Research Article

International Journal of
Distributed
Sensor Networks

International Journal of Distributed Sensor Networks
2016, Vol. 12(8)
© The Author(s) 2016
DOI: 10.1177/1550147716662947
ijdsn.sagepub.com
SAGE

Using malware for software-defined networking–based smart home security management through a taint checking approach

Ping Wang[1], Kuo-Ming Chao[2], Chi-Chun Lo[3], Wen-Hui Lin[1], Hsiao-Chung Lin[1] and Wun-Jie Chao[1]

Abstract

Numerous security concerns exist in smart home systems in which Internet of Things devices are connected through a home network to enable control using a centralised gateway with a handset device from the Internet. Safeguarding personal information privacy is an increasing concern in smart living services. To guarantee the mobile security of smart living services, security managers use taint checking approaches with dynamic taint propagation analysis operations to examine how a software-defined networking app uses sensitive information and investigate suspicious security vulnerabilities of devices and the effects of the spread of taint propagation over the Internet by identifying taint paths. For solving the dynamic taint propagation analysis problem, most approaches focus on cloud computing applications (apps) with malware threat analysis that involves program vulnerability analyses, rather than on the risk posed by suspicious apps connected to the cloud computing server. Accordingly, this article proposes a taint propagation analysis model incorporating a weighted spanning tree analysis scheme for tracking data with taint marking using several taint checking tools with an open software-defined networking architecture for solving the dynamic taint propagation analysis problem. In the proposed model, Android programs perform dynamic taint propagation to analyse the spread of risks posed by suspicious apps connected to the centralised gateway in a smart home system. In probabilistic risk analysis, risk and defence capability are used for each taint path to assist a defender in recognising the attack results against network threats caused by malware infection and to estimate the losses of associated taint sources. A case of threat analysis of a typical cyber security attack is presented to demonstrate the proposed approach. A new approach was used for verifying the details of an attack sequence for malware infection by incorporating a finite state machine to appropriately represent the real dynamic taint propagation analysis situations at various configuration settings and safeguard deployment. The experimental results proved that the threat analysis model enables a defender to convert the spread of taint propagation to loss and estimate the risk of a specific threat using behavioural analysis associated with 60 families of real malware. Consequently, our scheme was significantly effective in predicting the risk and loss of tainted data propagation for security concerns in smart home systems when the number of taint paths associated with the propagation rules discovered through taint analysis was increased.

Keywords

Mobile security, taint checking, software-defined networking, dynamic taint propagation, smart home system

Date received: 27 February 2016; accepted: 28 June 2016

Academic Editor: Suat Ozdemir

[1]Department of Information Management, Kun Shan University, Tainan, Taiwan
[2]DSM Research Group, Faculty of Engineering and Computing, Coventry University, Coventry, UK
[3]Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan

Corresponding author:
Ping Wang, Department of Information Management, Kun Shan University, Tainan 710-03, Taiwan.
Email: pingwang@mail.ksu.edu.tw

Introduction

Cloud computing uses the Internet to deliver information services to open networks and involves the

Table 1. Comparison of TC approaches for cloud app security.

Issue	Goal	
	Client site	Server site
Targeted at	Privacy of mobile devices	Commercial information of enterprise servers
Solving problem	Stain checking for the privacy leakage of mobile devices	Examine the stain propagation and leakage problems caused by the invasion of both the server and business repository in virtual machines
Information security needs	Find vulnerable apps containing security holes that are not exploitable	Evaluate the content of transmission messages that may be modified or stolen in cloud servers, which can compromise user privacy
	Assess the spread of taint propagation with loss for leaking personal privacy	Accurately assess taint analysis with time constraints

deployment of large-scale platforms; therefore, commercial data on the clouds might become targets of network attacks. For example, in September 2014, celebrities' private photographs stored in the iCloud were disclosed. To prevent such a hack, a security analysis of the information exchange within intersuspicious modules (intramodules) and program of an external network (interapp), such as illegal access memory, buffer overflow attacks of the intramodule and malware downloaded by the botnets of the interapp, are required. In practice, new malware attacks can bypass firewall-based detection by bypassing stack protection and using Hypertext Transfer Protocol logging, kernel hacks and library hack techniques and to the smart living appliances (apps). Effective security defence mechanisms involving threat analysis techniques are essential for detecting intruder attacks in open networks. Taint checking (TC) is a blacklisting approach because it asserts that certain values are dangerous for taint analysis. Generally, TC involves evaluating specific security risks from attacks such as memory corruption and a buffer overflow attack.[1] Unlike the traditional threat analysis technique, in TC, the focus is more on detecting exploits, and data flow analysis is used to determine whether the value is derived from a user input. In taint analysis, modules connected to data originating from untrusted network channels are marked as contaminated, and a series of arithmetic and logic operations to track data with taint marking.

To guarantee the security of cloud computing systems, network administrators use TC approaches with an open software-defined networking (SDN) architecture to examine how a cloud app uses sensitive information and investigate suspicious behaviours of users before deploying an app. An SDN architecture may facilitate network-related security applications because of the controller's central view of the network, and its capacity to track data with symbolic taint marking and identify the spread of intents. In performing malware threat analysis against unspecified malware attacks, network administrators can use a TC approach for tracking information flows between attack sources (malware) and detect vulnerabilities of targeted network apps.

Many TC approaches have been developed to enhance security by preventing malicious users from executing commands on a host computer,[2–5] which require computing emulations with a set of distinct mission hosts in a virtual security experimental environment. As described previously, most existing TC schemes focus on examining the security hole for mobile device apps but do not fully address the severe issues regarding the risk posed by suspicious apps connected to security holes in the cloud computing server. The comparative analyses of TC schemes are provided in Table 1.

The literature contains various proposals for solving the TC problem based on sandbox analysis techniques (SATs). Although SATs have many advantages, they have several limitations. For example, multiple taint sources cannot be traced online. In addition, taint sources can be easily infected through app connections. Furthermore, an SAT does not adequately answer critical questions regarding (1) the security holes of various risk levels in the development of network apps and (2) the attack paths selected for introducing safeguards. By contrast, the TC scheme emphasises specific security risks primarily associated with websites that are attacked using techniques such as SQL injection and buffer overflow attack approaches.[1] Thus, TC techniques effectively specify possible targeted information flows between the attack sources (malware) and the security holes of network apps. Moreover, the technical content of an SAT differs from that of the traditional threat risk approach; that is, an SAT focusses on the detection of the dynamic behaviour between the suspicious host and the test app's vulnerability, whereas the TC technique emphasises examining the scope of taint propagation within an intramodule or interapp. In addition, the connection to taint sources by judging the intents of a contaminated information flow is derived from either the user input or external taint sources for determining the exact signature of exploits (SOEs).

Accordingly, this study proposes a taint propagation analysis model that incorporates a weighted spanning tree analysis for tracking data with taint marking and identifying the spread of intents by considering the state transitions of a program in an SDN-based smart home security management system. Both dynamic taint analysis (DTA) and risk analysis are incorporated in the scheme and used for possible system exploits to solve the dynamic taint propagation analysis (DTPA) problem. In the following context, each taint analysis tool, Valgrind[6] and ComDroid,[7] associated with two malware behavioural analysis tools, Androguard[8] and Droidbox,[9] is used to assist defenders in examining data correct. This analysis helps defenders to analyse the SOEs accounting for the behavioural profiles for assessing the risk of a taint source to the commercial data in the cloud computing server. In developing the proposed model, we considered two crucial aspects: (1) the investigation of the taint paths, propagation rules (PRs) and SOEs for a taint source by tracking data with symbolic taint marking and (2) the evaluation of risk in accordance with the taint paths responding to specific attacks.

The remainder of this article is organised as follows. Section 'Related work' reviews previous studies in this field. Section 'Dynamic taint propagation model for threat risk analysis of mobile malware' introduces the proposed taint propagation analysis model based on the behavioural profile of suspicious apps. An analysis of the results is presented in section 'Cyber security app'. The taint risk in the DTPA process is discussed in section 'Discussion'. Finally, section 'Conclusion' provides the conclusions of this study.

Related work

This section describes the use of two important parts to solve the DTPA problem in open networks, namely, TC approaches for assessing threats and security using SDN architecture.

TC approaches for assessing malware threats

Typically, the following two basic approaches are used for TC: (1) static taint analysis (STA): examine the program text and analyse overmultiple paths of a program. Typically, STA is performed on a control flow graph in which statements are nodes, and an edge exists between nodes if there is a possible transfer of control. (2) DTA: investigate the instructions executed when the program runs. In particular, DTA inspects a single path once of a single run and determines exact taint values.[10] By contrast, the STA might generate extra taint paths from control flow graph and result in false positives.

Normally, three crucial steps are followed in DTA: (1) taint source analyses, (2) propagation analysis with taint marking and (3) testing of the code with the assertions. Since the 2000s, TC approaches have been used for exploring the vulnerabilities in information systems and identifying possible impacts from malware attacks by altering the flow of program execution. In comparison to STA, DTA tracks intents with taint marking to the information flow of the intramodule or interapp; consequently, it is relatively accurate for monitoring the running app and is suitable for tracking the flows of private sensitive data.[11]

Many TC approaches incorporate taint propagation algorithms for detecting malware. TC techniques for an analysis algorithm, including Argos,[2] Panorama,[3] TaintCheck,[4] ComDroid[7] and TaintDroid[12] approaches, have been used to increase the analysis precision by providing insights into how activities and outputs are formulated through the transmission of intents from tainted sources. These TC schemes are summarised in Table 2.

Security management using an SDN architecture

SDN began after Sun Microsystems released Java in 1995. SDN enables network administrators to manage network services through the abstraction of high-level functionality. SDN is a dynamic, manageable, cost-effective and adaptable architecture suitable for the high-bandwidth, dynamic nature of today's applications. SDN architectures decouple network control and data forwarding functions, enabling network control to be directly programmable and the underlying infrastructure to be abstracted from applications and network services.[15]

In smart living appliances, Internet of Things (IoT) devices are connected through a home network to enable control using a centralised network controller associated with remote network devices from the Internet. When integrated by wireless sensing and information communication technologies for IoT devices, home systems and appliances are advantageous because they can communicate in an efficient manner that provides living convenience, health promotion and safety benefits. Because personal information in smart living services cannot be disclosed without legal permission, using private information, including personal IDs, locations, biometrics and secret data, requires ensuring its security.

Theoretically, an SDN-based network architecture provides a user with secure remote monitoring and managed smart living appliances for the home environment. Examples include remote control lighting

Table 2. Security analysis approaches for TC.

Reference	Features	Suitable for	Limitations
Newsome and Song[4]	Developed the TaintCheck tool to capture a system-wide information flow for malware detection, including most types of exploits	TaintCheck produces low false positives for all suspicious programs because the static stain analysis performs according to all possible control flows	May generate too many control flows connected to security holes, often resulting in high computational costs for complex system modelling
Portokalidis et al.[2]	Argos is a full and secure system emulator based on Qemu, which uses dynamic translation to achieve a satisfactory emulation speed	Uses DTA to track data throughout execution and detects any attempts to use them illegally	Argos provides immediate network attack alerts through program execution, which requires a considerable amount of time
Yin et al.[3]	Implemented a TC tool, Panorama, to detect and analyse malware and offer valuable assistance to code analysts and malware scientists	Suitable for quickly comprehending the behaviour of an unknown sample malware by capturing fundamental traits	Malware must be analysed manually; therefore, it is time-consuming
Kim et al.[13]	Developed a DTA tool to track multiple mixed information flows among several processes across a distributed enterprise	Examines the taint propagation scope of information flows within the intramodule or interapp to taint sources	Sensitive to the information structure of an organisation, which heavily depends on the customisation of tool usage
McClurg et al.[10]	Proposed an integrated system involving DTA to mark certain information sources as tainted and keep track of the tainted data propagating in the system	Detects privacy leaks with a relatively minimal overhead, assisting defenders in determining exact taint values with a reduced computational overhead	Produces extremely large suspicious paths to analyse interapps and outputs from taint sources such as user inputs and reads from network connections or definite memory locations and API calls
Chin et al.[7]	Proposed an STA tool, ComDroid, to examine Android app interaction and identify security risks in app components	ComDroid can describe the disassembled output from Dedexer and log potential app component outputs and intent vulnerabilities to perform the dynamic symbolic execution of programs	The authors tracked the intent control flow across functions but did not distinguish between paths through if and switch statements. Instead, they followed all branches. This can lead to false-negatives
Enck et al.[12]	Built a real-time monitoring system called TaintDroid to track how apps use sensitive information on Android platforms at a low level	Precisely analyses how private information is obtained and released by apps downloaded on consumer phones	Requires integrating parts of TaintDroid's API calls into an app. Consequently, implementing TaintDroid as a stand-alone app is not possible
Rastogi et al.[14]	Developed a security analysis of the app AppsPlayground for an automated dynamic security analysis of Android apps	AppsPlayground is suitable for automatically detecting privacy leaks and malicious functionalities in apps	For automatic exploration, many events are simulated; this requires substantial time and is sometimes deficient because user interfaces for apps have complex requirements

TC: taint checking; API: application programming interface.

systems, fire warning, the remote monitoring of infant activity, pet status remote monitoring and the transmitting of personal health information to a cloud platform for analysis. Although mobile devices and IoT devices with an SDN architecture for smart living appliances have improved the convenience of our daily lives, they also pose a threat to personal privacy and information security for users. Moreover, networking manufacturers generally collect personal information and private habits without warning. Consequently, mobile devices or IoT devices are controlled by remote malicious users and cause the improper disclosure or sharing of information. In practice, devices in smart homes provide network-layer security and privacy control mechanisms to ensure the privacy and information security of users by monitoring network activity and detecting suspicious network behaviour. Accordingly, a dynamic security mechanism based on network-layer security and privacy control mechanisms is adopted in the SDN controller (Figure 1).

In addition to existing security mechanisms, novel approaches using SDN as a network-wide control mechanism for resolving high-risk security concerns, including distributed denial of service detection and

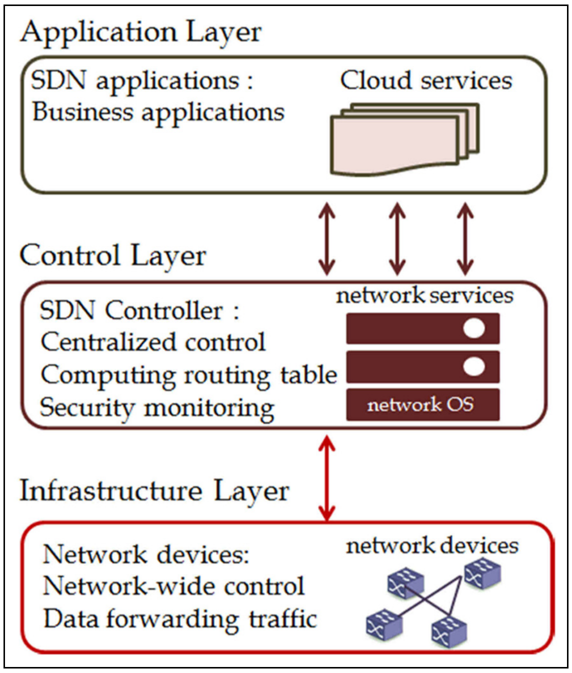

Application Layer
SDN applications : Business applications
Cloud services
Control Layer
SDN Controller : Centralized control Computing routing table Security monitoring
network services
network OS
Infrastructure Layer
Network devices: Network-wide control Data forwarding traffic
network devices

Figure 1. SDN-based network architecture.
Source: Open Networking Foundation.[15]

mitigation,[16,17] worm propagation[18] and botnet protection,[19] have been suggested by several researchers.

Dynamic taint propagation model for threat risk analysis of mobile malware

The proposed model was designed to examine information flows associated with suspicious taint sources, use a taint graph to track the suspicious connection with the taint source and identify the spread of tainted data.

Basic concept

Suppose there are taint sources and a taint sink of a targeted network (i.e. an SDN-based smart home system) in which hackers attempt to compromise network security. For solving the DTPA problem, our approach focusses on developing apps using malware threat analysis that involves program vulnerability analyses for identifying both security holes and malicious behaviours associated with malware attacks.

The first task is initialising and marking all modules for cloud services as a clean state in a targeted network. The basic concept involves using the weighted spanning tree algorithm (WSTA) and determining the cut sets for DTPA on the basis of taint marking; that is, the information type is assigned to tainted data. A taint graph was constructed by starting from a taint source to any connected module through intent transmission. A depth-first search (DFS) was then performed on a taint graph in accordance with taint marking to gradually connect the edges between the two adjacent modules, which pass tainted data (i.e. intents) through the network. In such a situation, the module is marked as contaminated, and the module is marked as clean if no tainted data transmit; that is, the edges between the two adjacent modules are disconnected (dotted lines in Figure 2).

At the end, after visiting all modules, the traversal path of taint propagation was constructed using the DFS spanning tree algorithm. The taint graph represents how tainted data or messages propagate during program execution and is depicted as a directed graph G = (V, E), wherein a set of vertices $V = \{u, v_1, v_2, \ldots, v_m\}$. u represents a taint source; $v_1, v_2, \ldots, v_s$ represent a set of modules in an app or a cloud service; $E = \{e_1, e_2, \ldots, e_n\}$ represent a set of edges between two neighbouring modules of G (Figure 2).

As shown in Figure 2, existing TC approaches do not consider the state transition of an app module in a system, including preconditions and postconditions. In other words, although the receiver module accepted intents, defenders used security protection, such as bug fixes in the operating system and a firewall in the network system, to repel the taint propagation results. However, if defenders do not patch the security vulnerabilities, tainted data would spread. Thus, this study incorporated a finite state machine (FSM) to appropriately represent DTPA using a network flow analysis technique (Figure 3). Figure 3 shows that the FSM may generate uncontaminated states for a module in a taint graph in defence situations and contaminated states for the same module shown in Figure 4. Using an FSM-based taint propagation of information analysis produced various results that represent the actual situations in which various configuration settings, security fixes and safeguard deployments were set up in the DTPA processes.

Figures 3 and 4 show that considering the state transition of a module for taint analysis is more suitable than those in existing TC approaches. As shown in Figure 4, if tainted data are received from an untrusted source for a module that accepts tainted data and output intents to the connected modules, it would be marked as a contaminated state. The module connects the edges between these modules. By contrast, a module is marked as uncontaminated if it disconnects the edges between two modules.

This study proposes an SDN-based smart home system with an OpenFlow protocol for network configuration, traffic management and security monitoring (Figure 5), in which an OpenFlow switch (network device) and a controller communicate via the OpenFlow protocol. In the proposed SDN system, the SDN architecture enables the network controller to track data with taint marking and determine the path

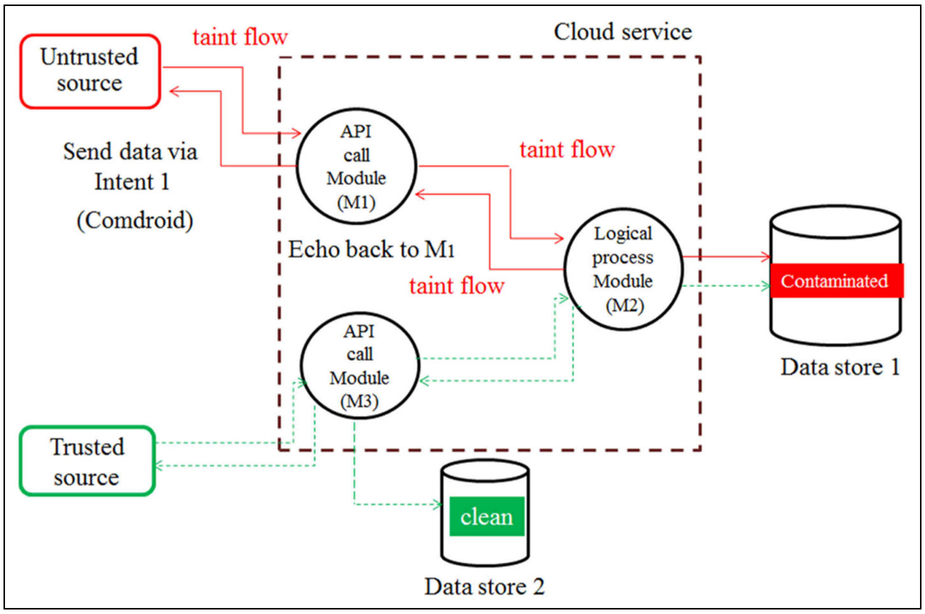


Figure 2. Use of taint analysis to determine whether the module is contaminated via intents.

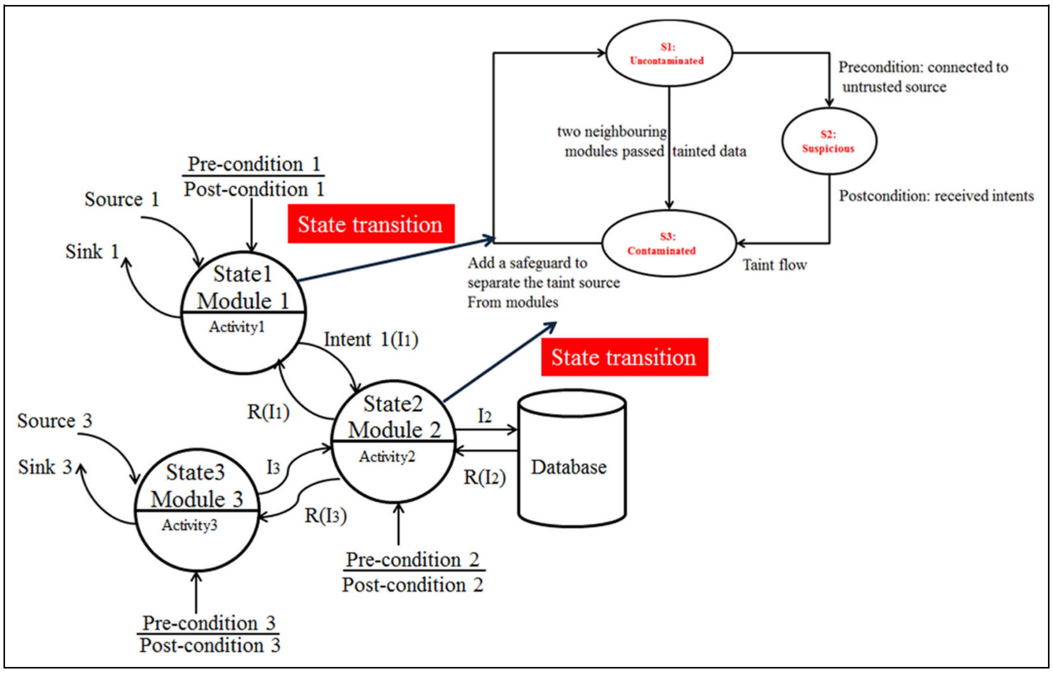

S1: Uncontaminated
Precondition: connected to untrusted source
S2: Suspicious
two neighbouring modules passed tainted data
Postcondition: received intents
S3: Contaminated
Taint flow
Add a safeguard to separate the taint source From modules
State transition
Pre-condition 1
Post-condition 1
Source 1
Sink 1
State1 Module 1 Activity1
Intent 1(I1)
State transition
R(I1)
State2 Module 2 Activity2
I2
R(I2)
Database
Source 3
Sink 3
State3 Module 3 Activity3
I3
R(I3)
Pre-condition 2
Post-condition 2
Pre-condition 3
Post-condition 3

Figure 3. Incorporation of an FSM into taint analysis.

of network packets across a set of networks of switches with two basic components: (1) a control tool (controller) for determining the network packet flow and (2) the OpenFlow routing table (Flow Table) for selecting the network packet transmission path. Thus, OpenFlow can enable the remote administration of packet forwarding tables for layer 3 switch by adding, modifying and removing packet matching rules and actions.[20]

As Figure 5 illustrates, two management tools for OpenFlow are incorporated into the smart home system to detect suspicious behaviour: (1) Open vSwitch, which is used to build a network device for data forwarding and packet routing management in the SDN infrastructure layer, and (2) Floodlight, which is an open-source tool used as a supervisor to manage the SDN control layer and application layer for access control and traffic and security monitoring.

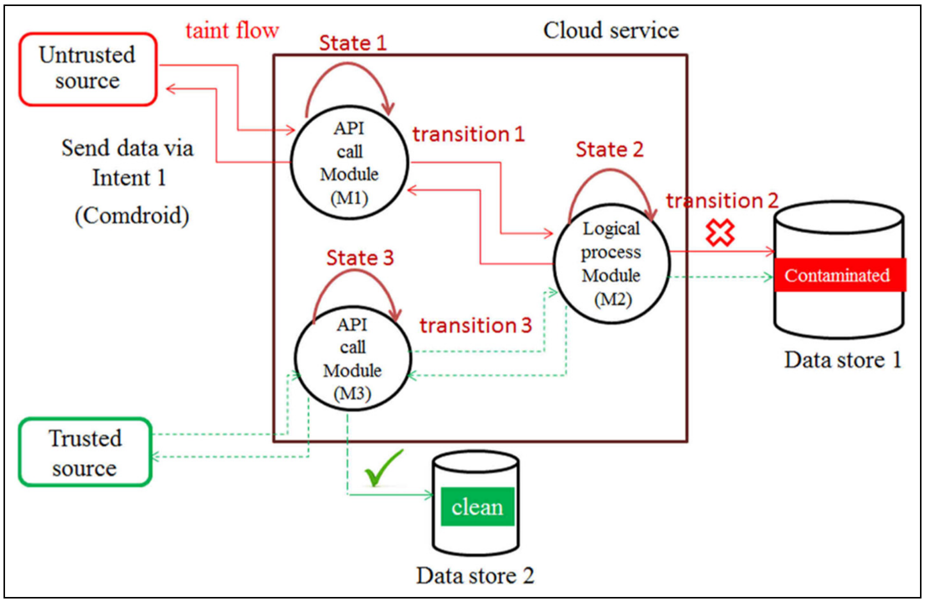


Figure 4. Taint analysis by considering the state transition of a module.

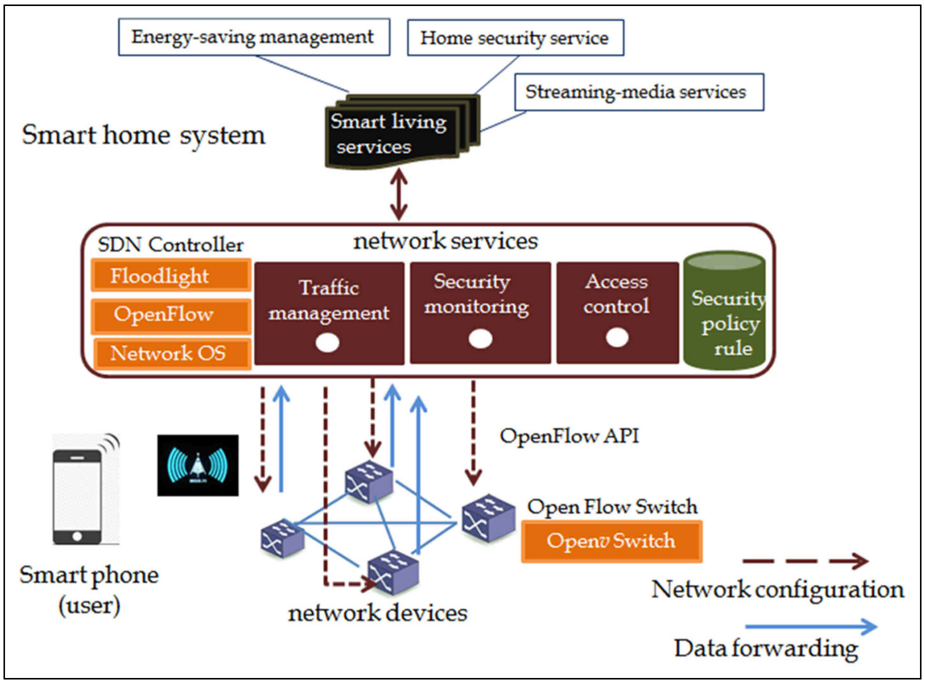


Figure 5. SDN-based smart home system (OpenFlow enabled).

Penetration testing with malware through a trace of intents is an effective approach that facilitates evaluating the security of SDN infrastructures to safely identify vulnerabilities of systems. In practice, sending intents to the improper app modules can lead to user privacy leakage and permissions transferred among apps. Therefore, a taint marking–based framework for tracking intents is proposed and described in the following.

Constructing a taint graph involves large amounts of intent transmission from taint sources to taint sinks and often requires numerous logic operations and cost analyses. Therefore, the WSTA associated with an open SDN controller was used to assist defenders in tracking data with taint marking for DTPA (Figure 6). In addition, the mechanism was developed to analyse the spread of taint propagation in the information flow between

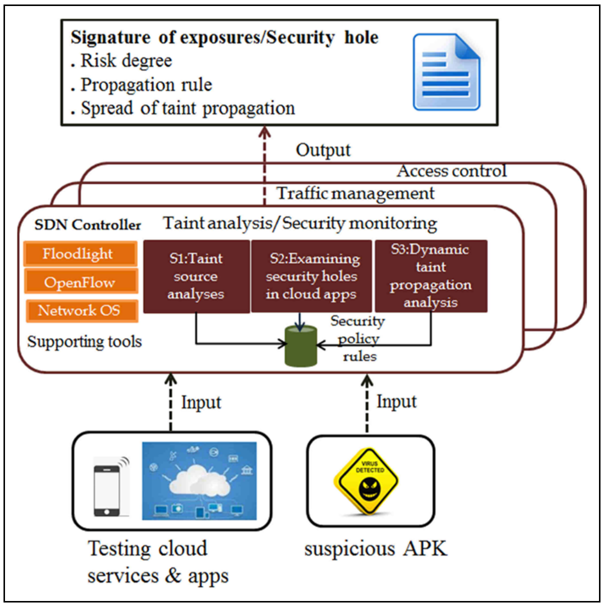

Signature of exposures/Security hole
. Risk degree
. Propagation rule
. Spread of taint propagation
Output
Access control
Traffic management
SDN Controller
Taint analysis/Security monitoring
Floodlight
OpenFlow
Network OS
Supporting tools
S1:Taint source analyses
S2:Examining security holes in cloud apps
S3:Dynamic taint propagation analysis
Security policy rules
Input
Input
Testing cloud services & apps
suspicious APK

Figure 6. Framework for DTPA with an open SDN controller.

mobile devices and cloud computing servers and to help defenders to define the SOEs for taint sources.

Figure 7 shows that the malware behavioural analysis incorporated in the framework is suitable for taint source detection and propagation analysis using application programming interface (API) hooking and taint marking to identify the taint paths of the propagation spread. Three subprocesses included in the DTPA process are taint source analysis, taint propagation analysis and determination of SOEs.

TC process of malware threat analysis

Assume that partial system vulnerabilities of a cloud service are known and a set of behavioural profiles associated with each malware has been identified. The following DTPA algorithm includes a new resolution process of taint propagation to track the taint paths of propagation according to the behavioural profiles of a specific threat, thereby estimating the SOEs of a successful attack by malware.

Step 1: taint source analyses

Step 1.1: malware behavioural analyses. To perform behavioural analysis of malware, a common approach is to use virtual machine analysis techniques. Generally, a sandbox provides capabilities such as stopping at a control point to prevent potentially dangerous programs from running, as well as monitoring and recording activity in a closed environment when the malware is running. Malware signatures were derived by combining both the security flaws of static code analysis and dynamic behavioural patterns of behavioural analysis with the support of a virtual machine emulator to facilitate the detection of unidentified mobile malware and variants.

Dynamic behavioural analysis was conducted to obtain the major runtime behaviour regarding access to the network, file, registry and disc of each app, which was compared with malicious and normal behavioural profiles. Static analysis was conducted to examine the specific source codes and binary strings of malware, and comparison results were recorded in a log file to discriminate the differences in the signatures of malware variants and new viruses. In the malware behavioural analyses, two free shareware products, Androguard[8] and DroidBox,[9] developed by the Honeynet project, were used to generate malware signatures as the source data set; these signatures were used to taint analyses later. An experiment involving the following four steps was conducted:

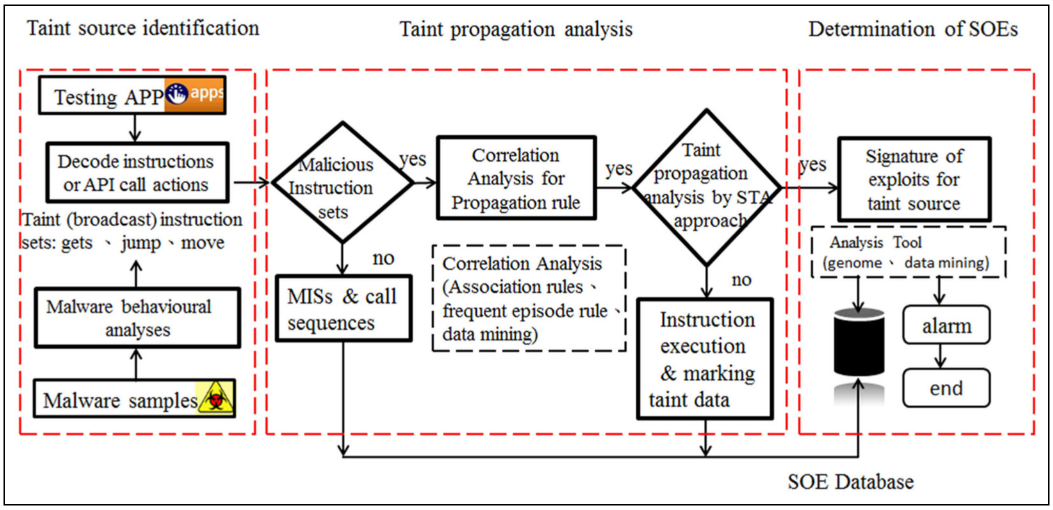

Taint source identification
Taint propagation analysis
Determination of SOEs
Testing APP apps
Decode instructions or API call actions
Taint (broadcast) instruction sets: gets 、 jump 、 move
Malware behavioural analyses
Malware samples
Malicious Instruction sets
yes
no
MISs & call sequences
Correlation Analysis for Propagation rule
Correlation Analysis (Association rules 、 frequent episode rule 、 data mining)
yes
Taint propagation analysis by STA approach
no
Instruction execution & marking taint data
yes
Signature of exploits for taint source
Analysis Tool (genome 、 data mining)
alarm
end
SOE Database

Figure 7. Execution process of the dynamic taint propagation analysis shown in Figure 6.

1. Download the suspected apps (.apk file format) from a mobile phone;
2. Perform in-depth behavioural analysis on the DroidBox platform;
3. Perform code analysis on the Androguard platform using reverse engineering;
4. Output the synthesis report.

Step 1.2: malicious activity analyses of taint sources. This step involves the violation detection for the permission rules as the taint source analyses. Four protection levels exist for the access to the system API with permissions on Android platforms: Normal, Dangerous, Signature and SignatureOrSystem. Apps request the appropriate permissions in their manifests to obtain a privileged access to system commands or protected API calls. To analyse the behaviours of the taint sources, the commands of program executables and API calls outputs connected modules to be listed as the basis of malicious instruction sets (MISs).

To identify the attack profiles of MISs, intent-based analyses are adopted to inspect the transmission of intents from taint sources to the receiving module. Intents can be sent between three of four components: Activities, Services, Receivers and Broadcast. Intents can be used to start Activities; start, stop and bind Services; and broadcast information to Broadcast Receivers.[7] Generally, intent operations include three types of instruction sets: (1) source: MISs for taint source analysis exchange parameters between the taint sources and sinks, such as startActivity, startService, stopService, bindService, read and gets; (2) sink: the instruction sets for a taint sink include receive, onReceive and write; and (3) propagation: commands for taint propagation comprise sendBroadcast and sendOrderedBroadcast. Once the MISs are located, intent-based analysis can be used to construct the malicious behavioural profiles for taint sources.

Step 2: examining security holes in cloud apps. This step identifies security holes using STA that involves program vulnerability analyses for developing apps. Theoretically, more security holes in apps cause malicious behaviours.

Available TC schemes for examining security holes using analysis mechanisms, such as Panorama,[3] Valgrind,[6] TEMU[21] and TAJ,[22] are capable of inspecting security holes in apps. Valgrind,[6] developed by Seward, was proposed for automatic multimemory debugging and detailed memory leak detection. Valgrind was originally designed to be a free memory debugging tool for a Linux operating environment, and it has evolved into a generic framework for creating dynamic analysis tools such as checkers and profilers for inspecting programs being developed. In particular, Valgrind can recompile a binary code to run on the host and target of the same architecture using a simulated CPU technique. Therefore, it is suitable for supporting the inspection of data at possible exploits through the examination of program control transfers.

Valgrind contains multiple tools. The most frequently used tool is Memcheck. Valgrind is a virtual machine using just-in-time compilation techniques, including dynamic recompilation. In STA, six types of test cases are examined using Memcheck: (1) invalid write, (2) conditioned jump on an uninitialised value, (3) memory boundary check for invalid read and write, (4) source and destination overlap, (5) inconsistencies between dynamic memory allocation and release and (6) memory leak. The evaluation of risk based on the security holes responding to a specific threat is described in the following.

Step 3: DTPA. This step helps defenders to examine whether a testing app is vulnerable to potential cyber attacks by comparing normal and malicious behavioural profiles (i.e. permissions, PRs and SOEs) between a legitimate app and a malware website. In DTPA, PRs and SOEs for taint sources are characterised using three subprocesses: (1) correlation analysis (CA) for MISs, (2) taint propagation analysis using the WSTA and (3) the determination of SOEs.

Step 3.1: correlation analysis for MISs. To identify the spread of taint propagation, PRs must be defined by tracking the information flow. Theoretically, PRs can be derived using CA for app components and relevant tainted data via the execution of MISs to analyse the stain propagation behaviour. Furthermore, many CA approaches incorporate association rules, frequent episode rules and data mining approaches to facilitate precise prediction for generating PRs.

PRs for tainted data can be analysed by comparing the distinct calling sequences of the malicious API calls in which the calling sequences are establishing using a pair of [Source_j, Sink_k Sink_{k+1}, …] between the malware website and the proper app. In practice, network analysis tools, such as Netflow and Wireshark, can be used to assist a defender in collecting large amounts of logs to identify the specific calling sequences of MISs. Finally, the relevant calling sequences of MISs based on program execution for each taint source j are summed to generate the PR, which has the form PR_i (Apk name, Taint source, Module and Calling sequence of MISs), as shown in Table 3.

As shown in Table 3, the calling sequence of MISs represents the propagation chain of a taint source i (i = 1,…, m) and comprises a set of methods used in evaluating the attack signature for each module. The module name associated with a specific MIS is

Table 3. Examples of the taint propagation rule.

Apk	Taint source	Module	Calling sequence of MISs
lottery.apk	SubscriberId service url	com.lotsynergy	[Source:SubscriberId,Sink$_1$:Landroid/telephony/TelephonyManager/getSubscriberId] [Source:service,Sink$_1$:Landroid/content/Intent] [Source:url,Sink$_1$:Ljava/net/URL/openConnection]

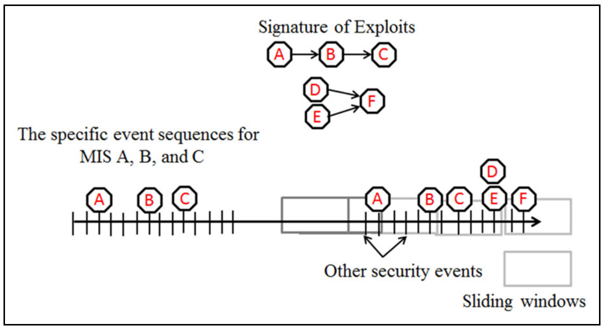


Figure 8. Determine the exact pattern of the taint propagation using sliding windows.

measured using a set of calling sequences, where a longer calling sequence might cause a higher risk and severe loss.

When deciding the calling sequence of MISs (Table 3), determining the exact pattern of the taint propagation actions is difficult. On the basis of the concept of an intrusion detection system, the occurrence of taint propagation is solved using frequent episode rules[23] by accumulating and associating the security logs as follows. Generally, episodes are partially ordered sets of events. The frequent episode rule is used to determine the specific event sequences for appropriately determining the MISs of a taint source, as shown in Figure 8.

Given an event sequence $s = (s;\ T_s;\ T_e)$ and a window width *win*, let the time window of an episode be given by $w = (w;\ T_s;\ T_e)$. The support degree of an episode is defined as the fraction of windows where the episode occurs. Theoretically, given s and *win*, the support degree of an episode (α) (i.e. MISs) for a single taint path j in s is

$$\sup(\alpha) = \frac{|\{\alpha \text{ occurs in } \omega\}|}{|\{W(s, win)\}|} \tag{1}$$

Once sup(α) is obtained, it can be used to predict the probability of an attack occurrence p_{ij} for a single taint path j from taint source i. In other words, sup(α) reveals the connections between attack events with MISs in the given security event sequence. Eventually, the normal behavioural profiles for apps are compared with the calling sequence of MISs for a taint source to determine whether an app is vulnerable to a specific taint source.

Step 3.2: taint propagation analysis using the WSTA. Suppose a spanning tree T models the network flow for DTPA using graph theory. Here, T is an undirected graph G used to construct a tree, including all vertices and edges without any loop.

To evaluate the taint risk of DTPA, the probability of attack success for taint path $p_{ij}(0-1)$ is assessed using equation (1). The defender must determine the normalised value of loss of taint propagation $l_j(0-1)$ and defence capability $d_j(0-1)$ of a taint path by considering the attack–defence scenarios of networks in the following.

According to equation (2), the taint risk matrix was used to describe the probability of attack success for taint source i ($i = 1,\ldots, m$), which was caused by a set of security holes that generated taint paths j ($j = 1,\ldots, n$) by propagating taint and has the form

$$P = [p_{ij}]_{mxn} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1j} & \cdots & p_{1n} \\ p_{21} & p_{22} & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & p_{33} & \vdots & \vdots & \vdots \\ & \cdots & \cdots & p_{ij} & \vdots & \\ \vdots & \cdots & \cdots & \cdots & \cdots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mj} & \cdots & p_{mn} \end{bmatrix} \tag{2}$$

X_i indicates that a taint source caused by a security hole exists in an app or cloud service, and $p_h(X_i)$ indicates that a defender has estimated the detection probability of a security hole for a security test. Suppose a defender performed taint analyses and recorded the possible taint paths of propagation spread to examine the links between the occurring threat event S_j and security holes $P_h(X_i|S_j)$, which is assessed using equation (1). Obviously, $P_h(X_i|S_j)$ is a posterior probability, and $S_j, j = 1, \ldots, n$ means that the threat events were produced by the spread of taint propagation. The probability of attack success caused by security holes and their tainted data, $P_h(S_j|X_i)$, can then be estimated by applying the Bayesian decision rule

$$p_{ij} = P_h(S_j|X_i) = \frac{P_h(X_i|S_j) \times P_h(S_j)}{P_h(X_i)} \tag{3}$$

$P_h(S_j|X_i)$ increases as the probability of both $p_h(X_i)$ and $P_h(S_j)$ is increased. The process for dynamic taint tracking using the WSTA is depicted in the following.

Initially, the WSTA ensures the connection between any two nodes of a network in which there is only a single path without a loop. The modelling of taint propagation based on the connectivity of the information flow is then used to construct a spanning tree algorithm. In particular, a taint graph is constructed by aggregating spanning subtrees, which involve edges with weights. Theoretically, the graph must assign a greater weight to an edge when a high degree of the spread of taint propagation causes a series threat.

A defender assigns a weighted value to each edge at time t by considering the normalised ratio of the loss of data leaks for a single taint path to those for all taint paths for a specific threat in a taint graph G, as follows

$$w_j(t) = \frac{l_j(t)}{\sum_{j=1}^{n} l_j(t)} \tag{4}$$

In practice, the weight of a traversal path for a taint path must be updated by the continual spread of taint propagation along its path at time $t+1$, and this may increase the loss $\Delta l_j(t+1)$. Thus, when reconstructing the possible taint paths, the weight on the taint path is obtained using

$$w_j(t+1) = \frac{l_j(t+1) + \Delta l_j(t+1)}{\sum_{j=1}^{n} l_j(t)} \tag{5}$$

By contrast, the defence investment resources on each path against tainted data pass through that helps a defender to identify the total cost to secure the possible taint paths. If the project time or defence resource is limited, the minimum spanning tree (MST) can be applied to determine the priority of guarding the taint path, thereby preventing maximum leaks to users using minimum resources. After defining the PRs for a taint source, the WSTA is used to identify the spread of taint propagation associated with the development of the TC tool.

Generally, two algorithms for the WSTA can be used: Dijkstra's algorithm and Prim's algorithm. The MST algorithm proposed by Dijkstra is used for identifying the shortest paths between nodes in a graph. It selects the unvisited vertex with the lowest distance, calculates the weights across it to each unvisited neighbour vertex and updates the neighbour's weighting value if it is smaller. The vertex is marked as visited when the neighbour vertex is updated. By contrast, Prim's algorithm is a greedy algorithm that identifies an MST for a weighted graph. It identifies a subset of edges that forms a tree including every vertex, and the total weight of all the edges in the tree is minimised. The algorithm operates by building this tree one vertex at a time from an arbitrary starting vertex and at each step adding a low-cost possible connection from the tree to another vertex. In WSTA analysis, taint risk assessment is performed in the following.

DTPA was used to examine the various vulnerabilities in information systems, identify risks regarding tainted data propagation and resolve high-risk security concerns. The proposed model was designed to describe the SOE and estimate the risk of each taint path for selecting appropriate safeguards to defend against cyber attacks. In DTPA, the taint risk (r_i) is characterised using two quantities referring to the national vulnerability database (NVD) of common vulnerability scoring system (CVSS): (1) the probability of attack success for s single taint path j (p_{ij}) and (2) the loss of taint propagation (l_j) caused by a set of security holes for a taint source i, which generates the possible taint paths j ($j = 1,\dots, n$). The taint risk is obtained using

$$r_i = \sum_{j=1}^{n} p_{ij} \times l_j \tag{6}$$

Step 3.3: determination of SOEs. The recognised security vulnerabilities of network apps have been investigated, examined and reported. For example, Mitre Corporation maintains a list of disclosed vulnerabilities in a system called common vulnerabilities and exposures (CVEs), in which vulnerabilities are scored using a CVSS.

The loss of data leaks may correlate closely to the spread of taint propagation. The SOEs indicate a set of possible attack profiles[24] caused by security holes for a taint source. In this study, the form for SOEs was defined as ($\mathrm{ID_j}$, CVE, taint source url, affected module name and PRs) and was generated by collecting a set of PR, $\mathrm{PR_k}$, $\mathrm{PR_{k+1}}$, $\mathrm{PR_{k+2}}$, ... for each taint source in accordance with the behaviour analysis of malware. Once the PRs and SOEs were identified, they were used to distinguish whether the app was malicious or benign by comparing the differences in the signatures of malware and legitimate apps. A detailed algorithm for tracking data with taint marking for dynamic propagation analysis is described by Program Description Language (PDL) and shown in Figure 9.

Cyber security app

In this section, the applicability of the proposed TC analysis model is demonstrated by considering an example of cloud security using an open SDN-based smart home system (Figure 10). In constructing the SDN infrastructure layer, our experiment incorporated both Raspberry $\mathrm{P_i}$ and a wireless access point (AP)

```
Algorithm DTPA: Dynamic taint propagation analysis
1: Begin;
2: Input a directed graph G = (V, E), V = {$u$, $v_1$, $v_2$,...,$v_m$}, E = {$e_1$, $e_2$,...,$e_n$};
3: $m$ = |V|; $n$ = |E|;
4: If ($m > 1$), then (taint source = TRUE);
5: Initialise an m × n matrix with all '0' elements; $[\ ]_{ij} = 1$ (boolean value indicating that the
     taint source passes the intent ij);
6: Discover a set of taint graph G;
7: Loop {
8: If (taint source), then {
9: Taint_graph ($v$) –performs a DFS by starting with a taint source $u$;
10: {
11: For (each unvisited neighbour $u$ of $v$) {
12: If two neighbouring modules have passed tainted data, then contaminated = TRUE;
13: If (contaminated) then
14: Mark the edge from $u$ to $v$;
15: Calculate the probability of attack success for taint path ($p_{ij}$) by using (1)-(3)
16: Determine a loss to the edge for a single taint path ($l_j$)
17:Assign a weight ($w_j$) to the edge by using (4)-(5) and consider that each path of tainted da-
     ta causes the losses of information leaks;
18: Add the weights for each taint path for spanning subtrees;
19: List the taint paths for spanning subtrees;
20: Assign a defence investment value to the edge;
21: Use the MST to determine the priority of guarding the taint path;
20: Write a list into the PR [$u_i$,$v_{j=1,...,n}$,];
21: Write a list into the SOE$_i$ ($u$, PR$_{j=1,...,n}$[$u_i$,$v_{j=1,...,n}$]);
22:   else
23: Traverse another taint source $u$ to $v$ in G;
24:   Call Taint_graph ($u$);
25:           }--endif;
26:        }--end for;
27:     }--endif;
28: }--end loop;
29: Output the MST and the taint paths of the propagation spread in G;
30: Estimate the risk ($r_i$) for a taint source $i$ by using (6) for selecting appropriate safeguards
     to defend against cyber-attacks.\
31: Return SOE$_i$ ($r_i$, PR$_{j=1,...,n}$[$u_i$,$v_{j=1,...,n}$]);
32: End.
```

Figure 9. Algorithm of DTPA for cloud apps.

with a set of management software tools including OpenWrt, Open vSwitch and Floodlight to construct an SDN-based smart home system, in which Floodlight serves as a supervisor to manage the SDN controller layer and application layer for detecting network anomalies in security monitoring. As Figure 10 illustrates, Open vSwitch assists OpenFlow switches in communicating with each other using the OpenFlow protocol via a secure channel to connect the SDN controller. In the proposed SDN project, the SDN controller uses an Ryu SDN framework in an Ubuntu operation system to determine the network configuration and network packet flow of network procedures, such as access control, network traffic and security monitoring.

The smart home gateway is responsible for connection to external communications on the Internet. SDN applications for smart home scenarios with home security monitoring, lighting control and temperature and humidity monitoring were synchronised with three subnetworks: 192.168.1.0/24, 192.168.2.0/24 and 192.168.3.0/24. Each of these subnetworks communicates via a Wi-Fi base station (wireless AP) with OpenFlow switches (Table 4).

Table 4. Experiment environment.

Role	Hardware	Software	IP address
Smart home gateway	TP-Link 841ND wireless AP	OpenWrt + Open vSwitch	192.168.0.1
SDN controller	A personal computer with Intel Core i5 4210U 4-GB RAM	Ubuntu 14.04.3 + Floodlight	192.168.0.104
OpenFlow switch#1–3[a]	Raspberry Pi 2 model B	Raspbian + Open vSwitch	192.168.0.252–254
Wireless AP#1–3	TP-Link 841ND wireless AP	OpenWrt + Open vSwitch	192.168.1.1, 192.168.2.1, 192.168.3.1

SDN: software-defined networking.
[a]OpenFlow switch adopts an Ethernet-enabled USB interface wireless AP.

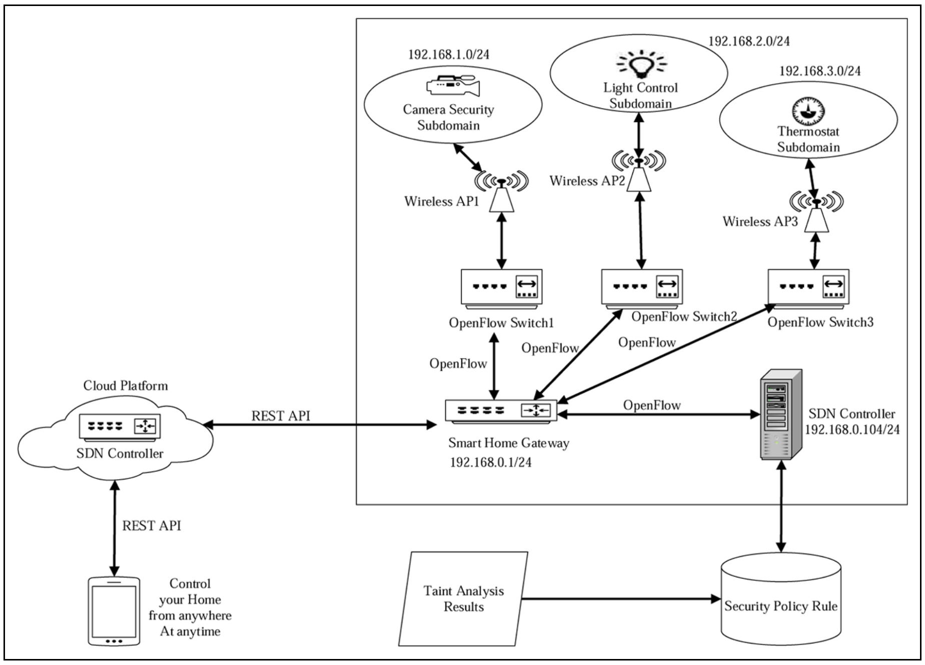


Figure 10. SDN-based smart home security management.

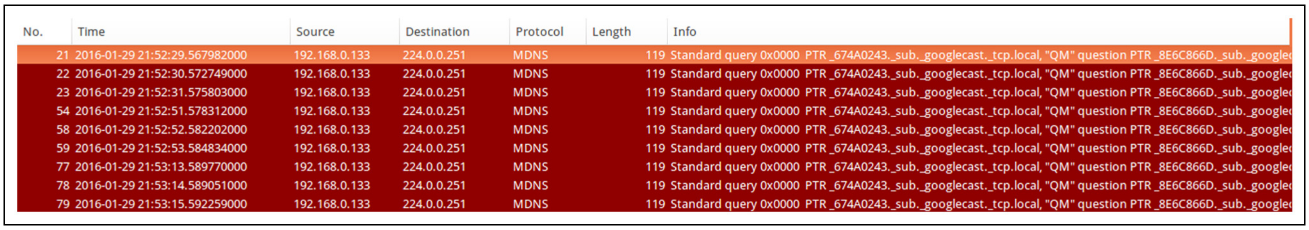
In the proposed SDN system, security monitoring entails periodically collecting network statistics from OpenFlow switches and then applying classification algorithms to those statistics to detect any network anomalies. If an anomaly is detected, the application instructs the controller on how to reprogram the data plane to mitigate it.

In the experiment, the DTPA is constructed using the following three-step procedure involving the taint source analysis, malware behavioural analysis and DTPA on an android platform. For example, malware Zitmo (Zeus-in-the-mobile) Trojan attacks Android and Blackberry smartphones. As shown in Figure 11 and Table 5, Zitmo was found to have a connection to the command-and-control (C&C) server used in a botnet, IP 224.0.0.251 for performing multicast Domain Name System (DNS) queries by examining the traffic information of OpenFlow switch with taint sources.

In the experiment, the DTPA is constructed using the following three-step procedure involving the taint source analysis, malware behavioural analysis and DTPA on an android platform.

Step 1: Taint source analyses

Step 1.1: malware behavioural analyses. A total of 60 malware families identified on mobile devices using active infections were obtained from blacklists published on the Dr Web and Contagio Blogger websites for an experiment conducted from February 2013 to February 2014. Initially, the suspected app (.apk) was downloaded from a smart device to both the



```
No.  Time                           Source          Destination  Protocol  Length  Info
  21 2016-01-29 21:52:29.567982000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
  22 2016-01-29 21:52:30.572749000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
  23 2016-01-29 21:52:31.575803000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
  54 2016-01-29 21:52:51.578312000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
  58 2016-01-29 21:52:52.582202000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
  59 2016-01-29 21:52:53.584834000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
  77 2016-01-29 21:53:13.589770000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
  78 2016-01-29 21:53:14.589051000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
  79 2016-01-29 21:53:15.592259000  192.168.0.133   224.0.0.251  MDNS         119  Standard query 0x0000  PTR _674A0243._sub._googlecast._tcp.local, "QM" question PTR _8E6C866D._sub._google
```

Figure 11. An example of malicious activity analyses of a taint source with OpenFlow switches.

Table 5. Malicious activities for malware Zitmo.

Malware	API call	Infected module
Zitmo	READ_PHONE_STATE (read phone state and identity)	com/security/service/MainActivity
	ACCESS_NETWORK_STATE (view network status)	com/security/service/MainActivity
	INTERNET (full Internet access)	com/security/service/MainActivity
	WAKE_LOCK (prevent phone from sleeping)	com/security/service/MainActivity

API: application programming interface.

DroidBox[9] and the Androguard analysis platform. The experimental results of the behavioural analysis and code analysis of the mobile viruses are shown in Table 6. Defenders can verify the details of an attack sequence to obtain the possible attack profile.

Step 1.2: malicious activity analyses of taint sources. To analyse the intent-based signature for taint source attacks and examine whether an app is vulnerable to special cyber attacks according to four protection levels, install the APP package file Android application package (APK) into the testing folder; execute TC tools, ComDroid, API_Monitor and Wireshark; examine illegal memory access; collect message passing to trace the taint source; inspect API call outputs; investigate taint paths for service, activity and receiver and broadcast communication among modules; and analyse binary interactions with the environment. An example of malicious activity analyses of a taint source is shown in Table 7.

In practice, the N-gram modelling scheme is used to obtain the more occurred API calls for MISs. We randomly examined 30 suspicious apps to examine the capability of vulnerability detection for ComDroid and API_Monitor in proposed SDN system, thereby determining the common weaknesses of the apps. Consequently, ComDroid detected a total of 179 exploits, 49 warnings for exposed components and 353 warnings for exposed intents across 30 suspicious apps. In sum, the most occurred API calls associated with the relevant MISs for the 30 suspicious apps were obtained using analyse intent source, sink and broadcast operations for Input/output (I/O), service, activity and communication, as shown in Table 8.

After defenders identify the calling API for MISs for the taint source, the attack profiles can be accumulated to perform TC analysis with the security holes.

Step 2: examining security holes in apps

In this step, two Valgrind-based analysis tools, Valgrind-3.9.0 for X86/Linux and X86/Android (4.0), were used to perform taint source analyses. Initially, Dedexer was used to disassemble DEX files; the Valgrind tool was then employed to record potential vulnerabilities for modules and intents, and Androguard was then used to confirm the results of Valgrind and to further track information flows between the attack sources (i.e. malware) and the possible tainted data of targeted network apps to reveal the possible taint source, as illustrated in Figure 12.

Essentially, six types of test cases for control variables of the module were examined using Valgrind: (1) invalid write, (2) conditioned jump on an uninitialised value, (3) memory boundary check for invalid read and write, (4) source and destination overlap, (5) inconsistencies between dynamic memory allocation and release and (6) memory leak. These test cases were examined by considering the taint policies of an organisation (Figures 13 and 14). For conducting a security check of network apps, Androguard can perform STA on the major activities (namely, MainActivity) for APPs to reveal the potential taint source.

Typically, two auxiliary tools in Androguard, dex2jar-0.0.9.15 and jd-gui, are used to disassemble the binary files and form the source code in which a folder contains classes.dex, a .APK file, a package and AndroidManifest.xml. The execution of tool./

Table 6. Relationships between viruses and behaviours.

Virus	Behaviour											
	Sent SMS	Information leaks	Authority override	Circumvented permissions	Started service	Broadcast receiver	Operation links	Outbound traffic	Inbound traffic	Encrypted API	File read	File write
SMS Zombie		×	×									
Chuli.A					×	×		×				×
Enesoluty		×			×		×					×
System security		×			×	×	×	×	×			
FakeGuard						×	×					×
Sumzand		×				×	×					
FakeLookout					×	×	×	×				×
FinFish						×						×
Loozfon		×			×		×					×
Kranxpay			×		×	×	×	×	×			×
Dropdialer							×	×	×			×
Gonfu		×	×	×	×		×	×				
FindandCall						×	×	×				×
FakeInst.a				×		×						×
Qicsomos	×		×								×	
Uxipp						×						
Tascudap						×						
Faketoken.b		×			×	×	×	×				×
Scipiex					×		×				×	×
Gen.2					×							×
Fakeguard					×	×	×					×
Startapp.5.origin				×	×	×	×	×		×		
Monad.c						×						×
Andef.b					×	×	×	×			×	

API: application programming interface.

Table 7. Malicious activities for a taint source.

Taint source	Module	Operation	Intent receiving communication			
			Activity sink	Service sink	Broadcast sink	Pending intent sink
DDreamLight com. Beauty.Leg-1.apk	com/Beauty/Leg/lightdd/c	handleMessage	1		1	
	com/Beauty/Leg/lightdd/Receiver	onReceive		1		
	com/Beauty/Leg/lightdd/CoreService					
	com/Beauty/Leg/lightdd/a/	onStart			1	
	com/Beauty/Leg/SexyImages/a	Run		1	1	
	com/Beauty/Leg/SexyImages		1			

Table 8. API calls for MISs with 30 taint sources.

API calls	Number of API calls	Types of instruction sets			
		Activity sink	Service sink	Broadcast sink	Pending intent sink
Ljava/lang/Void	1				1
Ljava/util/List	1		1		
Ljava/lang/Class	1		1		
Landroid/net/Uri	1	1			
Landroid/view/View	1	1			
Landroid/view/MenuItem	1	1			
Landroid/graphics/Bitmap	1			1	
Landroid/content/ServiceConnection	1		1		
Landroid/os/Message	3	3			
Landroid/webkit/WebView	3	3			
Landroid/content/DialogInterface	3	3			
Landroid/app/Activity	5	5			
Landroid/os/Bundle	11	8	2		1
Ljava/lang/String	14	8	1	1	5
Landroid/content/Intent	17	6	7	1	3
Landroid/content/Context	20	4	7	1	8

API: application programming interface; MIS: malicious instruction set.

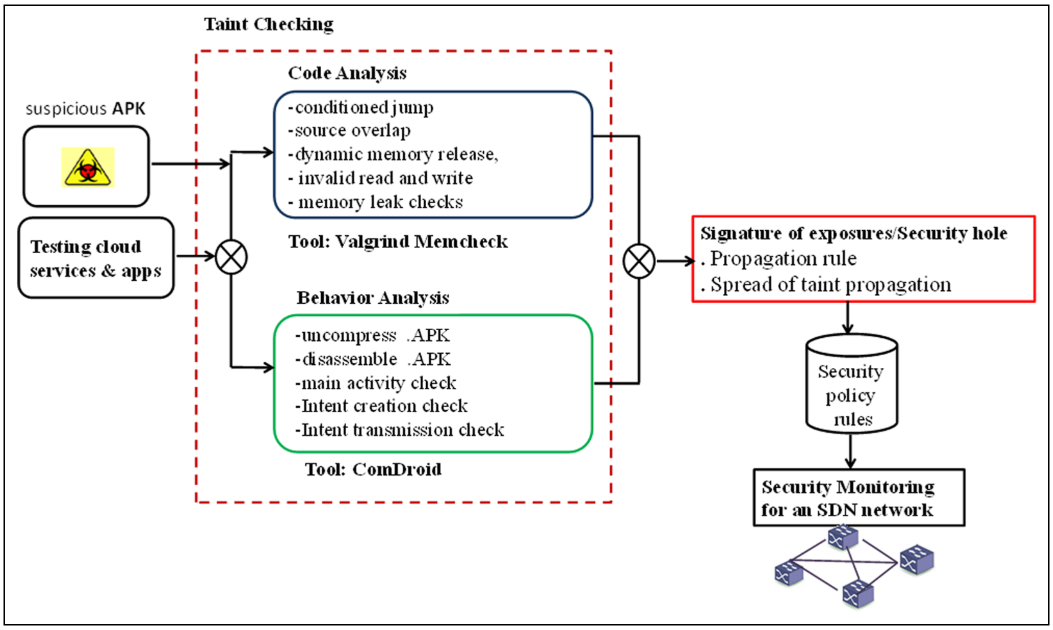
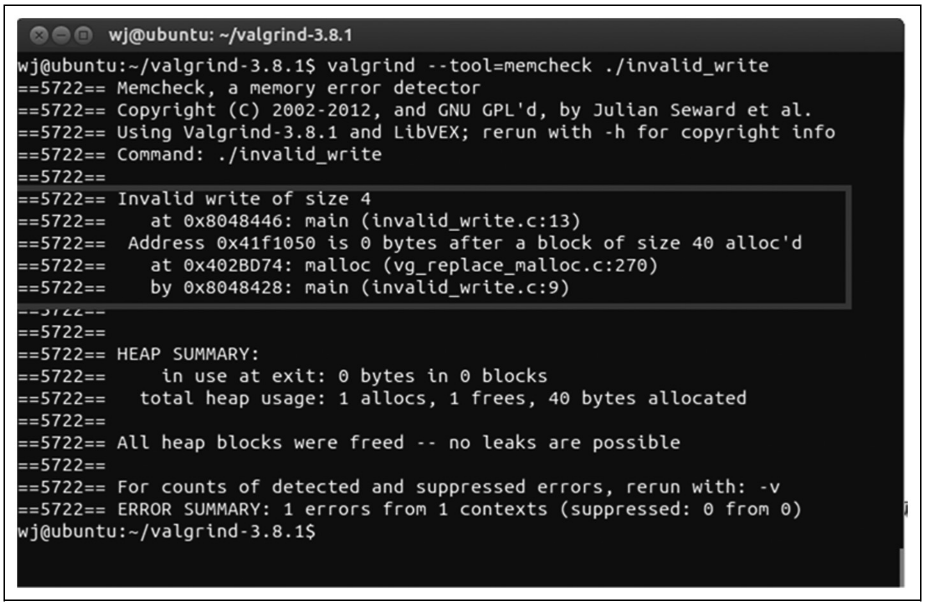
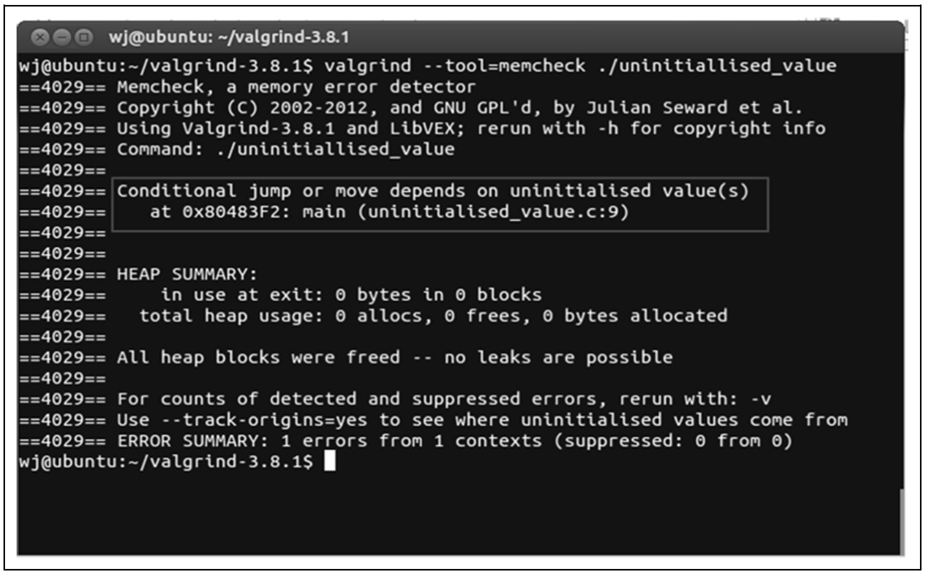


Figure 12. Experiment environment of taint analyses.



```
wj@ubuntu: ~/valgrind-3.8.1
wj@ubuntu:~/valgrind-3.8.1$ valgrind --tool=memcheck ./invalid_write
==5722== Memcheck, a memory error detector
==5722== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==5722== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==5722== Command: ./invalid_write
==5722==
==5722== Invalid write of size 4
==5722==    at 0x8048446: main (invalid_write.c:13)
==5722==  Address 0x41f1050 is 0 bytes after a block of size 40 alloc'd
==5722==    at 0x402BD74: malloc (vg_replace_malloc.c:270)
==5722==    by 0x8048428: main (invalid_write.c:9)
==5722==
==5722==
==5722== HEAP SUMMARY:
==5722==     in use at exit: 0 bytes in 0 blocks
==5722==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==5722==
==5722== All heap blocks were freed -- no leaks are possible
==5722==
==5722== For counts of detected and suppressed errors, rerun with: -v
==5722== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
wj@ubuntu:~/valgrind-3.8.1$
```

Figure 13. Invalid write checking.



```
wj@ubuntu: ~/valgrind-3.8.1
wj@ubuntu:~/valgrind-3.8.1$ valgrind --tool=memcheck ./uninitiallised_value
==4029== Memcheck, a memory error detector
==4029== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==4029== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==4029== Command: ./uninitiallised_value
==4029==
==4029== Conditional jump or move depends on uninitialised value(s)
==4029==    at 0x80483F2: main (uninitialised_value.c:9)
==4029==
==4029==
==4029== HEAP SUMMARY:
==4029==     in use at exit: 0 bytes in 0 blocks
==4029==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==4029==
==4029== All heap blocks were freed -- no leaks are possible
==4029==
==4029== For counts of detected and suppressed errors, rerun with: -v
==4029== Use --track-origins=yes to see where uninitialised values come from
==4029== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
wj@ubuntu:~/valgrind-3.8.1$
```

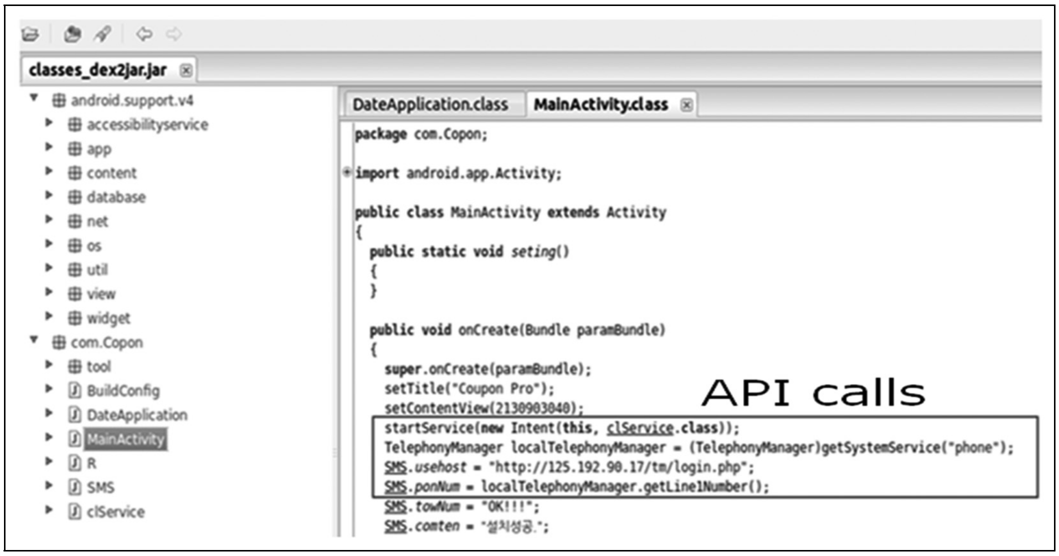
Figure 14. Conditioned jump on uninitialised value checking.

dex2jar.sh generates the jar file for classes.dex. In the disassemble process, the tool jd-gui can unzip the jar file and obtain the source code. After analysing the source code, the hacker was observed to have set a thread in clService in the downloaded package for automatically reporting the associated information to a zombie using the HttpPost method (Figures 15 and 16).

Step 3: DTPA

In DTPA, the SOE of a taint source is characterised using three subprocesses: (1) correlation analysis for MISs, (2) taint propagation analysis using an STA and (3) the determination of SOEs.

Step 3.1: correlation analysis for MISs. To identify the MISs of a taint source, the TC analysis tool ComDroid[7] was employed to analyse the statistical information of disassembled output from Dedexer and to log potential components and intent vulnerabilities. By using (3), detection probability of the MISs that appeared most for malicious behaviours were extracted from the collected file details.log regarding the intent creation and transmission operations in component modules (Table 9).



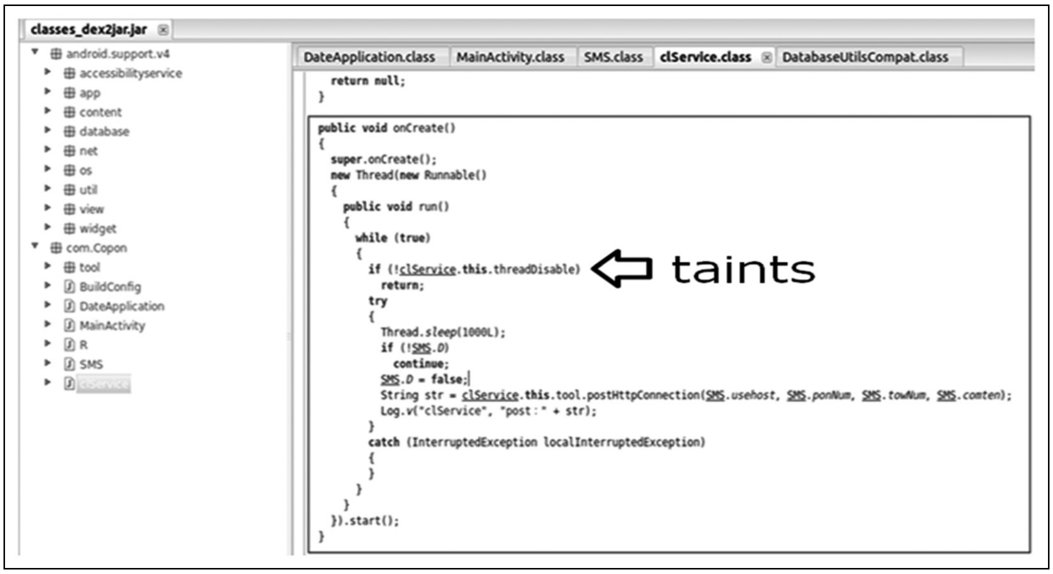
```
package com.Copon;

import android.app.Activity;

public class MainActivity extends Activity
{
  public static void seting()
  {
  }

  public void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    setTitle("Coupon Pro");
    setContentView(2130903040);
    startService(new Intent(this, clService.class));
    TelephonyManager localTelephonyManager = (TelephonyManager)getSystemService("phone");
    SMS.usehost = "http://125.192.90.17/tm/login.php";
    SMS.ponNum = localTelephonyManager.getLine1Number();
    SMS.towNum = "OK!!!";
    SMS.comten = "설치성공.";
```

Figure 15. Security check of the main activity for network apps.



```
    return null;
  }

public void onCreate()
{
  super.onCreate();
  new Thread(new Runnable()
  {
    public void run()
    {
      while (true)
      {
        if (!clService.this.threadDisable)
          return;
        try
        {
          Thread.sleep(1000L);
          if (!SMS.D)
            continue;
          SMS.D = false;
          String str = clService.this.tool.postHttpConnection(SMS.usehost, SMS.ponNum, SMS.towNum, SMS.comten);
          Log.v("clService", "post : " + str);
        }
        catch (InterruptedException localInterruptedException)
        {
        }
      }
    }
  }).start();
}
```

Figure 16. Taint check of the source code.

Step 3.2: taint propagation analysis using the WSTA. In this step, ComDroid was used with the WSTA to monitor tainted data for the following purposes: (1) inspect intent transmission from taint sources to the receiving module, (2) examine the intents of intermodule transmission and (3) determine whether the connected app and database are contaminated. In practice, ComDroid can track the intents of communications among apps by analysing the file details.log and listing information on activity, service and broadcast sink. Figure 17 illustrates the analysis of suspicious links between malware and an app after the module received intents. A defender can click on allActions to examine the activity in detail (Figure 18).

A defender can apply the analysis results of ComDroid to conclude the rule of taint propagation on the basis of the frequent episode patterns of security event logs using equation (1) for a specific taint source. The PRs derived from the experiment are shown in Table 10.

After the formulation of PRs, DTPA tools can automatically correlate these PRs to perform taint propagation analysis along with the possible taint paths. For example, for the APK:Geinimi-Banking Trojan invasion of cloud services (information available at www.ipay.com.cn), we downloaded samples from http://contagiominidump.blogspot.tw/2011/10/geinimi-banking-trojan-wwwipaycomcn.html to analyse the

Table 9. MISs of taint sources.

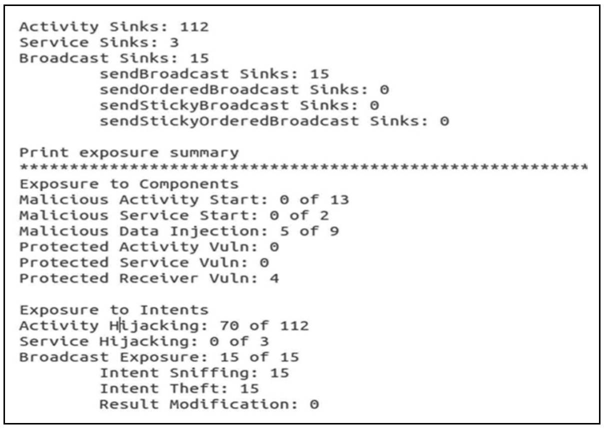
Malware	Package name	MISs
Smszombie	4d13d1bc63026b9c26c7cd4946b1bae0.apk	SendTextRequest, SendTextMessage, SendBroadcast
FakeGuard	com.stech.stopphishing.apk com.stech.spamguard.apk com.stech.stopphishing.apk	onReceive, connMgr.getActiveNetworkInfo

MIS: malicious instruction set.

Table 10. Examples of the taint propagation rule.

ID	Malware	Taint source	Package name	Calling sequence of MISs
1	SMSsender.apk	Sms_sent SubscriberId	com.c101421042723	[Source:SubscriberId, Sink1:Landroid/telephony/ TelephonyManager/ getSubscriberId]
2	vksafe.apk	File io SMS	com.vksafe	[Source:file io, Sink1:java/io/BufferedWriter/write, Sink2:com/ vksafe/WatcherService/openFileInput,Sink3:Ljava/io/ BufferedReader/readLine] [Source:sms,Sink1: Landroid/telephony/SmsManager/ sendTextMessage]

MIS: malicious instruction set.

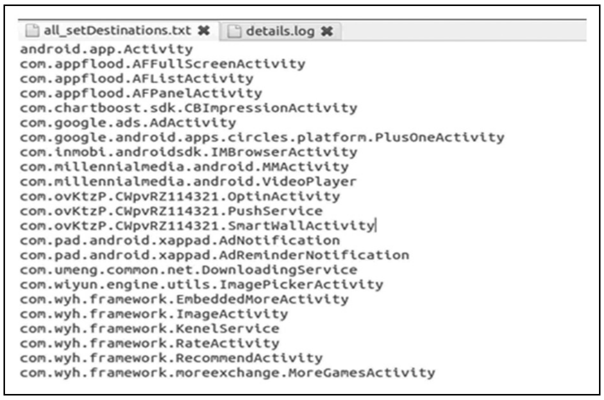


```
Activity Sinks: 112
Service Sinks: 3
Broadcast Sinks: 15
        sendBroadcast Sinks: 15
        sendOrderedBroadcast Sinks: 0
        sendStickyBroadcast Sinks: 0
        sendStickyOrderedBroadcast Sinks: 0

Print exposure summary
*********************************************************
Exposure to Components
Malicious Activity Start: 0 of 13
Malicious Service Start: 0 of 2
Malicious Data Injection: 5 of 9
Protected Activity Vuln: 0
Protected Service Vuln: 0
Protected Receiver Vuln: 4

Exposure to Intents
Activity Hijacking: 70 of 112
Service Hijacking: 0 of 3
Broadcast Exposure: 15 of 15
        Intent Sniffing: 15
        Intent Theft: 15
        Result Modification: 0
```

Figure 17. Application of ComDroid to inspect exposures between malware and an app.

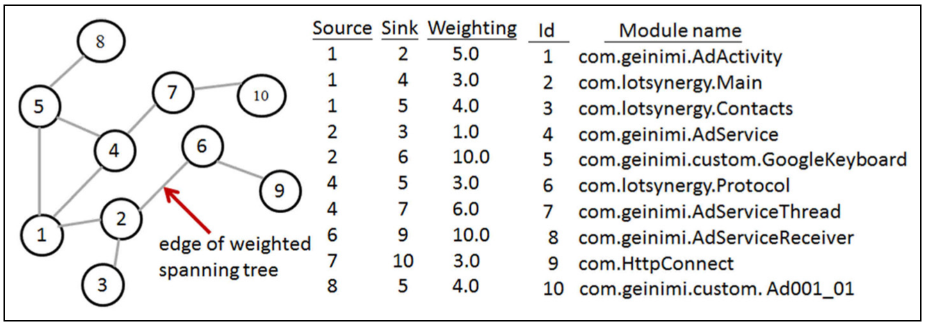


```
android.app.Activity
com.appflood.AFFullScreenActivity
com.appflood.AFListActivity
com.appflood.AFPanelActivity
com.chartboost.sdk.CBImpressionActivity
com.google.ads.AdActivity
com.google.android.apps.circles.platform.PlusOneActivity
com.inmobi.androidsdk.IMBrowserActivity
com.millennialmedia.android.MMActivity
com.millennialmedia.android.VideoPlayer
com.ovKtzP.CWpvRZ114321.OptinActivity
com.ovKtzP.CWpvRZ114321.PushService
com.ovKtzP.CWpvRZ114321.SmartWallActivity
com.pad.android.xappad.AdNotification
com.pad.android.xappad.AdReminderNotification
com.umeng.common.net.DownloadingService
com.wiyun.engine.utils.ImagePickerActivity
com.wyh.framework.EmbeddedMoreActivity
com.wyh.framework.ImageActivity
com.wyh.framework.KenelService
com.wyh.framework.RateActivity
com.wyh.framework.RecommendActivity
com.wyh.framework.moreexchange.MoreGamesActivity
```

Figure 18. Actions performed by the receiving intents in an app.

communication between Android apps and cloud services and then constructed a taint graph of information flow using the detail.log files derived from ComDroid associated with the WSTA (Figure 19). In setting the weights to an edge in a taint graph, a defender must consider the relative severity of data leaks. For example, a defender assigns a higher weight than that of mobile devices to the connection of a cloud server. The Kruskal MST algorithm was used to remove the taint-free source of information flow in communication and generate a new taint graph (Figure 20) to decide the priority of securing the taint paths.

Step 3.3: determination of SOEs. Vulnerabilities issued by United States Computer Emergency Readiness Team (US-CERT) for an Android platform are listed in Table 11. The proposed model was designed to describe the SOE and estimate the risk of each taint path for selecting appropriate safeguards to defend against cyber attacks. Once the system vulnerabilities, taint paths and PRs are identified, the defender can form the SOEs for tracking taint paths between the malicious website and the detected vulnerabilities of targeted network apps. The form for SOEs is defined as (ID_j, TS, PR_k, PR_{k+1}, PR_{k+2},…), generated by collecting a set of PRs in accordance with the taint sources and the relevant packages. Examples of the SOEs are shown in Table 12. Furthermore, the zombie infections for a taint source are located in Honeynet Map for managerial purposes.



Source	Sink	Weighting	Id	Module name
1	2	5.0	1	com.geinimi.AdActivity
1	4	3.0	2	com.lotsynergy.Main
1	5	4.0	3	com.lotsynergy.Contacts
2	3	1.0	4	com.geinimi.AdService
2	6	10.0	5	com.geinimi.custom.GoogleKeyboard
4	5	3.0	6	com.lotsynergy.Protocol
4	7	6.0	7	com.geinimi.AdServiceThread
6	9	10.0	8	com.geinimi.AdServiceReceiver
7	10	3.0	9	com.HttpConnect
8	5	4.0	10	com.geinimi.custom. Ad001_01

Figure 19. Spanning tree approach for analysing the taint path.

Table 11. Vulnerabilities for an Android platform.

CVE	Exploit pattern	Release date	Impact (10)	CVSS score
CVE-2014-8609	Android settings application privilege leakage vulnerability	15 December 2014	10.0	High (7.2)
CVE-2014-7911	Android <5.0 privilege escalation using ObjectInputStream	15 December 2014	10.0	High (7.2)

CVE: common vulnerability and exposure; CVSS: common vulnerability scoring system.

Table 12. Examples of SOEs.

ID	Taint source	Package name	Propagation rule
1	com.c101421042723	SMSsender.apk	[Source:SubscriberId, Sink1:Landroid/telephony/ TelephonyManager/getSubscriberId
2	com.lotsynergy	lottery.apk	[Source:SubscriberId,Sink1:Landroid/telephony/TelephonyManager/getSubscriberId] [Source:service,Sink1:Landroid/content/Intent] [Source:url,Sink1: Ljava/net/ URL/openConnection]
3	com.vksafe	vksafe.apk	[Source:file io, Sink1:java/io/BufferedWriter/write, Sink2:com/vksafe/ WatcherService/openFileInput, Sink3: Ljava/io/ BufferedReader/readLine] [Source:sms,Sink1:Landroid/telephony/SmsManager/ sendTextMessage

SOE: signature of exploit.

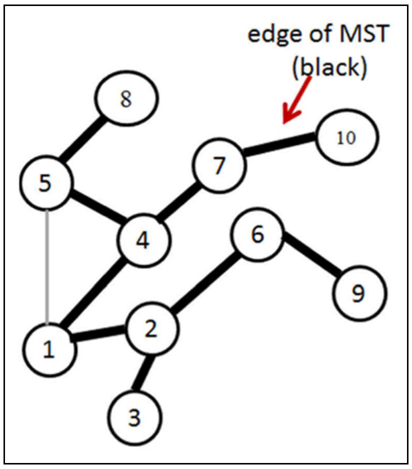


Figure 20. MST approach for analysing the taint path.

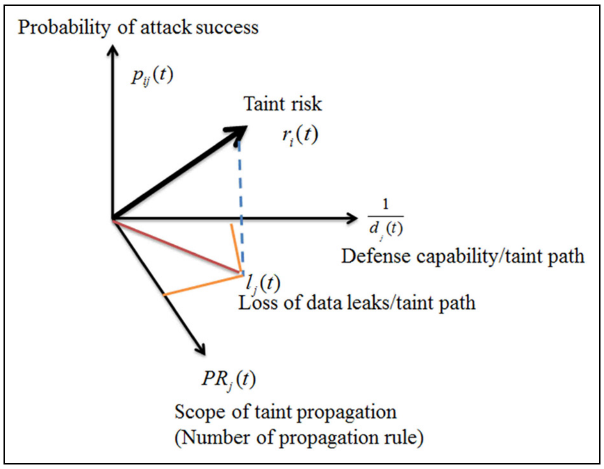

Probability of attack success
$p_{ij}(t)$
Taint risk
$r_i(t)$
$\frac{1}{d_j(t)}$
Defense capability/taint path
$l_j(t)$
Loss of data leaks/taint path
$PR_j(t)$
Scope of taint propagation
(Number of propagation rule)

Figure 21. Risk represented by loss of data leak and probability of attack success.

Table 13. Risks of a taint source with various spreads of taint propagation and impact.

Taint path	Probability of attack success p_{ij} (0–1)	Degree of taint propagation PR_j (0–1) (number of PRs/ maximum number of PRs)	Defence capability d_j (0–1)	Loss of taint propagation for the taint path j, l_j	Taint risk r_i (×100%)
j = 1	0.1	0.50 (4/8)	0.7 (MH)	0.714	71.4 (MH)
j = 2	0.5	0.75 (6/8)	0.5 (M)	1.500	75.0 (MH)
j = 3	0.9	1.00 (8/8)	0.9 (H)	1.110	99.9 (H)
j = 4	0.5	0.250 (2/8)	0.3 (ML)	0.833	41.6 (M)
j = 5	0.3	0.625 (5/8)	0.7 (MH)	1.071	32.1 (ML)
j = 6	0.7	0.750 (5/8)	0.5 (M)	1.250	87.5 (H)
j = 7	0.5	0.500 (4/8)	0.5 (M)	1.000	50.0 (M)
j = 8	0.9	0.375 (3/8)	0.5 (M)	0.750	67.5 (MH)
j = 9	0.7	0.250 (2/8)	0.5 (M)	0.500	35.0 (ML)
j = 10	0.3	0.875 (7/8)	0.7 (MH)	1.250	37.5 (ML)

PR: propagation rule.
The degree of taint propagation is calculated by converting individual value of PRs to normalised score and the defence capability is scored on a typical 5-point Likert scale, low(L) [0, 0.2), medium low (ML) [0.2, 0.4), medium (M) [0.4, 0.6), medium high (MH) [0.6, 0.8) and high(H) [0.8, 1.0].

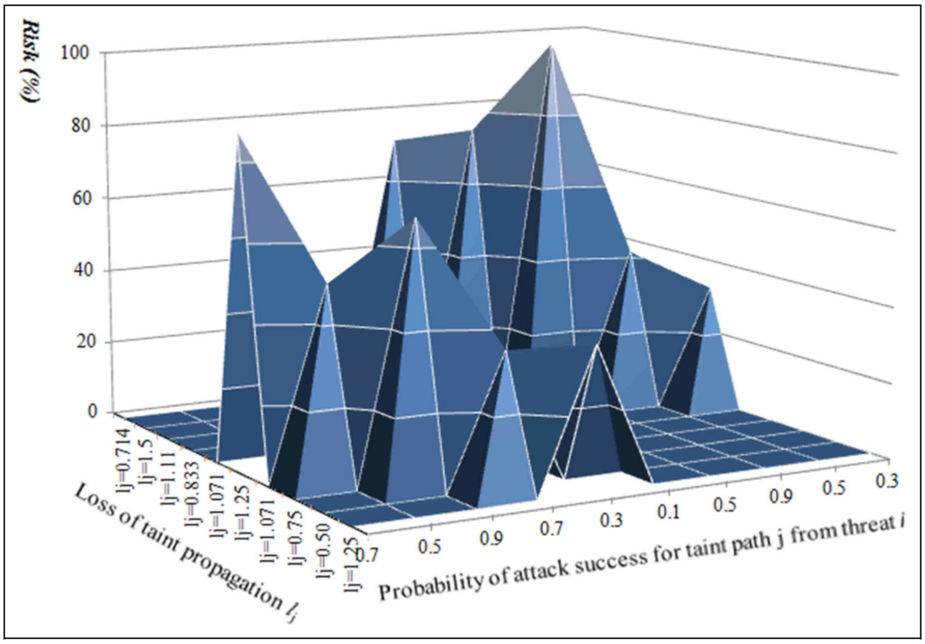


Figure 22. Risks of a taint source with various spreads of taint propagation.

Discussion

The taint risk in the DTPA process is discussed in the following. Because of limited contexts available, the experiment data sample for smart home system is reported in Table 13. In Table 13, 10 taint sources were tested using websites blacklisted by Google, including the captured malware samples from Contagio Blogger. Notably, taint sources were identified as risks from attacks caused by security holes in the apps and smart living services; Figure 21 shows the links among the probability of attack success for a taint source, the loss of possible spreads of the taint path and the taint risk. The results presented in Figure 22 show that the taint risk increases as both the degree of taint propagation and the loss of taint spread are increased. In addition, losses of tainted data propagation increase with an increasing degree of taint propagation but decrease with the defence capability. The high loss originated from the unsettled vulnerabilities that caused the spread of taint propagation represented by the number of PRs, which is associated with a low defence capability against taint propagation

$$l_{\mathrm{j}} = \underset{j}{\forall} \frac{PR_j}{d_j} \tag{7}$$

Conclusion

This article proposes a threat analysis model for solving the mobile security problem by performing in-depth behavioural analysis on the ComDroid platform to indicate the attack profile for malware infection, considering both code analysis and behaviour analysis for program vulnerabilities in a smart home system. In the proposed model, an improved DTPA scheme was obtained by revising the approach in Newsome and Song[4] and used as an enhancement method for the DTA of cloud security. Our scheme enables a defender to convert the spread of possible taint paths to loss and practically estimate the risk of a specific threat. Additionally, the proposed scheme not only considers the taint source, PRs and SOEs from tainted data to system vulnerability but also estimates the losses when tainted data are propagated. Consequently, the proposed method improves the defence reactions to a risk associated with the evolution of a system's security concerns and assists defenders in making appropriate decisions when tracing possible taint sources.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported jointly by TWISC@NCKU and the Ministry of Science and Technology of Taiwan under grant nos MOST 104-2632-E-168-001, MOST 104-2218-E-001-002 and MOST 105-2410-H-168-002.

References

1. Wikipedia. Taint checking, http://en.wikipedia.org/wiki/Taint_checking
2. Portokalidis G, Slowinska A and Bos H. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. *SIGOPS Op Syst Rev* 2006; 40(4): 15–27.
3. Yin H, Song D, Egele M, et al. Panorama: capturing system wide information flow for malware detection and analysis. In: *Proceedings of the 14th ACM conference on computer and communications security*, Alexandria, VA, 29 October–2 November 2007, pp.116–127. New York: ACM.
4. Newsome J and Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software, 2005, http://valgrind.org/docs/newsome2005.pdf
5. Wang P, Chao KM, Lo CC, et al. Using dynamic taint approach for of malware threat. In: *IEEE international conference on e-business engineering*, Beijing, China, 23–25 October 2015, pp.408–416. New York: IEEE.
6. Valgrind, http://valgrind.org/docs/manual/dist.news.html
7. Chin E, Felt AP, Greenwood K, et al. Analyzing inter-application communication in Android. In: *Proceedings of the 9th international conference on mobile systems (MobiSys2011)*, Bethesda, MD, 28 June–1 July 2011, pp.239–252. New York: ACM.
8. Desnos A. Androguard, http://code.google.com/p/androguard/wiki/Usage
9. The Honeynet project. Droidbox, 2012, http://www.honeynet.org/gsoc/slot11
10. McClurg J, Friedman J and Ng W. Android privacy leak detection via dynamic taint analysis, Northwestern University, Evanston, IL, 2013, http://www.jrmcclurg.com/papers/internet_security_final_report.pdf
11. Balakrishnan G and Reps T. WYSINWYX: what you see is not what you eXecute. *ACM Trans Program Lang Syst* 2010; 32(6): 1–84.
12. Enck W, Gilbert P, Chun BG, et al. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In: *Proceedings of the 9th Usenix symposium on operating systems design and implementation*, 2010, pp.393–408.
13. Kim HC, Keromytis AD, Covington M, et al. Capturing information flow with concatenated dynamic taint analysis. In: *Proceedings of international conference on availability, reliability and security*, Fukuoka, Japan, 16–19 March 2009, pp.355–362. New York: IEEE.
14. Rastogi V, Chen Y and Enck W. AppsPlayground: automatic security analysis of smartphone application. In: *Proceedings of the third ACM conference on data and application security and privacy (CODASPY13)*, San Antonio, TX, 18–20 February 2013, pp.209–220. New York: ACM.
15. Open Networking Foundation. Software-defined networking: the new norm for networks. *ONF White Paper*, 13 April 2012, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf
16. Braga R, Mota E and Passito A. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: *IEEE 35th conference on local computer networks (LCN)*, Denver, CO, 10–14 October 2010, pp.408–415. New York: IEEE.
17. Giotis K, Argyropoulos C, Androulidakis G, et al. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Comput Netw* 2014; 62: 122–136.
18. Jin R and Wang B. Malware detection for mobile devices using software-defined networking. In: *Proceedings of the 2013 second GENI research and educational experiment workshop (GREE)*, Salt Lake City, UT, 20–22 March, pp.81–88. New York: IEEE.
19. Feamster N. Outsourcing home network security. In: *Proceedings of the 2010 ACM SIGCOMM workshop on*

home networks, New Delhi, India, 3 September, pp.37–42. New York: ACM.
20. Kreutz D, Ramos FMV and Veríssimo P. Towards secure and dependable software-defined networks. In: *Proceedings of the second ACM SIGCOMM workshop on hot topics in software defined networking (HotSDN)*, Hong Kong, China, 16 August 2013.
21. TEMU, http://bitblaze.cs.berkeley.edu/temu.html
22. Tripp O, Pistoia M, Fink SJ, et al. TAJ: effective taint analysis of web applications. In: *Proceedings of the 30th ACM SIGPLAN conference on Programming language design and implementation*, Dublin, 15–20 June 2009, pp.87–97. New York: ACM.
23. Mannila H, Toivonen H and Verkamo IA. Discovery of frequent episodes in event sequences. *Data Min Knowl Disc* 1997; 1(3): 259–289.
24. Chad WA, Bamshad M and Robin B. Defending recommender systems: detection of profile injection attacks. *Serv Oriented Comput Appl* 2007; 1(3): 157–170.
25. Contagio malware dump, http://contagiodump.blogspot.tw (accessed February 2015).