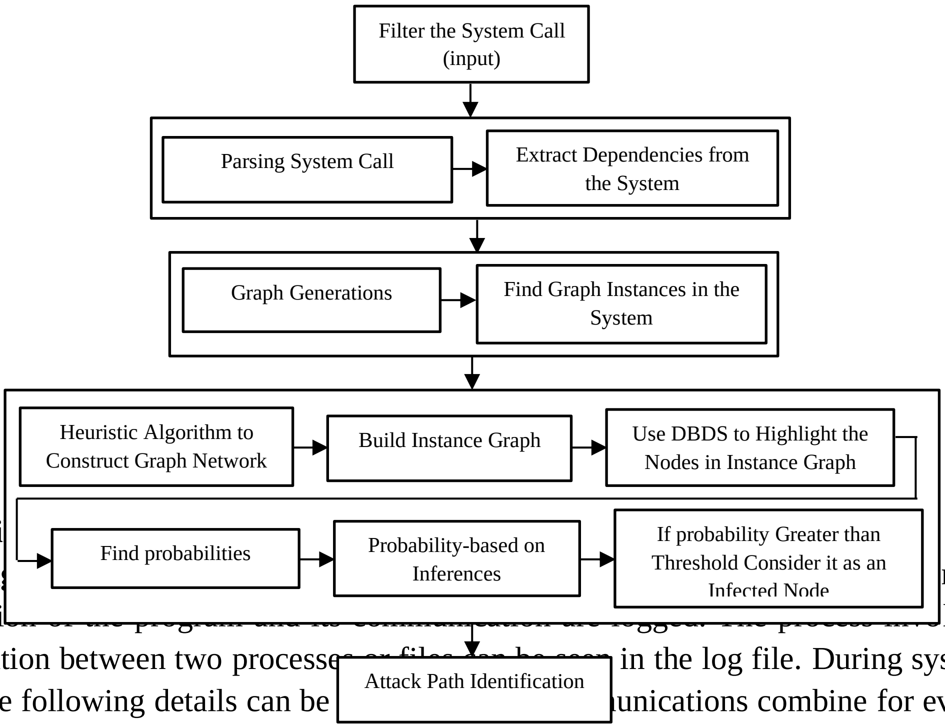
a. Samples Set D = $\{(x_{1,}t_1), (x_{2,t_2}) \ldots (x_{i,t_i}) \ldots (x_{n,t_n})\}$ with x_i denotes i_{th} input vector are trained and t_i denotes equivalent output one.

b. Using BP neural network, predicting result can be written as:

$$\hat{t_i} = \sum_{j-1}^{M} \omega_2(j) \tanh \left(\sum_{j-1}^{r} \omega_1 (j,k) x_{jk} + \theta_1(j)\right) + \theta_2$$

7. **Step 7: Output-** zero-day attack path is identified.

The steps of the proposed methodology have been depicted in Figure.3.


Filter the System Call (input)
Parsing System Call
Extract Dependencies from the System
Graph Generations
Find Graph Instances in the System
Heuristic Algorithm to Construct Graph Network
Build Instance Graph
Use DBDS to Highlight the Nodes in Instance Graph
Find probabilities
Probability-based on Inferences
If probability Greater than Threshold Consider it as an Infected Node
Attack Path Identification

5.2 A Detai

a) Log rocesses. The execution of the program and its communication are logged. The process involved and communication between two processes or files can be seen in the log file. During system call auditing, the following details can be unications combine for every host instance graphs b) node numbers and file absolute pathnames are OS-aware knowledge. c) Timestamp for each system call. The bandwidth and CPU cost, which suffers due to redundant or possibly innocent objects, are solved by System call filtering.

b) **Parsing into System Object:** The OS object instances and dependency relations are parsed through a system call. These parameters are also involved in the parsing. It distinctively identifies and names the nodes and supports to conclude the edge direction between them. Each host file can be parsed into system objects.

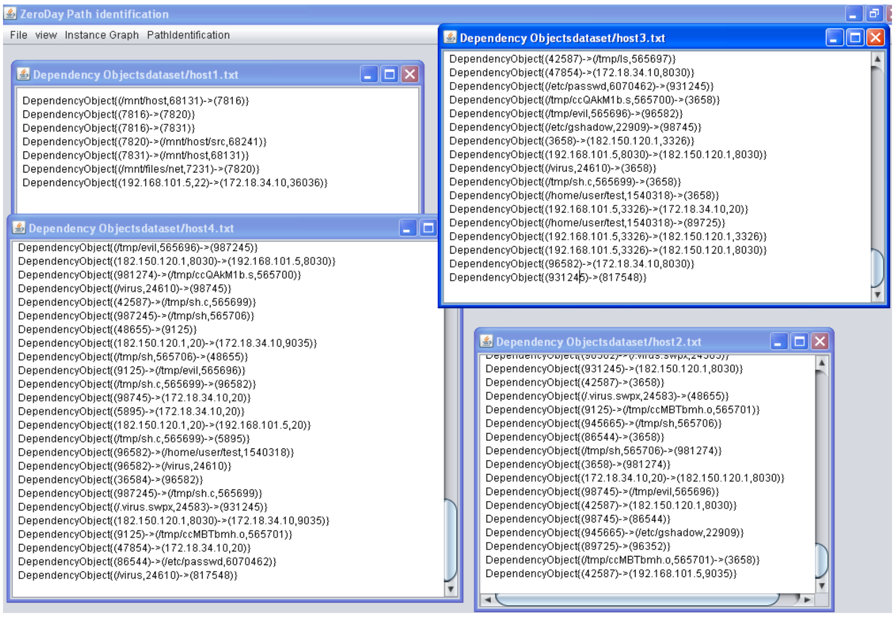
c) **Instance Graph:** These parsed things are turned to be nodes and directed edges from the system call parsing. The procedure of producing the object instance graph from system object dependencies is provided below.

d) **Construct Dependency Objects:** It is built as a super graph to identify the intrusion propagation by investigating the system calls.

System Object Dependency Graph

1. When the system call trace for x^{th} host is represented as $\sum x$, this graph becomes directed graph G (V_x, E_x) for this host in which $_{vx}$ indicates nodes set, initialized to {∅}; and E_x indicates directed edges set, initialized to {∅}. When the system calls syscalls and dep, the dependency relation parsed from syscall. As per dependency rules, Dep ∈ {(src ← sink) , (src → sink), (src ↔ sink)}, src and sink indicate OS objects, then V_x = V_xU{src, sink} and E_x= E_xU{dep}. dep gets start and end timestamps from syscall.

2. When (a→ b) ∈ $E_x$pand (bc)∈ E_x, then c transitively based on a. System calls have parsed into system objects and dependencies. The dependency rules are given to parse system calls. Depending on these rules, the system calls are parsed into 3 portions: an src object, a sink object, and dep relation. The objects and dependency relations are united to form a directed graph.


ZeroDay Path identification
File
view
Instance Graph
PathIdentification
Dependency Objectsdataset/host1.txt
Dependency Objectsdataset/host4.txt
Dependency Objectsdataset/host3.txt
Dependency Objectsdataset/host2.txt

```
DependencyObject((/mnt/host,68131)->(7816))
DependencyObject((7816)->(7820))
DependencyObject((7816)->(7831))
DependencyObject((7820)->(/mnt/host/src,68241))
DependencyObject((7831)->(/mnt/host,68131))
DependencyObject((/mnt/files/net,7231)->(7820))
DependencyObject((192.168.101.5,22)->(172.18.34.10,36036))
```

```
DependencyObject((/tmp/evil,565696)->(987245))
DependencyObject((182.150.120.1,8030)->(192.168.101.5,8030))
DependencyObject((981274)->(/tmp/ccQAkM1b.s,565700))
DependencyObject((/virus,24610)->(98745))
DependencyObject((42587)->(/tmp/sh.c,565699))
DependencyObject((987245)->(/tmp/sh,565706))
DependencyObject((48655)->(9125))
DependencyObject((182.150.120.1,20)->(172.18.34.10,9035))
DependencyObject((/tmp/sh,565706)->(48655))
DependencyObject((9125)->(/tmp/evil,565696))
DependencyObject((/tmp/sh.c,565699)->(96582))
DependencyObject((98745)->(172.18.34.10,20))
DependencyObject((5895)->(172.18.34.10,20))
DependencyObject((182.150.120.1,20)->(192.168.101.5,20))
DependencyObject((/tmp/sh.c,565699)->(5895))
DependencyObject((96582)->(/home/user/test,1540318))
DependencyObject((96582)->(/virus,24610))
DependencyObject((36584)->(96582))
DependencyObject((987245)->(/tmp/sh.c,565699))
DependencyObject((/.virus.swpx,24583)->(931245))
DependencyObject((182.150.120.1,8030)->(172.18.34.10,9035))
DependencyObject((9125)->(/tmp/ccMBTbmh.o,565701))
DependencyObject((47854)->(172.18.34.10,20))
DependencyObject((86544)->(/etc/passwd,6070462))
DependencyObject((/virus,24610)->(817548))
```

```
DependencyObject((42587)->(/tmp/ls,565697))
DependencyObject((47854)->(172.18.34.10,8030))
DependencyObject((/etc/passwd,6070462)->(931245))
DependencyObject((/tmp/ccQAkM1b.s,565700)->(3658))
DependencyObject((/tmp/evil,565696)->(96582))
DependencyObject((/etc/gshadow,22909)->(98745))
DependencyObject((3658)->(182.150.120.1,3326))
DependencyObject((192.168.101.5,8030)->(182.150.120.1,8030))
DependencyObject((/virus,24610)->(3658))
DependencyObject((/tmp/sh.c,565699)->(3658))
DependencyObject((/home/user/test,1540318)->(3658))
DependencyObject((192.168.101.5,3326)->(172.18.34.10,20))
DependencyObject((/home/user/test,1540318)->(89725))
DependencyObject((192.168.101.5,3326)->(182.150.120.1,3326))
DependencyObject((192.168.101.5,3326)->(182.150.120.1,8030))
DependencyObject((96582)->(172.18.34.10,8030))
DependencyObject((931245)->(817548))
```

```
DependencyObject((96582)->(/.virus.swpx,24583))
DependencyObject((931245)->(182.150.120.1,8030))
DependencyObject((42587)->(3658))
DependencyObject((/.virus.swpx,24583)->(48655))
DependencyObject((9125)->(/tmp/ccMBTbmh.o,565701))
DependencyObject((945665)->(/tmp/sh,565706))
DependencyObject((86544)->(3658))
DependencyObject((/tmp/sh,565706)->(981274))
DependencyObject((3658)->(981274))
DependencyObject((172.18.34.10,20)->(182.150.120.1,8030))
DependencyObject((98745)->(/tmp/evil,565696))
DependencyObject((42587)->(182.150.120.1,8030))
DependencyObject((98745)->(86544))
DependencyObject((945665)->(/etc/gshadow,22909))
DependencyObject((89725)->(96352))
DependencyObject((/tmp/ccMBTbmh.o,565701)->(3658))
DependencyObject((42587)->(192.168.101.5,9035))
```

Figure.4 Dependency Objects

e) **Construct Object Instance Graph:** Object instance graph stands kind of dependency graph. Node is an object instance with a particular timestamp. Moreover, different instances have different versions of an identical object at different time points, with a different infection state. This graph has an equivalent or larger size.

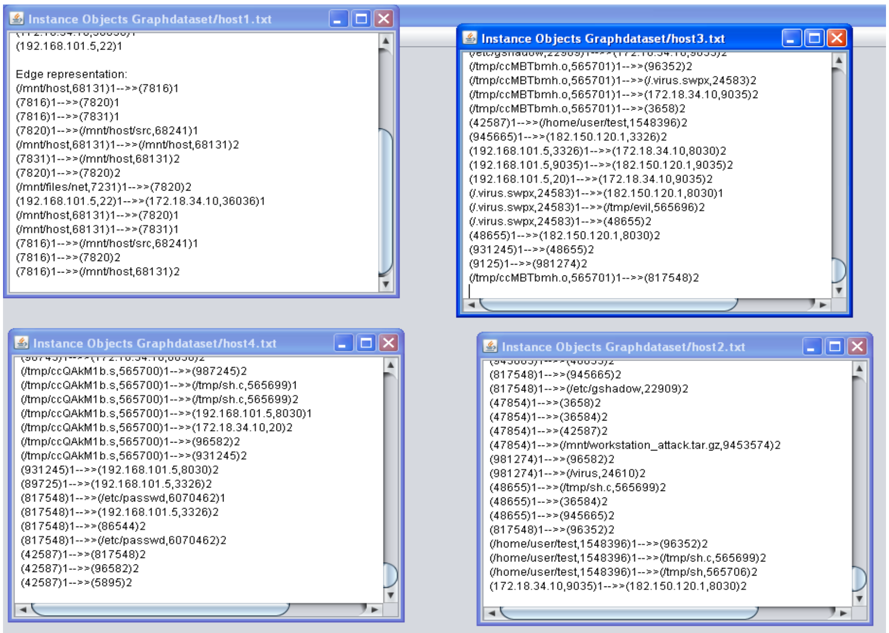
If system call trace T [$t_{begin,}$ $t_{end]}$ in a time window is indicated as $\sum_T$ and system objects set included in $\sum_T$ is indicated asO_T, object instance graph $G_T(V.E)$. in which V denotes node-set, initialized to {∅} and E denotes directed edges set, initialized to {∅}.

1. When syscall ∈ $\sum_T$ has parsed into 2 system object instances such as $src_x, sink_y$, x, y ≥ 1, and dependency relation dep_z: $src_x \rightarrow sink_y$ in which src_x is indicated as x^{th} instance of system object src εO_T and $sink_y$ indicated as y^{th} instance of system object sink∈ O_T, V = V ∪{$src_x, sink_y$}, E = E ∪ {dep_z}. The timestamps for syscall, $dep_{c,}$ and $sink_j$ are t_syscall, t_dep_z, t_src_x, and t_$sink_y$. The t_dep_z gets t_syscall from syscall. Before appending src_x and $sink_y$ into V, the x and y indexes are analyzed.
2. For ∀src_x , $sink_y$ ∈ V, i, j ≥ 1, when x_{max} and y_{max} are maximum indexes of instances for src and sink;
3. When ∃src_z V, z ≥1,then x = x_{max}, and t_src_x remains identical; else, x = 1, and t_src_x is changed to t_syscall Ifi ∃ $sink_i$ V, i≥1, then y = y_{max}; else, y = 1. t _$sink_y$ has changed to t_syscall, If j is equal to 2, then E is equal to E U{dep_s: $sink_y - 1 \rightarrow sink_y$}.
4. When a→b ∈ E and bc ∈ E, then c transitively based on a new instance

First, the instance graph can reflect correct infection causality relations by implying time information stamped onto specific instances. Second, the instance graph can break the cycles contained in SODG.

```
Algorithm 1- Object Instance Graph Generation
Input: system object dependencies set D,
Output: G (V, E) instance graph
For each dep: src→sink∈D Do
  Look up the most recent instance $src_k$ of src, $sink_z$ of sink in V
  If $sink_z$ ∉V then
     Create new instances $sink_1$
     V←V U {$sink_1$ }
  If $src_k$ ∉ then
     Create new instances $src_1$
     V←V U {$sink_1$}:
      E←EU {$src_1$→$sink_1$}
  Else
      E←E U {$src_k$ → $sink_1$}
  End if
End if
If $sink_z$ ∈ V then
      Create new instances$sink_{z+1}$
      V ← V U {$sink_{z+1}$}
      E ← E U {$sink_z$ → $sink_{z+1}$ }
 If  $src_k$ ∉ V then
       Create new instances $src_1$
```

```
            $V \leftarrow V \cup \{src_{1}\}$
            $E \leftarrow E \cup \{src_1 \rightarrow sink_{z+1}\}$
      Else
            $E \leftarrow E \cup \{src_k \rightarrow sink_{z+1}\}$
      End if
    End if
End for
```


Instance Objects Graphdataset/host1.txt
Instance Objects Graphdataset/host3.txt
Instance Objects Graphdataset/host4.txt
Instance Objects Graphdataset/host2.txt

```
(192.168.101.5,22)1

Edge representation:
(/mnt/host,68131)1-->>(7816)1
(7816)1-->>(7820)1
(7816)1-->>(7831)1
(7820)1-->>(/mnt/host/src,68241)1
(/mnt/host,68131)1-->>(/mnt/host,68131)2
(7831)1-->>(/mnt/host,68131)2
(7820)1-->>(7820)2
(/mnt/files/net,7231)1-->>(7820)2
(192.168.101.5,22)1-->>(172.18.34.10,36036)1
(/mnt/host,68131)1-->>(7820)1
(/mnt/host,68131)1-->>(7831)1
(7816)1-->>(/mnt/host/src,68241)1
(7816)1-->>(7820)2
(7816)1-->>(/mnt/host,68131)2
```

```
(/tmp/ccMBTbmh.o,565701)1-->>(96352)2
(/tmp/ccMBTbmh.o,565701)1-->>(/.virus.swpx,24583)2
(/tmp/ccMBTbmh.o,565701)1-->>(172.18.34.10,9035)2
(/tmp/ccMBTbmh.o,565701)1-->>(3658)2
(42587)1-->>(/home/user/test,1548396)2
(945665)1-->>(182.150.120.1,3326)2
(192.168.101.5,3326)1-->>(172.18.34.10,8030)2
(192.168.101.5,9035)1-->>(182.150.120.1,9035)2
(192.168.101.5,20)1-->>(172.18.34.10,9035)2
(/.virus.swpx,24583)1-->>(182.150.120.1,8030)1
(/.virus.swpx,24583)1-->>(/tmp/evil,565696)2
(/.virus.swpx,24583)1-->>(48655)2
(48655)1-->>(182.150.120.1,8030)2
(931245)1-->>(48655)2
(9125)1-->>(981274)2
(/tmp/ccMBTbmh.o,565701)1-->>(817548)2
```

```
(/tmp/ccQAkM1b.s,565700)1-->>(987245)2
(/tmp/ccQAkM1b.s,565700)1-->>(/tmp/sh.c,565699)1
(/tmp/ccQAkM1b.s,565700)1-->>(/tmp/sh.c,565699)2
(/tmp/ccQAkM1b.s,565700)1-->>(192.168.101.5,8030)1
(/tmp/ccQAkM1b.s,565700)1-->>(172.18.34.10,20)2
(/tmp/ccQAkM1b.s,565700)1-->>(96582)2
(/tmp/ccQAkM1b.s,565700)1-->>(931245)2
(931245)1-->>(192.168.101.5,8030)2
(89725)1-->>(192.168.101.5,3326)2
(817548)1-->>(/etc/passwd,6070462)1
(817548)1-->>(192.168.101.5,3326)2
(817548)1-->>(86544)2
(817548)1-->>(/etc/passwd,6070462)2
(42587)1-->>(817548)2
(42587)1-->>(96582)2
(42587)1-->>(5895)2
```

```
(817548)1-->>(945665)2
(817548)1-->>(/etc/gshadow,22909)2
(47854)1-->>(3658)2
(47854)1-->>(36584)2
(47854)1-->>(42587)2
(47854)1-->>(/mnt/workstation_attack.tar.gz,9453574)2
(981274)1-->>(96582)2
(981274)1-->>(/virus,24610)2
(48655)1-->>(/tmp/sh.c,565699)2
(48655)1-->>(36584)2
(48655)1-->>(945665)2
(817548)1-->>(96352)2
(/home/user/test,1548396)1-->>(96352)2
(/home/user/test,1548396)1-->>(/tmp/sh.c,565699)2
(/home/user/test,1548396)1-->>(/tmp/sh,565706)2
(172.18.34.10,9035)1-->>(182.150.120.1,8030)2
```

Figure.5 Object Instance Graph

f) **Instance Graph Pruning:** The complexity and speed of the processing are reduced by instance graph pruning. The duplicate entries or dependency connection between system objects is removed. Although different system calls can trigger it, it is not uncommon that a similar dependency can happen various times between any pair of system objects.

g) Incorporate Evidence: In the environment, infected nodes are predefined. These nodes combine probability values for infected nodes and aid in recognizing these attack paths. This module combines evidence into instance-graph grounded BP-Neural Network by labeling the contamination state of the involved object instance as infected or appending the Local Observation Model node as a child node to an instance of the object for modeling the uncertainty towards observations.

h) Back Propagation-Neural Network: Using a training set, this method calculates probability from Neural Network. Through these training sets, the affected rate of the test node is identified and shown in Figure.6.

It contains an input layer, single or numerous hidden, and one output layer.

Set of Sample say D={$(x_{1,}t_1), (x_{2,t_2}) \dots (x_{i,t_i}) \dots (x_{n,t_n})$}with x_i denotes i_{th} input vector are trained and t $_i$ denotes equivalent output one.

Using BP neural network, predicting result can be written as:

$$\hat{t_i} = \sum_{j-1}^{M} \omega_2(j)\tanh\left(\sum_{j-1}^{r} \omega_1(j,k)x_{jk} + \theta_1(j)\right) + \theta_2$$

Where the number of input neurons and hidden neurons are indicated by P and M, biases OJ and 82, weights joining hidden and output layer (J), (j), the weights joining input and hidden layer (J), (j, k) and kth elements of jth input are denoted by $x_j = [x_{jk,}k = 1,2,\dots P]$.

It has a description of layer quantity and neuron quantity per layer, and every neuron employs activation function type and existing association CV among the neural units. The error term is minimized by

$E_p = \frac{1}{2}\sum_{req}^{n}(t_i\text{-}\hat{t_i})^2 = \frac{1}{2}\sum_{err}^{n} e_i^2$

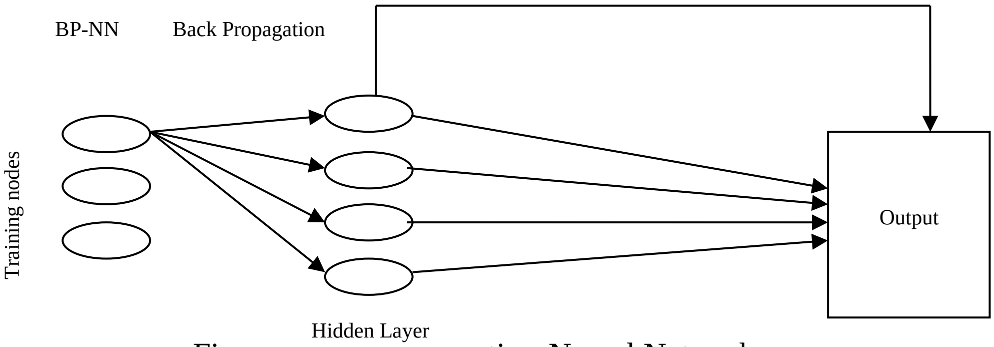
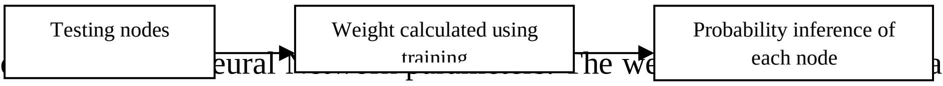


Figure.6 Back Propagation-Neural Network


Testing nodes
Weight calculated using training
Probability inference of each node

It is used t eural network parameters. The we as network error function, which can be followed as

F (W) =$\beta\overline{E_p} + \alpha\overline{E_\omega}$

With $e_\infty = \frac{1}{2}\sum_{j=1}^{m} \omega_{1,2}(f)^{2'}$$\alpha$ and β are called hyperparameters, weight between hidden and output layer and weight between input and hidden layer are denoted by CV. Weight CV for posterior probability function in Bayesian rule is

P $(\omega|D,\alpha,\beta,H) = \frac{p(D|\omega,\beta,H)p(|\omega|\alpha,H)}{p(D|\alpha,\beta,H)}$

Then,

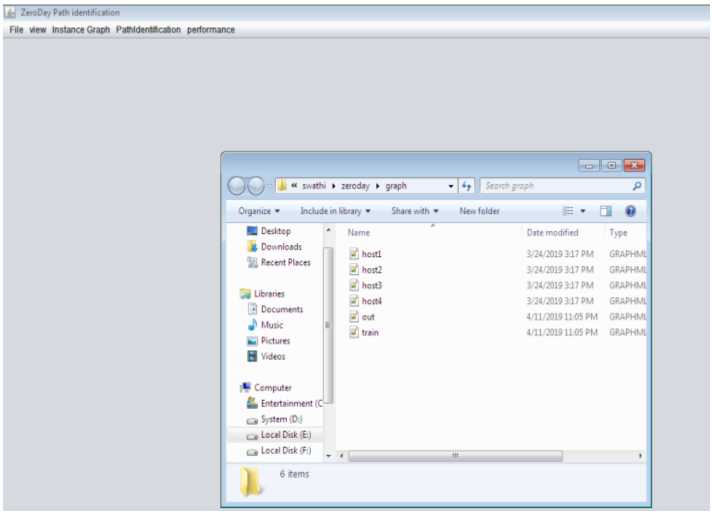
$$P\,(D|\alpha,\beta,H) = \frac{p(D|\omega,\beta,H)p(\omega|\alpha,H)}{p(\omega|D,\alpha,\beta,H)}$$

i) Zero-Day Path Identification: Zero-day attack paths are identified by inferred probabilities and the nodes having high probabilities and edges inter-linking the instance graph. These paths are shown in Figure.7. It has a high probability on its own or both descendant and ancestor having high probabilities to motivation on these nodes in instance graph. These emphasized nodes are involved in the infection propagation, and therefore it must be kept and shown in Figure.8 (a) & (b). A high probability called a tuning parameter (threshold) is applied to control the bottom probability. Output stored in graphml format. It can be viewed in yed Editor.

Algorithm 2- Identification of Zero-day Attack Path

```
Input: G (V, E) instance graph and v ∈ V vertex
Output: $G_z(V_z, E_z)$ zero-day attack path
 Function DFS (G, v, direct)
    Initially V is marked as visited
    If (direct = ancestor)
        $next_v$= parent of v that $next_v$ → v∈E
        Flag = high probability ancestor
    Else if (direct = descendant)
        $next_v$=child of v that v → $next_v$∈E
        Flag = high probability descendant
    End if
    Do
        If $next_v$ has not marked as visited
            If (prob [$next_v$] ≥ threshold or $next_v$) = flag find high probability =True
            Else
                DFS (G, $next_v$, direct)
            End if
        End if
        If find_high_probability =True, then v is marked as flag
        End if
    While (all $next_v$ of v)
End function
Do
    DFS (G, v, ancestor)
    DFS (G, v, descendant)
While(all v ∈ E)
Do
    If prob [v] ≥threshold or (v is labeled as has high probability ancestor and v is labeled as has high probability descendant) $V_z \leftarrow V_{z \cup v}$
        $V_z \leftarrow V_{z \cup v} V_z \leftarrow V_{z \cup v}$
    End if
```

```
While (all $v \in V$)
Do
        If $v \in V_z$ and $\omega \in V_z$ then
            $E_z \leftarrow E_{z \cup e}$
        End if
While ($e: v \rightarrow \omega \in E$)
```


ZeroDay Path identification
File view Instance Graph PathIdentification performance
« swathi ▸ zeroday ▸ graph
Search graph
Organize ▾ Include in library ▾ Share with ▾ New folder
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Entertainment (C
System (D:)
Local Disk (E:)
Local Disk (F:)
6 items

Name	Date modified	Type
host1	3/24/2019 3:17 PM	GRAPHML
host2	3/24/2019 3:17 PM	GRAPHML
host3	3/24/2019 3:17 PM	GRAPHML
host4	3/24/2019 3:17 PM	GRAPHML
out	4/11/2019 11:05 PM	GRAPHML
train	4/11/2019 11:05 PM	GRAPHML

Figure.7 Zero-Day Attack Detection

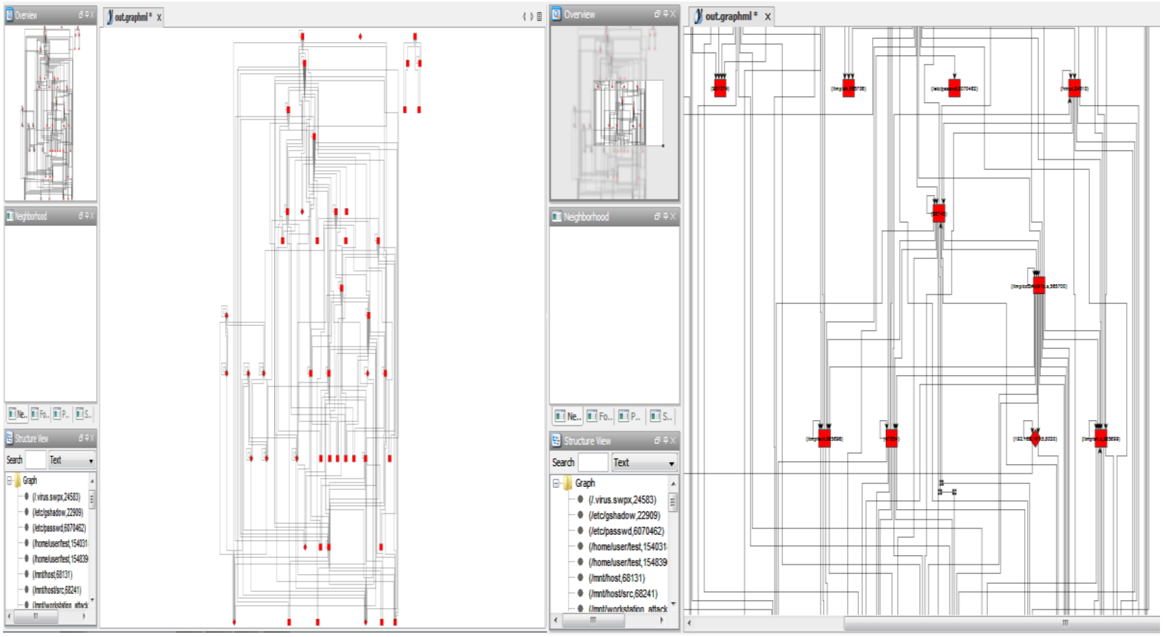


Figure.8 (a)

Figure.8 (b)

Figure.8 (a) and (b) is Yed Editor for showing output.

6. Experimental Result

The proposed method is implemented using a cloud simulator where Java is the programming language, and SQL Server is the database. CloudSim is combined with Java based IDEs such as Net Beans and Eclipse. The CloudSim library can be accessed. A software component generates and performs the virtual machines called Virtual Machine

Monitor (VMM). In this tool, Simulation parameters are used for estimating the existing and proposed techniques. It is a standard tool that offers a generalized simulation framework that permits the Cloud Computing environment and application services research. yEd is a common diagramming platform with a multi-document interface. It is a cross-platform application written in Java and used to draw various categories of diagrams such as entity-relationship diagrams, network diagrams, flowcharts. It permits the use of custom vector and raster graphics as diagram elements. The performance of the proposed method is evaluated using the various parameters, and the results obtained are explained in the section below.

Performance Metrics

The performance of the proposed work is estimated using the following metrics and shown in Figure.9.

a) **Accuracy-** Accuracy provides the required related results used for classification.

$$\text{Accuracy} = \frac{\text{TruePositive} + \text{TrueNegative}}{\text{TruePositive} + \text{FalsePositive} + \text{TrueNegative} + \text{FalseNegative}}$$

To compare the expected zero-day path vs. predicted, calculate correctness and misclassification.

b) **Correctness-** Correctness is defined as a number of correctly classified nodes by a total number of nodes.

Correctness= Number of Correctly Classified Nodes/Total Nodes

c) **Misclassification-** It is defined as the sum of False Positive and Negative, divided by the total number of nodes.

Misclassification Rate: (False Positive +False Negative)/Total

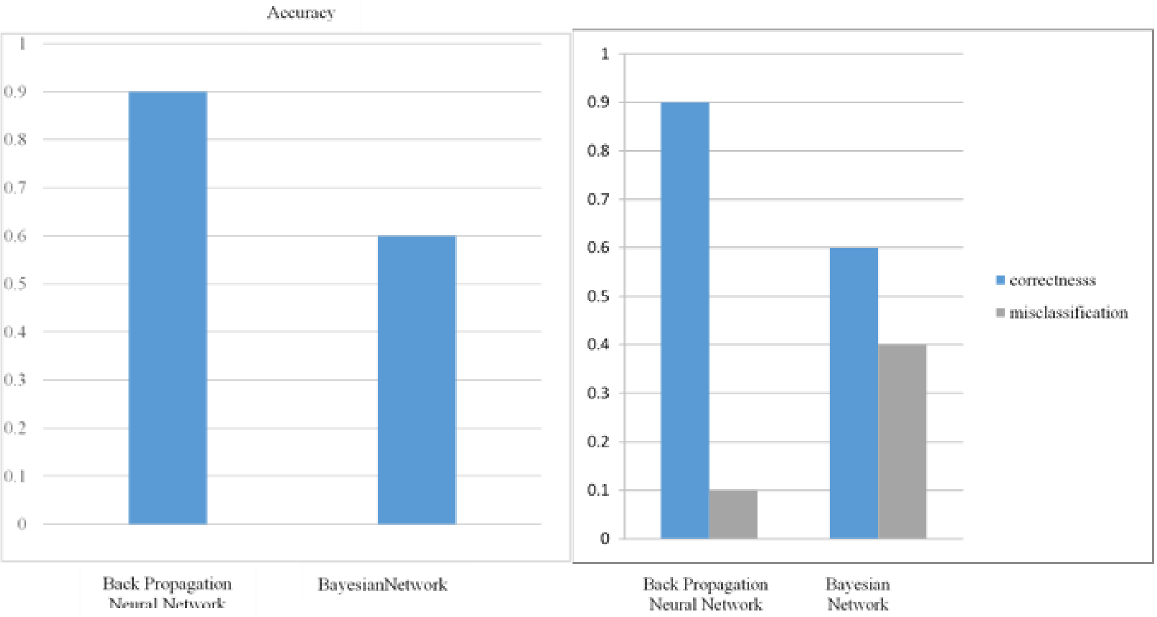


Figure.9 Performance chart

Expected Outcomes

1. Increased Efficiency in terms of Accuracy and Correctness.
2. Enhanced Data Security by accurate detection of the attack.
3. Prevent Loss of Revenue for the Organization.
4. Increases time range for accurate detection and reduces misclassification.

7. Conclusion

In the past few years, Due to the growth of Internet popularity and security-unaware, users are familiar sources for the speedy growth of attackers in the network. This work employs the hybrid detection technique to identify a zero-day attack path. This paper presented a new method that resulted from integrating several software tools aiming to observe and collect information about zero-day attacks. Experimental result shows that the proposed work achieves a better result and is comparatively far better than the existing work. In the network, numerous chain of attacks sequences are identified in the path, which consists of both zero-day exploits and non-zero day exploits is considered as the limitation. Future enhancement is to predict only the zero-day exploits and prevent further zero-day attacks.

Significant References

1. Xiaoyan Sun, Jun Dai, Peng Liu, Anoop Singhal, and John Yen., "Using Bayesian Networks for Probabilistic Identification of Zero-day Attack Paths," IEEE Transactions on Information Forensics and Security, IEEE, 2018.

2. Pawan Kumar Tiwari, P. S. . (2022). Numerical Simulation of Optimized Placement of Distibuted Generators in Standard Radial Distribution System Using Improved Computations. International Journal on Recent Technologies in Mechanical and Electrical Engineering, 9(5), 10–17. https://doi.org/10.17762/ijrmee.v9i5.369

3. Nahid Hossain et al. "SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data," USENIX Security Symposium, 2017.

4. Ghazaly, N. M. . (2022). Data Catalogue Approaches, Implementation and Adoption: A Study of Purpose of Data Catalogue. International Journal on Future Revolution in Computer Science &Amp; Communication Engineering, 8(1), 01–04. https://doi.org/10.17762/ijfrcsce.v8i1.2063

5. Shiqing Ma et al. "MPI: Multiple Perspective Attack Investigation with Semantics Aware Execution Partitioning" Proceedings of the 26th USENIX Security Symposium, 2017.

6. Bhargav R. Avasarala, Brock D. BOSE, John C. Day Donald Steiner," System and method for automated machine-learning, zero-day malware detection," BlueVectorInc, United States Patent, 2017.

7. Arellano-Zubiate, J. ., J. . Izquierdo-Calongos, A. . Delgado, and E. L. . Huamaní. “Vehicle Anti-Theft Back-Up System Using RFID Implant Technology”. International Journal on Recent and Innovation Trends in Computing and Communication, vol. 10, no. 5, May 2022, pp. 36-40, doi:10.17762/ijritcc.v10i5.5551.

8. Singh et al. "Zero-day Attacks Detection and Prevention Methods – Apriorit," 2017.

9. Gill, D. R. . (2022). A Study of Framework of Behavioural Driven Development: Methodologies, Advantages, and Challenges. International Journal on Future Revolution in Computer Science &Amp; Communication Engineering, 8(2), 09–12. https://doi.org/10.17762/ijfrcsce.v8i2.2068

10. Xiaoyan Sun, Jun Dai, Peng Liu, Anoop Singhal, John Yen, “Towards Probabilistic Identification of Zero-day Attack Paths," CNS, 2016.

11. Ravinder Kaur and Maninder Singh, "A Survey on Zero-Day Polymorphic Worm Detection Techniques," IEEE Communications Surveys and Tutorials, 2014.

12. A. Mishra and B. B. Gupta," Hybrid Solution to Detect and Filter Zero-day Phishing Attacks," International Conference on Emerging Research in Computing, Information, Communication and Applications, Elsevier, 2014.
13. Wang et al. "k-Zero-Day Safety: A Network Security Metric for Measuring the Risk of Unknown Vulnerabilities," IEEE Transactions on Dependable and Secure Computing, Volume: 11, Issue: 1, P.no: 30 - 44, 2014.
14. Mikhail Zolotukhin and Timo Hamalainen, "Detection of Zero-day Malware Based on the Analysis of Opcode Sequences" IEEE 11th Consumer Communications and Networking Conference (CCNC), 2014.
15. P. M. Paithane and D. Kakarwal, "Automatic Pancreas Segmentation using A Novel Modified Semantic Deep Learning Bottom-Up Approach", Int J Intell Syst Appl Eng, vol. 10, no. 1, pp. 98–104, Mar. 2022.
16. Nouby M. Ghazaly, A. H. H. . (2022). A Review of Using Natural Gas in Internal Combustion Engines. International Journal on Recent Technologies in Mechanical and Electrical Engineering, 9(2), 07–12. https://doi.org/10.17762/ijrmee.v9i2.365
17. Jun Dai, Xiaoyan Sun, and Peng Liu, "Patrol: Revealing Zero-Day Attack Paths through Network-Wide System Object Dependencies," Springer, 2013.
18. Ananthakrishnan, B., V. . Padmaja, S. . Nayagi, and V. . M. "Deep Neural Network Based Anomaly Detection for Real Time Video Surveillance". International Journal on Recent and Innovation Trends in Computing and Communication, vol. 10, no. 4, Apr. 2022, pp. 54-64, doi:10.17762/ijritcc.v10i4.5534.
19. Yuan Ning, Yufeng Liu, Qiang Ji., "Bayesian - BP Neural Network Based Short-term Load Forecasting for Power System," 3rd International Conference on Advanced Computer Theory and Engineering (1CACTE), IEEE, 2010.