Regular Paper

IoT-PEN: An E2E Penetration Testing Framework for IoT

Geeta Yadav[1,a] Kolin Paul[1,b] Alaa Allakany[2,3,c] Koji Okamura[4,d]

Received: November 8, 2019, Accepted: June 1, 2020

Abstract: The lack of inbuilt security protocols in cheap and resource-constrained Internet of Things (IoT) devices give privilege to an attacker to exploit these device's vulnerabilities and break into the target device. Attacks like Mirai, Wannacry, Stuxnet, etc. show that a cyber-attack often comprises of a series of exploitations of victim device's vulnerabilities. Timely detection and patching of these vulnerabilities can avoid future attacks. Penetration testing helps to identify such vulnerabilities. However, traditional penetration testing methods are not End-to-End, which fail to detect multi-hosts and multi-stages attacks. Even if an individual system is secure under some threat model, the attacker can use a kill-chain to reach the target system. In this paper, we introduced first-of-its-kind, *IoT-PEN*, a Penetration Testing Framework for IoT. The framework follows a client-server architecture wherein all IoT nodes act as clients and "a system with resources" as a server. *IoT-PEN* is an End-to-End, scalable, flexible and automatic penetration testing framework for discovering all possible ways an attacker can breach the target system using target-graphs. Finally, the paper recommends patch prioritization order by identifying critical nodes, critical paths for efficient patching. Our analysis shows that *IoT-PEN* is easily scalable to large and complex IoT networks.

Keywords: internet of things security, penetration testing

1. Introduction

The fifth generation wireless systems (5G) are expected to provide an Internet of things (IoT) specific connectivity interface [1]. IoT provides a platform to connect the physical devices and everyday objects over the internet. Most of the IoT devices use very cheap sensors/ actuators, which are unlikely to be secure. Therefore, over time, IoT is creating a completely new and complex set of problems for the security community. Different from enterprise systems, securing large scale, resource-constrained IoT devices is a very challenging task. If we analyze some of the recent attacks on IoT, they exploit a series of vulnerabilities on different systems to reach the target system, e.g., WannaCry, in 2017, was initially carried out using a malicious PDF (malware embedded). It further exploited the vulnerability of the SMBv2 protocol that executed a dropper code and made a connection to an unregistered IP address. Also, it created a mssecsvc2.0 service and made an entry in the registry. It leads to encrypting local files and asking a ransom note for $300 in Bitcoin [2].

In 2016, a DDoS attack on the Dyn DNS company and French service provider OVH exhibited that an attacker can turn the Internet of Things into the Internet of Vulnerabilities (IoV) [3]. The attacker exploited default passwords and the outdated TELNET service to get control of millions of web cameras that were manufactured by a specific Chinese company. In 2010, a very sophisticated attack Stuxnet was reported on Iranian nuclear centrifuges [4], [5]. It entered the network via a USB stick (air-gapped network) and replicated itself to all connected machines running Microsoft windows. It then searched a specific version of Siemens Step7 Industrial Control Systems (ICS) and modified the Program Logic Controllers (PLC). The main target for Stuxnet was Siemens Step7 system and it would sit silently on other systems.

A brief study of Wannacry, DDoS, Stuxnet shows that a series of unrelated attack events that are logically connected can provide an attack-path to the target node, even though there is no direct path to the target node. It highlights the need for defining the notion of usable security for large scale IoT networks.

Definition 1 A Secure SoS, is defined as an SoS where there is no path from source to target despite all systems having one or more vulnerabilities.

The system administrator needs to secure the end-to-end system. To do a timely detection of vulnerabilities/ possible attacks, system administrators prefer penetration testing of the systems periodically. The penetration testing is an authorized simulated cyber-attack on a computer system.

State-of-the-art penetration tools, i.e., Netsparker, Acunetix, Probably, BackTrack, Idappcom Metasploit, Nessus, etc. provide the functionality only to the subscribed users. Open-source tools, i.e., Wapiti, ZAP (Zed Attack Proxy), Vega, W3af, etc. have minimal functionality and are targeted to a particular threat model. Almost all of these techniques are non-automatic and perform isolated Pentesting. These solutions fail in the case of IoT, where the devices are quite heterogeneous. Performing penetration testing manually on a large number of IoT devices is a challenging task for system administrators. The traditional penetration testing systems are targeted to the pentesting of a system individually,

1 Indian Institute of Technology, New Delhi, India
2 Cybersecurity Center, Kyushu University, Fukuoka 819–0395, Japan
3 Faculty of Computers and Information, Kafrelsheikh University, Egypt
4 Research Institute for Information Technology Kyushu University, Fukuoka 819–0395, Japan
a) anjugeeta11@gmail.com
b) kolin@cse.iitd.ac.in
c) alaa_83moh@yahoo.com
d) oka@ec.kyushu-u.ac.jp

which fails to detect multi-stage multi-host attacks. This highlights an urgent need of new algorithms, tools, frameworks for securing such resource constrained devices.

To overcome these weaknesses, we propose *IoT-PEN* - A penetration testing framework for IoT. *IoT-PEN* is an automatic, flexible and End-to-End (E2E) testing framework. The framework assumes a server-client architecture with "a system with resources" as a server and "IoT nodes" as clients. In *IoT-PEN*, a customized script runs over an SoS and finally reports possible exploitations. Flexible *IoT-PEN* specifies a plug and play concept for penetration-testing. *IoT-PEN* consists of different modules to take into account the heterogeneity of IoT systems. A user can select the required modules and a customized framework is generated. Apart from this, the main motivation of *IoT-PEN* is the end-to-end penetration framework. Currently, all proposed approaches focus on the individual system or component testing, which fails to detect multi-host, multi-stage vulnerabilities. In addition, *IoT-PEN* recommends the patch prioritization order by identifying critical nodes, critical paths for efficient patching. *IoT-PEN* mainly focuses on securing the target node i.e., an IoT terminal-node. The proposed framework works well for a well managed IoT system or Industrial Internet of Things (IIoT). *IoT-PEN* penetration testing framework for IoT is based on target-graphs and the NVD database.

Target graphs are similar to attack-graphs [6] except for that target-paths start from some random node and try to reach the target node and target-paths start from the target node and check all possible sources through which the target can be intruded. System administrators usually evaluate the system security using target graphs to identify the source of their system's vulnerabilities and to decide which security measures should be taken to defend their systems. Each path in a target-graph is a sequence of exploits that leads to an undesirable state, e.g., obtaining administrative access to a critical host.

Now, if we observe the target graphs for IoT networks, these are expected to be very complex and have numerous target-paths. Patching/Fixing each possible target-path is a challenging/impossible task and not all target-paths lead to the attacker getting root privilege of the target system. We need to identify critical paths, critical nodes and critical vulnerabilities [7], followed by the fixing of these critical nodes. Most importantly, even if we patch the critical severity vulnerabilities, an attacker can exploit multiple non-critical vulnerabilities and exploit the targeted system. Therefore, we need a prioritization order to patch these complex networks efficiently. So, the last stage of*IoT-PEN* is responsible for generating recommendations in terms of critical vulnerability, node and path.

IoT-PEN uses the NVD database, which is regularly updated and is a comprehensive database, to identify the possible vulnerabilities on a particular version of the software. NVD maintains examined information about software and hardware vulnerabilities of different domains [8]. It considers the product name, the product version and the vender. It indexed vulnerabilities to Common Vulnerability Enumeration (CVE) [9] Ids (CVE-ID) that helps to provide a common name for publicly available vulnerabilities. The NVD team analyzes each reported CVE using the Common Vulnerability Scoring System (CVSS) framework. Each CVE is linked with a Common Platform Enumeration (CPE) also that indicates the vulnerable component. CPE is a structured naming scheme generally used for information technology systems, packages and software [10]. Apart from NVD, a separate, standardized and comprehensive directory Common Weakness Enumeration (CWEs), which list weaknesses found in software, has been created by MITRE. The main aim of CWE is to understand the weaknesses in software and for developing automated tools for fixing and prevention of the lacunae. The information provided by NVD and CWE plays a vital role in understanding trends and patterns in software and hardware vulnerabilities.

1.1 Contributions

Major contributions of the paper are enumerated below:

(1) *IoT-PEN* is a novel flexible, scalable, automatic and E2E Penetration testing framework for the resource-constrained IoT devices. It identifies all possible paths from where an attacker can reach to the target node.

(2) We propose a novel mechanism to perform the penetration testing, particularly for the IoT system using a client-server architecture without the intervention of the system administrator. Also, it is capable of identifying the insider as well as possible outsider attacks.

(3) We evaluated the *IoT-PEN* performance and scalability in terms of the execution time vs. the number of nodes, of number of target-paths generated vs. the number of nodes in the network, the number of attack paths vs. the vulnerabilities on the nodes.

(4) We performed extensive experiments to find the prioritization order of the vulnerabilities using the following security metrics, i.e., density-based prioritization, CVSS score based prioritization, betweenness centrality and PatchRank.

In the next section, we describe an example that mimics a scaled-down version of an IoT system. This system demonstrates the problem using reported vulnerabilities on a Philips Hue based IoT system. We use this system as a running example in the paper. This is followed by the detailed discussion of design and implementation of *IoT-PEN* in Section 3. *IoT-PEN* performance evaluation is done in Section 4 followed by a brief study of related work in Section 5.

2. Problem Setup

An IoT system consists of sensors, actuators, bridges, gateways, routers, etc. Each component is itself a complete system. Therefore, we can consider an IoT network equivalent to an SoS. We use an example of an actual small IoT system (Phillips Hue bulbs system) consisting of a smart bulb, Hue bridge, IoT gateway, server, mobile application. The communication links between all these components and reported vulnerabilities (at least one) are shown in **Fig. 1**. The generation of prerequisites and post-conditions for each vulnerability is explained in detail in Section 3. We follow the Definition 1 to define a secure system, i.e., even if each node has one or more vulnerabilities, the absence of any target-path to the target node (bulb) is sufficient to

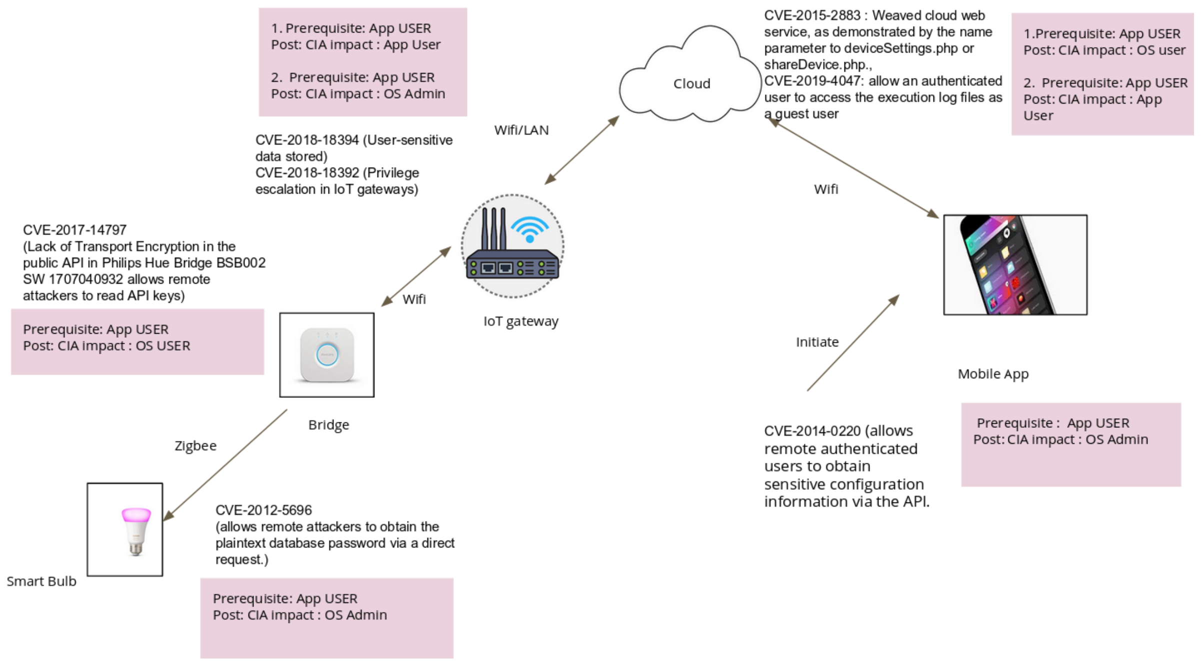
1. Prerequisite: App USER
Post: CIA impact : App User
2. Prerequisite: App USER
Post: CIA impact : OS Admin
CVE-2018-18394 (User-sensitive data stored)
CVE-2018-18392 (Privilege escalation in IoT gateways)
Wifi/LAN
Cloud
CVE-2015-2883 : Weaved cloud web service, as demonstrated by the name parameter to deviceSettings.php or shareDevice.php.,
CVE-2019-4047: allow an authenticated user to access the execution log files as a guest user
1.Prerequisite: App USER
Post: CIA impact : OS user
2. Prerequisite: App USER
Post: CIA impact : App User
Wifi
CVE-2017-14797 (Lack of Transport Encryption in the public API in Philips Hue Bridge BSB002 SW 1707040932 allows remote attackers to read API keys)
Prerequisite: App USER
Post: CIA impact : OS USER
Wifi
IoT gateway
Initiate
Mobile App
Bridge
Zigbee
CVE-2014-0220 (allows remote authenticated users to obtain sensitive configuration information via the API.
Prerequisite : App USER
Post: CIA impact : OS Admin
CVE-2012-5696 (allows remote attackers to obtain the plaintext database password via a direct request.)
Smart Bulb
Prerequisite: App USER
Post: CIA impact : OS Admin

Fig. 1 Multi-host, multi-stage vulnerability exploitation example with pre and post condition.

define the target as a secure system.

In this example, the attacker wants to get control of the smart bulb and she has user-level access on a mobile phone, which has Philips Hue application running. The attacker can get the cloud API authentication key by exploiting vulnerability CVE-ID CVE-2014-0220 on the mobile phone on which the Philips Hue app is running. The attacker logs into the cloud as an authenticated user exploiting vulnerability CVE-ID CVE-2015-2883. Now, she can weave cloud web service for connected device/application settings followed by the access of execution log exploiting vulnerability CVE-ID CVE-2019-4047. Further, she searches for connected devices and logs in using default user login-passwords and able to get user-sensitive data exploiting CVE-ID CVE-2018-18394. She is able to get root access exploiting CVE-ID CVE-2018-18392. Further, she modifies the data being transmitted to the bulb, exploiting the vulnerability with CVE-ID CVE-2017-14797. It specifies that even though an attacker is unable to attack the bulb control directly, it can successfully take control of the bulb by exploiting multi-host, multi-stage vulnerabilities. It is not possible to identify these sequences of attacks using individual system penetration testing only. Therefore, we need a framework that can handle these scenarios.

In the next section, we discuss the design and implementaion of *IoT-PEN* in detail.

3. Design and Implementation

We follow the single responsibility principle, as we divide *IoT-PEN* framework into independent micro-services, capable of running on their own. The modules are mostly hexagonal, following the open-closed principle - open to extensibility and closed to modification. This approach allows for horizontal expansion, both conceptually and infrastructure-wise. A target node is a terminal node (smart bulb, smart refrigerator, smart lock etc.) in a hierarchical architecture of IoT, which is to be secured from an attacker. *IoT-PEN* focuses on penetesting an IoT application E2E.

3.1 Design (Target Graph Structure):

We use the following nomenclature along with the corresponding definitions from NVD database.
Locality of the attacker: {Network, adjacent, local, physical}
Authentication = {None, Required}
Privilege required = Privilege gained = {none, user, root}
CPE: {Application, Operating System, Hardware}.
$Vul := \{vul_1, vul_2, ..., vul_m\}$ is a set of all vulnerabilities from the NVD dataset.

Target graph is a digraph $G_A = (P, Q)$, where P denotes the set of nodes and Q denotes a set of edges. $P := \{v = d : d \epsilon D\}$, where D is a list of devices. An edge (p_i, p_j, vul_k), <Source Node, Target Node> denotes that vulnerability vul_k of node p_j can be exploited by an attacker who has gained access of node p_i. For this, p_i should satisfy the locality required to exploit the vulnerability and post-condition of node p_i should be sufficient to exploit vul_k on node p_j.

A vulnerability is considered as a CVE entry and represented by a tuple with three elements $<CVE_{ID}$, prerequisite, Postcondition>.

The **prerequisite** of a vulnerability $vul_k \epsilon Vul$, is a function of {Vulnerability Description, Attack Vector (Locality), Authentication, Privilege, CPE}. Prerequisite denotes the required attacker Privilege to enter into the system.

The **post-condition** of a vulnerability $vul_k \epsilon Vul$ where $Vul := \{vul_1, vul_2, ..., vul_m\}$, is a function of {Vulnerability Description, Impact of vulnerability, Privilege, CPE}. Post-condition denotes the attacker Privilege acquired after exploiting vul_k.

3.2 Implementation

The framework consists of 5 stages, a detailed discussion of each stage is as follows:

Input: The input given to the *IoT-PEN* is the network topology. The Network topology contains the unique id of all devices (in our case, we are considering Internet Protocol (IP) addresses) and the direct reachability matrix.

Stage 1 (Pentesting setup installation): In *IoT-PEN*, we follow a client-server architecture. All the client nodes and server are prepared for pen-testing. A patch (installation of the vulnerability scanner tools, set up to listen to the server command and act accordingly) is applied to all the nodes in the topology. All the required libraries/ tools also need to be installed on the server.

Stage 2 (Get current state information of each node): The server node multi-cast command to all the nodes to be pen tested to share their current state information. IoT nodes respond with their state information to the server for further processing. *IoT-PEN* can be easily extended to be compatible with IoT protocols. For our experiments section, we used the Nmap vulnerability scanner for extracting the vulnerabilities and the MQTT protocol for sharing the IoT node's current state information.

Stage 3 (Extract CPE from .xml file generated by Nmap): In this step, we parsed the XML file generated by the vulnerability scanner tools and extracted the current state of nodes in terms of operating system and applications/ services version and vendor info followed by generating CPE from collected state info. We locally synched the NVD dataset for fast processing. The NVD dataset containing CVE-CPE mapping is changed to a dataset with CPE-CVE mapping for improving the time complexity.

Stage 4 (Prerequistics and post-conditions generation for all the reported vulnerabilities & Target-graph generation): In the previous stage, we have collected all the possible vulnerabilities on the target systems. The prerequisite of a vulnerability is a function of privilege required by the attacker, user Interaction and the locality of the attacker. CVSS provides these parameters for each reported vulnerability.

In CVSS V3, privilege denotes the privilege level required by the attacker to exploit a given vulnerability. The values assigned to this parameter are Low, High and None (user, kernel and none), where a Low value denotes that only basic user capabilities are required; a High value denotes that to exploit a given vulnerability a significant control over the assets is required. None denotes that no privilege is required to exploit the vulnerability.

The locality of the attacker is defined as "attack vector" in CVSS V3, which denotes the vicinity of attacker w.r.t. the vulnerable component. The values assigned to the locality are "Network, Adjacent Network, Local and Physical". User Interaction denotes whether interaction is required or not (None and Required).

The Post-condition is a function of the privilege gained and the Network reachability. In the NVD database, there is no parameter in CVSS analysis, which denotes the privilege gained by the attacker. Therefore, the privilege gained is found out using Impact, description and vulnerable platform from CPE parameter from NVD; where Impact in terms of Confidentiality, Integrity and Availability (CIA) triad denotes damages induced at the target system. In vulnerability description, we look for keywords ("gain root", "gain unrestricted root shell access", "obtain root", "gain unauthorized access", etc.) as discussed in Table 2. CPE categorizes the vulnerable component into operating systems, firmware and applications. Correlating all these parameters, we can devise the privilege gained by the attacker after exploiting a particular vulnerability. We followed a Modified approach of rule-based prerequisite and post-condition generation proposed in Ref. [11]. The modified approach for prerequisite and Post-conditions for these vulnerabilities are based on rules specified in **Table 1**, **Table 2** and **Table 3**. Then we generate target paths using Algorithm 1. Procedure PENTEST_NODE describes the individual node pen-testing. Procedure E2E_PREMODULE calculates the minimum privilege required to login to the systems and the maximum privilege gained after exploiting multi-stage vulnerabilities. Procedure E2E_MODULE generates the $Target_{paths}$. Multi-stage attacks are based on the exploitation of the different vulnerabilities on the same host node. In this case, there will be an edge (d_i, d_i, vul_k) in the target-graph if the current privilege level of node d_i matches with the prerequisite of vul_k. The multi-host attack is based on exploiting vulnerabilities on different nodes. In this case, an edge (d_i, d_j, vul_k) in the target-graph if the current privilege level of node d_i matches with the prerequisite of vul_k and devices d_i, d_j are network reachable.

Table 1 Rules for generating prerequisites using locality, authentication and privilege.

Locality of attacker	User interaction	Privilege	CPE	prerequisite
-	-	None	-	None
Local	-	High	OS	Admin OS
Local	None	High	App	Admin OS
Network	Required	High	OS	Admin OS
Local	-	Low	OS	User OS
Local	None	Low	APP	User OS
Network	Required	Low	OS	User OS
Local	Required	High	App	Admin App
Network	Required	High	App	Admin App
Local	Required	Low	App	User App
Network	Required	Low	App	User App

Stage 5 (Analysis of target-paths & Recommendations): In stage 4, we generated target graphs followed by identifying all possible target paths to a specific node. Target-graphs generated from target paths are expected to be very complex for an IoT network. So, patching/Fixing each possible target-path is a challenging task and all target-paths do not lead the attacker to gain the root privilege of the target system. Therefore, we need a prioritization order to secure critical systems on a priority [7]. This stage of *IoT-PEN* is responsible for generating recommendations in terms of critical vulnerability, node and path. The prioritization order considers the criticality of the path, the node and the vulnerability.

(1) All the possible paths through which an attacker can attack the Target system.
(2) Optimization techniques for the Target-paths.
  (a) Identify Critical (Path, Node, Vulnerability) using various approaches explained in Algorithm 2.
  (b) Patching order for these vulnerabilities.

For optimizing these target graphs, we define some security metrics as follows:

Node count based prioritization: In this case, critical target-

Table 3 Rules for generating post-conditions using the NVD dataset. CPE - O (Operating System), A (Application) , Impact score (5.9 (All CIA impact value), [3.4 - 5.9))) (Partial CIA impact), <3.4 (Some CIA impact value None)).

Description	Impact Score	CPE	PostCondition
“gain root” or “gain unrestricted, root shell access” or “obtain root” or “gain privilege” or “gain host OS privilege” or “gain admin” or “obtain local admin” or “obtain admin” or “gain unauthorized access” or “to root” or “to the root” or “elevate the privilege” or “elevate privilege” or “root privileges via buffer overfow” or “unspecified vulnerability” or “unspecified other impact” or “unspecified impact” or “other impacts” or “buffer overflow” or “command injection” or “write arbitrary, file” or “command execution” or “execute command” or “execute root command” or “execute commands as root” or “execute arbitrary” or “execute dangerous” or “execute php” or “execute script” or “execute local” or “execution of arbitrary” or “execution of command” or “remote execution” or “execute code” &! “execute arbitrary SQL”	5.9	ANY	Admin OS
“obtain password” or “obtain credential” or “sniff ... credentials” or “sniff ... passwords” or “steal ... credentials” or “steal ... passwords” or “cleartext credential” or “cleartext password” or “obtain plaintext” or “obtain cleartext” or “discover cleartext” or “read network traffic” or “un-encrypted” or “unencrypted” or “intercept transmission” or “intercept communication” or “obtain and decrypt passwords” or “conduct offline password guessing” or “bypass authentication”	5.9	O	Admin OS
“gain privilege” or “gain unauthorized access” or “unspecified vulnerability” or “unspecified other impact” or “unspecified impact” or “other impacts” or “obtain password” or “obtain credential” or “sniff ... credentials” or “sniff ... passwords” or “steal ... credentials” or “steal ... passwords” or “cleartext credential” or “cleartext password” or “obtain plaintext” or “obtain cleartext” or “discover cleartext” or “read network traffic” or “un-encrypted” or “unencrypted” or “intercept transmission” or “intercept communication” or “obtain and decrypt passwords” or “conduct offline password guessing” or “bypass authentication” or	3.4 - 5.9	O	User OS
“gain admin” or “obtain admin” or “obtain password” or “obtain credential” or “sniff ... credentials” or “sniff ... passwords” or “steal ... credentials” or “steal ... passwords” or “cleartext credential” or “cleartext password” or “obtain plaintext” or “obtain cleartext” or “discover cleartext” or “read network traffic” or “un-encrypted” or “unencrypted” or “intercept transmission” or “intercept communication” or “obtain and decrypt passwords” or “conduct offline password guessing” or “bypass authentication” or “SQL injection”	3.4 - 5.9	A	Admin App
“hijack the authentication of users” or “hijack the authentication of arbitrary users” or “hijack the authentication of unspecified victims”	-	-	User App
-	<3.4	-	None

Table 2 Rules for generating prerequisites using the description field (A-Application).

Description	CPE	Locality of attacker	Privilege	Prerequisite
“allows local administrators” or “allow local administrators” or “allows the local administrator”	-	local	High	Admin OS
“allows local users” or “allowing local users” or “allow local users” or “allows the local user”	-	local	High	User OS
“remote authenticated admin” or “remote authenticated users with administrative privileges”	A	Network	Low	Admin App
“remote authenticated user” and ! “remote authenticated users with administrative privileges”	A	Network	Low	User App

paths are defined as paths with the least number of nodes. The critical node on the critical target-path is defined as the node with the maximum number of target-paths passing. The critical vulnerability on the critical node is defined as a vulnerability with maximum CVSS severity scores.

CVSS based prioritization: CVSS provides an exploitability score for each reported vulnerability, which depicts the easiness to exploit a vulnerability and its score lies in (0,10), wherein the higher the exploitability score, the easier it is to exploit the system. A path criticality score is defined as the maximum of the average severity scores of all nodes in a target path. A node criticality score is defined as the maximum of an average severity score of all vulnerabilities on that node. The critical vulnerability on the critical node is defined as a vulnerability with maximum CVSS scores.

PatchRank based prioritization [12]: This algorithm is based on an attacker-defender game-theoretical model. It considers an attacker-defender behavior based on NVD database parameters, the node priority as well as the network topology to decide the prioritization list. The attacker-defender scenario is formulated as a mixed Nash equilibrium scenario, where attackers and defenders compete with each other to increase their payoff. In this metric, the critical path is defined as a path with the maximum number of critical nodes. A critical node is identified as a node with the maximum probability of exploitation considering the attacker-defender model. A critical vulnerability on a critical node is identified as a vulnerability with the maximum probability of exploitation considering the attacker-defender model defined in Ref. [12].

Betweenness centrality (BC) based prioritization [13]: The critical node in a critical path is defined as a node through which a maximum number of target paths cross. A critical vulnerability is defined as vulnerability that will be exploited in the maximum number of the target-paths. Therefore, the criticality of a node relies on the indegree of a node as well as edge weight, leading us to a BC measure. A critical path is defined as a path with the maximum number of critical nodes.

In the next section, we analyze the performance & scalability

Algorithm 1 IoT-PEN Algorithm

```
Input:
    $Node_{List}$                 ▷ List of all the nodes in the network to be pentested.
    $Net_{topology}$              ▷ Network topology provided by the system administrator.
Output:
    $Target_{paths}$

procedure IoT_PEN($Node_{List}$ , $Net_{topology}$,)
(1)  Scan_reports = $Pentest\_node(Net_{topology})$
(2)  $(Vul_{Pre}, Vul_{Post}, Net_{topology}, minPriv, maxPriv, maxfrommin, Target\_node, vulnerability\_list, PC, PR) = E2E\_Premodule(Scan\_reports, Node_{List}, Net_{topology})$
(3)  $Target_{paths} = E2E\_module(Vul_{Pre}, Vul_{Post}, Net_{topology}, minPriv, maxPriv, maxfrommin, Target\_node, vulnerability\_list, PC, PR)$
    return $Target_{paths}$
end procedure

procedure PENTEST_NODE($Net_{topology}$)
(1)  Apply patch to IoT nodes, where patch contains open-source tools installation to get each node state.
(2)  Server publishes instructions to IoT nodes to scan their current state of operating system and services.
(3)  IoT nodes perform scanning and share the output file (.xml) to the server = Scan_reports[node]
    return Scan_reports[node]
end procedure

procedure E2E_PREMODULE (Scan_reports, $Node_{List}$, $Net_{topology}$)
(1)  Parse Scan_reports[node] received at server to get CPE for operating system and services for each node = $CPE_{list}$
(2)  NVD data have a mapping of CVE to CPE list. We converted it from CPE to CVE list. Also based on the rules specified in Table 1, Table 2 and Table 3, for each
       vulnerability, prerequisite $Vul_{Pre}$ and post-condition $Vul_{post}$ are generated .
(3)  $vulnerability\_list$[node] = Extract CVE ids w.r.t. CPE list in the $CPE_{list}$ for each node.
(4)  For each vulnerability j on node i (in $nodevul_{list}$), find prerequisite ($PR_{ij}$) and post-condition ($PC_{ij}$) as generated in Step 2.
(5)  For each node in $Net_{topology}$ find :
       (a)  $minPriv[node]$ = Minimum privilege needed to exploit any vulnerability on the node.
       (b)  $maxPriv[node]$ = Maximum privilege gained after exploiting multi-stage vulnerability.
       (c)  $privgainedmax_{pre}[node]$ = prerequisite corresponding to maximum privilege gained.
       (d)  $maxfrommin[node]$ Can maximum privilege be achieved by the exploitation of vul-chain?
    return $(Vul_{Pre}, Vul_{Post}, minPriv, maxPriv, privgainedmax_{pre}, maxfrommin, vulnerability\_list, PC, PR)$
end procedure

procedure E2E_MODULE ($Vul_{Pre}, Vul_{Post}, Net_{topology}, minPriv, maxPriv, maxfrommin, Target\_node, vulnerability\_list$, PC,PR)
(1)  $current\_node = Target\_node.pop()$, $Target\_paths$ = []
(2)      while $current\_node != null$ do
(3)          $vul_{list} = vulnerability\_list[current\_node]$
(4)          Extract $minPriv[current\_node], maxPriv[current\_node], privgainedmax_{pre}[current\_node], maxfrommin[current\_node]$ .
(5)          Calculate privilege gained after multi-stage vulnerability exploitation .
(6)          $neighbour_{node}$ = Extract nodes reachable from current_node using $Net_{topology}$
(7)          for each node in $neighbour_{node}$ do
(8)              vul_list = vulnerability[node]
(9)              for each vul in $vul\_list$ do
(10)                 if maxPossiblefrommin[node] is true and vul.locality == current_node.locality then
(11)                     if $minPriv[current\_node] > PC_{node,vul}$ or $minPriv[current\_node] == 0$ then Target_paths.add(node, current_node, vul)
(12)                     end if
(13)                 end if
(14)             end for
(15)         end for
(16)     end while
    return $Target_{paths}$
end procedure
```

Algorithm 2 Critical Path, Node, Vulnerability Selection

```
Input:
    $Target\_paths$          ▷ $Target\_paths$ generated by Algorithm 1
    $Target\_graph$          ▷ $Target\_graph$ corresponding to $Target\_paths$
Output:
    Critical path, Critical node, Critical vulnerability

procedure CRITICAL($target\_graph$, $target\_paths$, metric)
(1)  if metric == Node count
       (a)  $Vul_{critical}$ = max (Exploitability score on a critical node) (From NVD Database)
       (b)  $Node_{critical}$ = max (Number of attack paths passing through the node)
       (c)  $Path_{critical} = Target_{paths}$ with min (number of nodes in the $target\_paths$)
(2)  else if metric == CVSS
       (a)  $Vul_{critical}$ = max (Exploitability score on a critical node) (From NVD Database)
       (b)  $Node_{critical} = max(\sum_{i=1}^{N} Vul_{exploitability}/N)$
       (c)  $Path_{critical} = Target\_paths$ with max $(\sum_{i=1}^{M} Node_{exploitability}/M)$
(3)  else if metric == PatchRank
       (a)  $Vul_{critical}$ = max (Probability of exploitation of vulnerability considering attacker-defender model) (From NVD Database)
       (b)  $Node_{critical}$ = max (Probability of exploitation of node considering attacker-defender model)
       (c)  $Path_{critical}$ = max (Number of critical nodes in $Target\_paths$)
(4)  else if metric == Betweeness centrality
       (a)  $Vul_{critical}$ = Attack paths exploiting that vulnerability / Total number of attack paths passing through the node
       (b)  $Node_{critical}$ = Attack paths passing a particular node / Total number of attack paths
       (c)  $Path_{critical}$ = max (Number of critical nodes in $Target\_paths$)
    return $Vul_{critical}, Node_{critical}, Path_{critical}$
end procedure
```

of *IoT-PEN*. We also examine various security matrices explained in Section 3.2 for a complex IoT network.

4. Performance & Scalability

We measured *IoT-PEN* performance on a Ubuntu Linux machine (Version - 16.04.1 LTS (Xenial Xerus), 16 GB RAM). For experiments, we dynamically created nodes and network topologies. We measured the time taken by individual steps of *IoT-PEN* (i.e., Information gathering (Node state using MQTT protocol), Database Sync, CPE-CVE mapping generation, Prerequisites and Post-conditions generation and finally target-graph generation) to verify the applicability of *IoT-PEN* to a complex IoT network.

The proposed method is evaluated to ensure the applicability of *IoT-PEN* in a IoT network. The time taken by each step is shown in **Table 4**. To gather information about each node state, one needs to apply a patch on every node. Afterward, the server node collects each node states in parallel. We repeated each experiment five times and then took the average to reduce errors in the experimental values.

For analyzing information gathering time, we used a raspberry pi-3 (IoT terminal-node). Initially, the server will send a command to the IoT terminal-node to collect the state information using the Nmap tool. Nmap generates a .xml file containing information of all the installed software (version, vendor) and operating system (name, version). This file is transferred to the server for further processing. We defined information gathering time as the time interval between terminal-node state information request and response received on the server. We measured the time taken for information gathering on IoT nodes, which takes 20.41 seconds on an average. In parallel, we can execute step 2 that locally creates a database for NVD CVEs taking 299.23 seconds the first time; for later steps, we can use a script to sync only the updated data. For generating CPE-CVE mapping and the generation of prerequisites and post-condition for each node takes around 53 and 15 seconds, respectively. Therefore, for the initial setup, 367.23 seconds are spent for data-sync and converting data to the desired form.

Next, we observed the time-taken to generates all possible target paths by varying the number of nodes in the network from 5 to 1,000, as shown in **Fig. 2**. Each node has five vulnerabilities and these vulnerabilities are randomly assigned from the NVD database. For a single node, it takes 54 seconds to generate Target-graphs, which increases linearly with the number of nodes to 400 nodes. Beyond that, it drastically increases to a large value, indicating a direct increase in the search space that increases the probability of a path from a source to a destination. The number of target-paths generated varies directly with the number of nodes, as shown in **Fig. 3**. However, keeping the number of nodes in network constant and varying number of vulnerabilities, the number of target paths is initially low. Then it suddenly goes high and then linearly varies with the number of vulnerabilities, as shown in **Fig. 4**. The sudden increase may be due to more probability of successful hoping of the attacker from the end nodes to the target node. However, if we see, target paths to the root node only, then there is no such variance due to a fixed amount of vulnerabilities on the target node. Observing Fig. 2, we

Table 4 Time elapsed in various stages.

Step	Stage	Time elapsed
1	Information gathering (Node state)	20.41 sec (For one node)
2	Database Sync	299.23 sec
3	CPE-CVE mapping generation	53 sec
4	Pre-Post conditions generation	15sec

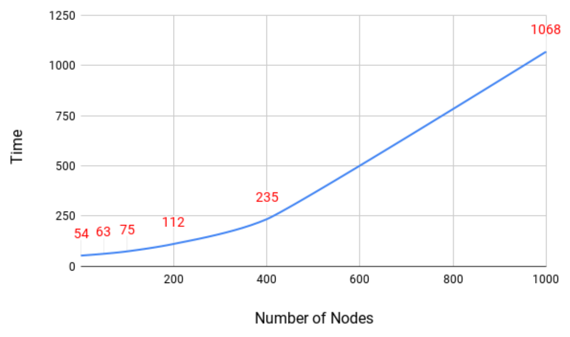

Fig. 2 Running time vs. number of nodes (Number of vulnerabilities on each node = 5).

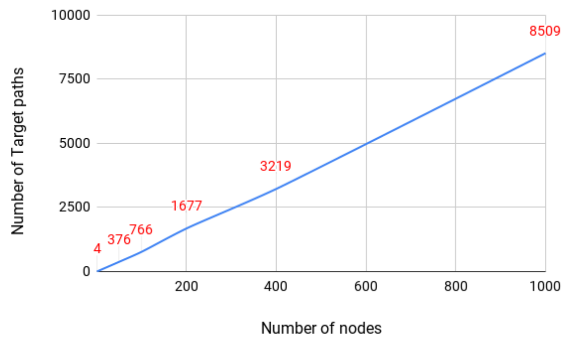

Fig. 3 Target-paths generated vs. number of nodes (Number of vulnerabilities on each node = 5).

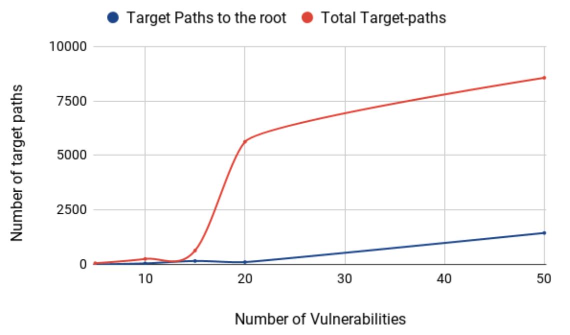

Fig. 4 Number of target-paths vs. number of vulnerabilities (Number of nodes = 10).

can see that time taken for finding the target-paths to the target node, with the varying number of nodes keeping vulnerabilities static. In **Fig. 5**, the number of vulnerabilities is varying keeping the number of nodes static. The time taken to find target-paths in (1 node, ten vulnerabilities) and (10 nodes, 1 vulnerability) is approximately the same. This is due to the fact that *IoT-PEN* is checking multi-host and as well as multi-stage vulnerability exploitations. However, if we see target paths to the target (root node) only then, it varies linearly with the number of vulnerabili-

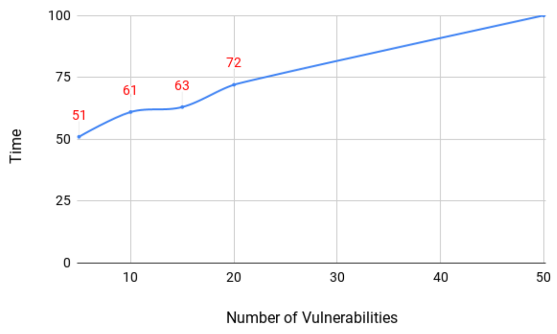

Fig. 5 Running time of IoT-PEN vs. number of vulnerability (Number of nodes = 10).

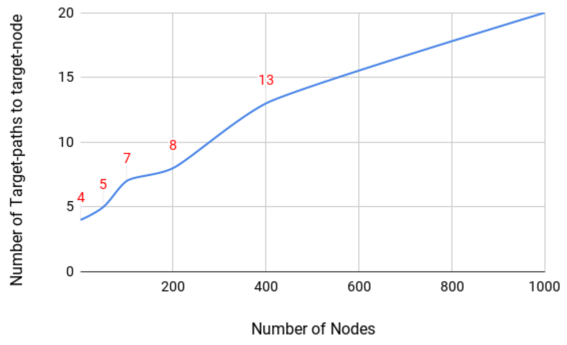

Fig. 6 Total target-paths to the root node generated vs. number of nodes (Number of vulnerabilities on each node = 5).

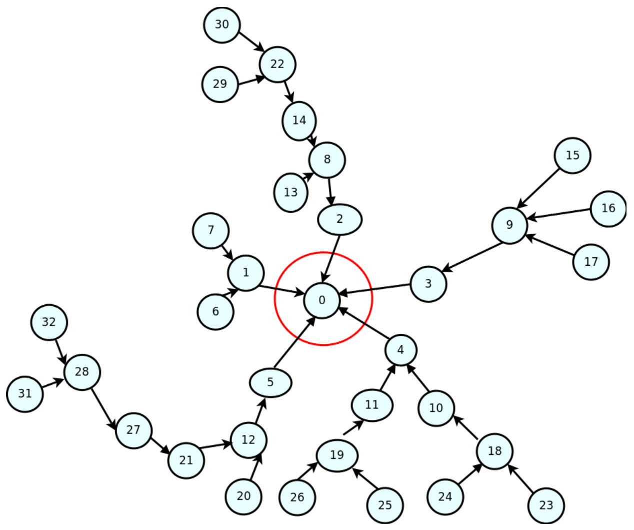

Fig. 7 IoT example network.

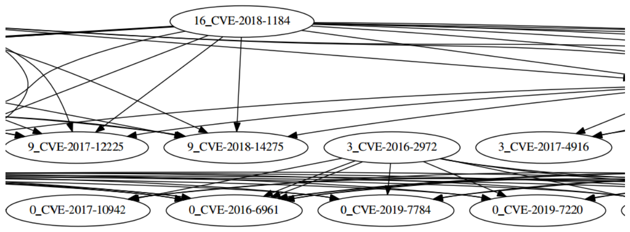

Fig. 8 A part of target graph of IoT example network.

ties on the target node, as shown in **Fig. 6**.

4.1 Finding the Critical Target-path, Critical Node and Critical Vulnerability:

Here, we analyzed the prioritization approaches discussed in Section 3.2, i.e., Node count-based, CVSS based ranking, PatchRank, BC-based approaches. We compared these approaches based on the network (dense/sparse) and vulnerabilities score (same/different). We define the threshold, the state when all attack paths reaching the target node are patched considering Definition 1.

For comparing various security matrices explained in Section 3.2, we used a complex directed graph, as shown in **Fig. 7**. The example tries to show the IoT network complexity, where the end nodes acts as an IoT sensors node, which further sends data to aggregators/gateways. Node 0 acts as a server. For generalization, we consider each node has five vulnerabilities that are randomly assigned from the NVD database. Afterwards, we find all possible Target-paths from twelve end nodes i.e. (6, 7, 15, 16, 17, 23, 24, 25, 26, 30, 31, 32) to node 0 considering the sensors/actuators are the weakest link in the IoT network. A part of the large target graph is shown in **Fig. 8**, where node_instance_id is a unique key for each instance consisting of node_id and vulnerability CVE-ID. An edge from 16_CVE-2018-1184 to 9_CVE-2017-1225 shows that once the attacker gets into node 16, she can enter into node 9 after exploiting CVE-ID CVE-2017-1225. After identifying the possible target-paths to node 1, for a system administrator, choosing a node to be patched efficiently is a challenging task. Even though she has all the information of Target-paths, which can be followed by an attacker to enter into the target system, it is a non-trivial task to proceed next. So, we further extended *IoT-PEN* to finding the critical target-path, the critical node and the critical vulnerability. We compare various security matrices and figure out the applicability of an approach in various scenarios considering all nodes have the vulnerability of the same severity score and with a different severity score.

All the nodes have vulnerabilities of the same severity score: In this scenario, if the attacker has got the required privilege on any leaf node, then depending upon the post-condition, she can break into the target node. We experimented with this scenario several times, in case of low and medium severity case, there were no target paths generated, as the attacker only got access to terminal-nodes and failed to proceed next. However, in the case of high and critical severity scenarios, there were paths from each end node to node 1. In the following scenario, using node count based prioritization, the critical path is from node 32 to node 1, the critical node is node 0 and critical vulnerabilities are a set of vulnerabilities on that node. To reach a secure state, the administrator needs to patch 11 nodes (one optimal solution can be: 0, 1, 4, 8, 9, 12, 18, 19, 22, 27, 28). In CVSS prioritization, the system administrator needs to patch all 33 nodes. In PatchRank, considering that all nodes have the same severity vulnerability, the rank is determined by the weight/importance of the node, the administrator needs to patch nodes (0, 4, 8, 9, 12, 18, 19, 22, 28). However, varying the weights, the result varies in case of the PatchRank. In the case of betweenness centrality, all the target paths pass through target node neighbors. The critical target-path has maximum number of the node. The criticality of

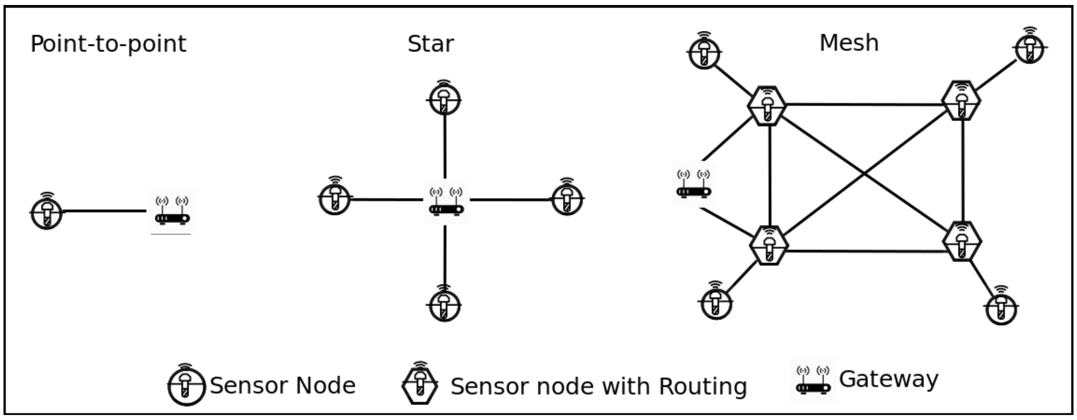

Fig. 9 Various IoT topologies

node is decided by the value of (attack paths passing a particular node/Total number of attack paths). Here, we need to patch node four first.

All the nodes have vulnerabilities of different severity scores: In this scenario, there may be a path from node 32 to node 21, but not from node 21 to target node. So, prioritizing vulnerabilities using betweenness centrality and PatchRank works efficiently by patching the critical vulnerability on critical first, which will drastically decrease the target-path search space. However, in the case of experimenting with this, it completely depends upon the random assignment of vulnerabilities and node weights/importance factor. Therefore, given a static scenario with each node vulnerability and node score, the tool gives the recommendations.

In an IoT network, different types of network topologies e.g., point, star, mesh exist, which uses a client-server architecture as described in **Fig. 9**. In the case of point topology, the attacker can either directly attack the sensor node or can propagate via the gateway. In point topology, PatchRank or CVSS will perform better as exploitability scores of the vulnerability will determine a target path. In the case of a star network topology, all the multi-stage multi-host attacks will involve the gateway always. Therefore, the betweenness centrality is a better security metric for a star topology network. In the case of mesh, each node is connected to another; after reaching the gateway, an attacker can have a multitude of target paths to the target node. Therefore, considering betweenness centrality, it gives an order to patch the system in a cost-efficient way. However, PatchRank will provide a more secure ordering.

It is evident from the above studies that *IoT-PEN* is a scalable solution considering the heterogeneity and scalability of IoT devices. However, *IoT-PEN* fails in case of zero-day vulnerability-as it needs locality, authentication privilege, etc. for prerequisite and post-condition to exploit a particular vulnerability from an NVD vulnerability analysis.

5. Related Work

There have been some research work for penetration testing. A manual penetration approach proposed by Denis et al. [14] performs individual system penetration testing using the Kali Linux for smartphones, blacktooth, etc. PENTOS [15] is a pentest tool specially designed for IoT devices. It does not apply to heterogeneous IoT nodes. Moreover, the framework tests the communication network for a few specific attacks. Xueqiu et al. in Ref. [16] propose an automatic generation algorithm combining the penetration graph generation method with the CVSS information together. However, they did not evaluate their framework in terms of scalability and IoT applicability. Aksu et al. in Ref. [11] focus on the prerequisite and post-condition generation for vulnerabilities by rule-based techniques and machine learning. We modified their rules for our framework. AlGhazo et al. in Ref. [17], proposed a framework which enlists a set of all possible sequences in which atomic-level vulnerabilities can be exploited to compromise certain system-level security given the networked system description. Ou et al. in Ref. [18] presented a Prolog programming language based network security analyzer MulVal. Almost all the works, focus on isolated penetration testing of each system and are manual and not scalable for complex IoT networks. This study extends our previous work in Ref. [19]. In this paper, our focus is on extensive experiments to evaluate the performance of *IoT-PEN* and find the prioritization order of the vulnerabilities. We are using the four security metrics; density-based prioritization, CVSS score based prioritization, betweenness centrality, and PatchRank. We also analyzed the impact of network topologies on IoT-PEN.

6. Conclusion

IoT-PEN is an automatic E2E penetration framework for IoT, a large and complex network. It works well for resource constrained IoT devices, as most of the computation is done on cloud (servers) rather than IoT devices. The framework uses the open-source tool for scanning followed by the vulnerability mapping to CVE-ID. The evaluation of *IoT-PEN* using a complex graph verifies the applicability of *IoT-PEN* to an IoT network. We first pentest individual systems and find the vulnerabilities present in those systems. These vulnerabilities are mapped with the NVD database. Prerequisite and Post-condition w.r.t. that vulnerability are generated from the NVD database. After identifying all possible target-paths to the target node, *IoT-PEN*, recommends the prioritization of the nodes.

Acknowledgments This research is supported by DST-JST project "Security in the IoT space", DST Grant Number

RP03321, JST Grant Number JPMJSC16H3.

References

[1] Li, S., Xu, L.D. and Zhao, S.: 5g internet of things: A survey, *Journal of Industrial Information Integration*, Vol.10, pp.1–9 (2018) (online), available from ⟨http://www.sciencedirect.com/science/article/pii/S2452414X18300037⟩.

[2] Zimba, A., Wang, Z. and Chen, H.: Multi-stage crypto ransomware attacks: A new emerging cyber threat to critical infrastructure and industrial control systems, *ICT Express*, Vol.4, No.1, pp.14–18 (2018).

[3] Angrishi, K.: Turning internet of things(iot) into internet of vulnerabilities (iov): Iot botnets, *CoRR*, Vol.abs/1702.03681 (2017) (online), available from ⟨http://arxiv.org/abs/1702.03681⟩.

[4] Falliere, N., Murchu, L.O. and Chien, E.: W32.stuxnet dossier (2010) (online), available from ⟨https://www.wired.com/images_blogs/threatlevel/2010/11/w32_stuxnet_dossier.pdf⟩ (accessed 2018-11-13).

[5] Yadav, G. and Paul, K.: Assessment of scada system vulnerabilities, *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation* (*ETFA*), pp.1737–1744 (2019).

[6] Sheyner, O. and Wing, J.: Tools for generating and analyzing attack graphs, *Formal Methods for Components and Objects*, de Boer, F.S., Bonsangue, M.M., Graf, S. and de Roever, W.-P. (Eds.), pp.344–371, Springer Berlin Heidelberg (2004).

[7] Mehta, V., Bartzis, C., Zhu, H., Clarke, E. and Wing, J.: Ranking attack graphs, *Recent Advances in Intrusion Detection*, Zamboni, D. and Kruegel, C. (Eds.), pp.127–144, Springer Berlin Heidelberg (2006).

[8] Martin, R.A. and Christey, S.: Vulnerability type distributions in cve (2007) (online), available from ⟨https://cwe.mitre.org/documents/vuln-trends/index.html⟩.

[9] NIST: Common vulnerabilites and exposures (online), available from ⟨https://cve.mitre.org⟩.

[10] NVD: Official common platform enumeration (CPE) dictionary (2007) (online), available from ⟨https://nvd.nist.gov/products/cpe⟩.

[11] Aksu, M.U., Bicakci, K., Dilek, M.H., Ozbayoglu, A.M. and Tatli, E.i.: Automated generation of attack graphs using nvd, *CODASPY '18*, pp.135–142, ACM (online), DOI: http://doi.acm.org/10.1145/3176258.3176339 (2018).

[12] Yadav, G. and Paul, K.: Patchrank: Ordering updates for scada systems, *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation* (*ETFA*), pp.110–117 (Sep. 2019).

[13] Buccafurri, F., Lax, G., Nicolazzo, S., Nocera, A. and Ursino, D.: Measuring betweenness centrality in social internetworking scenarios, *On the Move to Meaningful Internet Systems: OTM 2013 Workshops*, Demey, Y.T. and Panetto, H. (Eds.), pp.666–673, Springer Berlin Heidelberg (2013).

[14] Denis, M., Zena, C. and Hayajneh, T.: Penetration testing: Concepts, attack methods, and defense strategies, *2016 IEEE Long Island Systems, Applications and Technology Conference* (*LISAT*), pp.1–6 (Apr. 2016).

[15] Visoottiviseth, V., Akarasiriwong, P., Chaiyasart, S. and Chotivatunyu, S.: Pentos: Penetration testing tool for internet of thing devices, *TENCON 2017 - 2017 IEEE Region 10 Conference*, pp.2279–2284 (Nov. 2017).

[16] Jia, X.Q., Wang, S., Xia, C. and Lv, L.: Automatic generation algorithm of penetration graph in penetration testing, *2014 9th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp.531–537 (Nov. 2014).

[17] Al Ghazo, A.T., Ibrahim, M., Ren, H. and Kumar, R.: A2g2v: Automated attack graph generator and visualizer, *Mobile IoT SSP'18*, pp.3:1–3:6, ACM (online), DOI: http://doi.acm.org/10.1145/3215466.3215468 (2018).

[18] Ou, X., Boyer, W.F. and McQueen, M.A.: A scalable approach to attack graph generation, *CCS '06*, pp.336–345, ACM (2006).

[19] Yadav, G., Paul, K., Allakany, A. and Okamura, K.: Iot-pen: A penetration testing framework for iot, *2020 International Conference on Information Networking* (*ICOIN*), pp.196–201 (2020).

Geeta Yadav is a Ph.D. scholar in Amar Nath & Shashi Khosla, School of Information Technology, Indian Institute of Technology (IIT) Delhi, India. Her research interests are mainly SCADA system security, IoT Security. Before joining IIT, she worked as a software engineer in Samsung, India and Samsung, Korea.

Kolin Paul is a professor in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Delhi, India. His research interests focus on the use of (embedded) systems in the design of affordable health care and the security of cyber physical systems. He received a Ph.D. in computer science and engineering. He serves as the Microsoft Chair Professor and leads hardware security and reconfigurable efforts at IIT Delhi.

Alaa Allakany is a department member in Faculty of Computers and Information, Kafrelsheikh University, Egypt, He recieved Ph.D. degree from Graduate school of Information Science and Electrical Engineering, Kyushu University, Japan. His rsearch interest are The Future Internet Technologies, IoT Security.

Koji Okamura received his B.S., M.S. and Ph.D. from Kyushu University in 1988, 1990 and 1998, respectively. He became an associate professor of the Computer Center and Graduate School of Information Science and Electrical Engineering in 1998 and a professor at Kyushu University in 2011. He serves as the director of the Cybersecurity Center at Kyushu University and vice director of the Research Institute for Information Technology, and vice CISO of Kyushu University. He is a member of IPSJ, IEICE, IEEE-CSt.