行政院國家科學委員會專題研究計畫 成果報告

惡意程式偵測與樣本資料庫之建置與研究
研究成果報告(精簡版)

計 畫 類 別	：個別型
計 畫 編 號	：NSC 100-2218-E-009-024-
執 行 期 間	：100 年 10 月 01 日至 101 年 07 月 31 日
執 行 單 位	：國立交通大學資訊工程學系（所）

計 畫 主 持 人：林盈達

報 告 附 件 ：出席國際會議研究心得報告及發表論文

公 開 資 訊 ：本計畫可公開查詢

中 華 民 國 101 年 09 月 10 日

中 文 摘 要 ： 我們利用開放程式碼開發了一個可收集、偵測及分析惡意程式之工具鏈。此工具鏈可透過三種管道: 電子郵件、網路連結、及端點對端點，來主動收集惡意程式。我們的方法可以在一個月內收集超過 800 支惡意程式，而傳統被動式收集只能收集 354 支惡意程式。更重要地，我們所收集的惡意程式有超過 16%是在過去利用傳統被動式收集所無法收集到的。相較於被動式收集的惡意程式有超過 79%皆為殭屍型惡意程式，我們所設計的方法所收集的惡意程式分別有 59%的木馬惡意程式及其他類型的惡意程式。另外，由於手機越來越受到歡迎，攻擊者可將惡意程式碼嵌入在正常應用程式中以攻擊使用者，因此，我們設計了一個利用系統呼叫的行為分析方法來偵測 Android 平台上的惡意程式。我們抽取惡意程式的共通系統呼叫子序列並且利用 Bayers 機率模型來過濾那些在重封裝惡意程式中較低出線機率的序列。透過我們所設計的方法，偵測惡意程式的正確率可以達到 97.6%。

中文關鍵詞： 開放程式碼、惡意程式收集、工具鏈、Android、系統呼叫

英 文 摘 要 ： We propose an open-source malware tool chain includes malware collection, detection, and analysis. It actively collects malware through three channels: e-mail, web-links, and peer-to-peer. The experimental results show the differences between the passive approach and our active mechanism in the aspects of quantity, timeliness, distribution, and activeness of captured malware. Our mechanism captures 800 malware programs in one month, while the passive approach collects 354 malware programs. Furthermore, 16% of actively captured malware were still not defined in the databases of anti-virus products, while none of them had been captured by the passive approach. 79% of the captured malware in the passive approach are bots and 59% of the captured malware in our mechanism are Trojan horses. The passive collection and active collection have more malware with strong activeness and weak activeness, respectively. In addition, mobile devices security becomes highly desirable. Adversaries can easily repackage the malicious code into the different benign applications for distribution. The work of detecting and analyzing the malicious application becomes a challenge of Android. We propose a behavior-based detection mechanism based on system call sequences. We extract the common system call subsequences of malicious applications and purpose a comparison approach to deal with multiple threads produced by the applications. We also utilize the Bayes probability model to filter subsequences which have lower probability of appearance in the repackaged applications. Finally, we can detect repackaged applications by those extracted subsequences. The detection result demonstrates that our approach has 97.6% high accuracy.

英文關鍵詞： open-source, malware collection, tool chain, Android, system call

摘要

我們利用開放程式碼開發了一個可收集、偵測及分析惡意程式之工具鏈。此工具鏈可透過三種管道: 電子郵件、網路連結、及端點對端點，來主動收集惡意程式。我們的方法可以在一個月內收集超過 800 支惡意程式，而傳統被動式收集只能收集 354 支惡意程式。更重要地，我們所收集的惡意程式有超過 16%是在過去利用傳統被動式收集所無法收集到的。相較於被動式收集的惡意程式有超過 79%皆為殭屍型惡意程式，我們所設計的方法所收集的惡意程式分別有 59%的木馬惡意程式及其他類型的惡意程式。另外，由於手機越來越受到歡迎，攻擊者可將惡意程式碼嵌入在正常應用程式中以攻擊使用者，因此，我們設計了一個利用系統呼叫的行為分析方法來偵測 Android 平台上的惡意程式。我們抽取惡意程式的共通系統呼叫子序列並且利用 Bayers 機率模型來過濾那些在重封裝惡意程式中較低出線機率的序列。透過我們所設計的方法，偵測惡意程式的正確率可以達到 97.6%。

關鍵字: 開放程式碼、惡意程式收集、工具鏈、Android、系統呼叫

Abstract

We propose an open-source malware tool chain includes malware collection, detection, and analysis. It actively collects malware through three channels: e-mail, web-links, and peer-to-peer. The experimental results show the differences between the passive approach and our active mechanism in the aspects of quantity, timeliness, distribution, and activeness of captured malware. Our mechanism captures 800 malware programs in one month, while the passive approach collects 354 malware programs. Furthermore, 16% of actively captured malware were still not defined in the databases of anti-virus products, while none of them had been captured by the passive approach. 79% of the captured malware in the passive approach are bots and 59% of the captured malware in our mechanism are Trojan horses. The passive collection and active collection have more malware with strong activeness and weak activeness, respectively. In addition, mobile devices security becomes highly desirable. Adversaries can easily repackage the malicious code into the different benign applications for distribution. The work of detecting and analyzing the malicious application becomes a challenge of Android. We propose a behavior-based detection mechanism based on system call sequences. We extract the common system call subsequences of malicious applications and purpose a comparison approach to deal with multiple threads produced by the applications. We also utilize the Bayes probability model to filter subsequences which have lower probability of appearance in the repackaged applications. Finally, we can detect repackaged applications by those extracted subsequences. The detection result demonstrates that our approach has 97.6% high accuracy.

Keyword: open-source, malware collection, tool chain, Android, system call

1. Introduction

Malware is a collective term for a variety of nefarious-purpose software that enters a system without a user's authorization. Examples of malware include worms, viruses, Trojan horses, rootkits, backdoors, and more recently, bots. Malware can come from many different sources. For instance, a malware can be downloaded to a computer when a user clicks on a malicious URL, or it could arrive at a computer as an e-mail attachment or files acquired from P2P file sharing network.

In order to know how to prevent malware attacks, malware collection is necessary. Herein, we present an open-source tool chain [1], Honey-Inspector, which actively collects malicious software from emails, Web-links, and P2P file sharing software. In addition, we detect malware using anti-virus software and design analysis tools to analyze the behavior of malware. Most works collect malware through emails or Web-links. Recently, P2P file sharing software is another critical path of malware infection [2-5]. Our analysis tools focus on analyzing host and network behaviors of malware. In our mechanism, we combine and automate malware collection, detection, and analysis into the tool chain. It would speed up malware collection and analysis.

As cloud-based applications become popular, preventing damages by malware is a major security challenge in the mobile environments [6]. Since Android is one of the most popular mobile operating system, we propose a behavior-based analysis mechanism to identify malicious applications on Android. Our proposed scheme is a novel behavior-based detection version which relies on system call sequences. The key idea is to observe the system call sequences that are trigger by applications, i.e., although a malicious code can camouflage into a benign application, the malicious behavior still appears in the system call sequences. To filter out the benign behaviors, we also employ the Bayes probabilistic model for evaluating the appearance probability of extracted system call sequences to find the significant malicious sequences. Through this approach, the repackaged malware can be detected with high accuracy and the false positive rate can be reduced significantly.

2. Research Objective and Methods

Existing malware collection techniques usually rely on honeypots [7], or other methods. Honeypots are one of the major techniques used to gain malware. A traditional honeypot [8] usually provides vulnerabilities of publicly known systems or unpatched system software to lure attackers. Nevertheless, this type of honeypot cannot discover malicious events from client-side attacks [9] that download malicious programs to victims' hosts when victims browse emails or websites. As a result, server honeypots are slow in collecting malware, and might not detect malware programs with client-side attacks at all. Our mechanism is a high-interaction client honeypot and executes malware in a virtual machine. Our virtual machines can imitate applications' behavior and respond in such a way that each malware perceives it is in a real system.

- In the proposed framework, we utilize multiple A/V tools to determine if a collected binary is malicious or not in order to obtain a low false negative (FN) rate. Overall, there are two different types of malware detection strategies [10]: *Behavior-based* detection: This detection method can detect zero-day attacks which are unknown to malware detectors. The difficulty of this technique is determining what features to memorize. And *signature-based* detection: This technique tries to define the behavior or characteristics of malware (signature), and then stores these definitions in a database. Then, it can detect if a program/process is malicious according to the signature database.

The purpose of the proposed mechanism is to collect new malware and analyze the behavior of malware. Our mechanism collects malware from malicious URLs, emails, and P2P file sharing software. We have also designed tools to analyze the host and network behaviors of the captured malware. We present the tool chain, Honey-Inspector, which includes Proactive Malware Capture and Detection (PMC&D), Host Behavior Analysis (HBA), and Network Behavior Analysis (NBA). The architecture and process of Honey-Inspector are shown in Figure 1 and the number in Figure 1 is the order of our system workflow.

First, PMC&D proactively collects suspicious files from existing network applications, including URLs, emails, and P2P file sharing techniques. It searches for malicious keywords using the P2P file sharing technique or on websites. Then, PMC&D divides the downloaded files into benign and malicious categories using multiple A/V software programs. Once a suspicious program is identified as malicious software by one of multiple A/V software, the module classifies the malware and stores it in the database. HBA and NBA run the malware program on the virtual machine based on the Microsoft Windows XP platform. After the malware is executed, it may modify the file system and the registry of the virtual machine. HBA takes a snapshot of the file system and the registry of the virtual machine before the malware is executed, and then compares this to the infected file system and registry after running the malware. On the other hand, the module in NBA traces the network traffic when malware is executed. The goal of the HBA and NBA is to analyze the host and the network behavior of the malware, respectively. When a malware program runs on a virtual machine, it may change the file system or the registry, or launch network attacks, such as a Denial of Service (DoS) attack. HBA and NBA can analyze the behavior of the malware in the infected virtual machine.

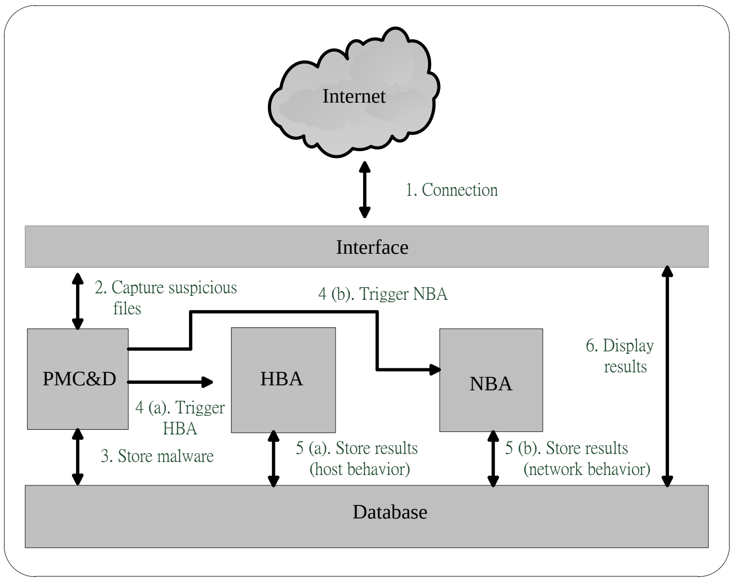


Figure 1: The architecture and process of Honey-Inspector

In Figure 1, PMC&D gets attached files and parses suspicious URLs or keywords from email servers. Then, PMC&D downloads malicious files from websites or P2P file-sharing software by keyword searching. Whenever PMC&D finds suspicious files, it stores them in the database. Then, PMC&D detects suspicious software by four A/V tools. At present, our mechanism uses four A/V tools, namely Kaspersky, Avast, Avira AntiVir, and Nod32. If a suspicious file is identified as malware by one of the A/V tools, PMC&D stores the malware and detection results in the database. The detection is not perfect. It may not be able to detect unknown malware or malware with evasion capabilities such as the multi-process malware design [11]. However, if storage space is not an issue, PMC&D can be set to keep every collected file regardless of the detection result by the A/V tools. We assume that some malware programs do modify the contents of file systems and registries. In Figure 1, HBA sets up a new virtual machine based on Microsoft Windows XP platform and makes a copy of the clean registry and the file system. Then, HBA runs a malware program. Once the malware has modified the registry and/or the file system of the virtual machine, HBA uses the DiffReg module to detect the infected registry and the DiffFS module to identify the file system in the infected virtual machine. The above modules can find what the malware modified. Finally, HBA stores the comparison results of the malware execution in the database. We also assume that some malware programs would generate the network traffic. In Figure 1, NBA sets up new virtual machines and executes a malicious program on individual virtual machine for a period of time. The Netsniff module monitors the network traffics of virtual machines. If a malware program generates network traffic, such as sending an email or ICMP packet transmission, the Netsniff module finds out malicious network links and stores the traces of network traffic in the database. Finally, NBA removes the malware from the infected virtual machines.

None of our active collection mechanism ever invokes the execution of a collected sample. The connection rate of the URL crawling process is rate-controlled. As a result, the active collection mechanism itself will not cause any kind of damage to the Internet. The behavior analysis modules (NBA and HBA) do require putting a collected sample into execution. However, the analysis environment is a closed network environment. The malware will not be able to damage the Internet either.

In mobile environments, to disseminate malware to users, attackers usually embed their malicious codes into the normal applications, and then publish these repackaged applications to the official Android market or the third-party market [12]. This dissemination path is simple, efficient, and effective because the repackaged applications are like benign applications to fool users into obtaining the secret information [13]. In order to detect the malicious repackaged applications, the proposed mechanism observers the behaviors during the execution period of repackaged applications rather than the outward appearance of repackaged applications. The key idea is that, even attackers can embed the malicious codes into varies applications, the behaviors of executing applications are consistent with the barely malicious code instructions. Hence, the proposed mechanism observes the execution behaviors of application and then extracts the common behavior patterns to distinguish malware.

For collecting application behaviors, we utilize the kernel-based system calls [14] which provide interact between processes and the operating system kernel for receiving service or resource requests. All service and resource requests can be observed by monitoring the system call interface, and the sequential requests form up a sequence of system calls. The observed system call sequence can be regarded as the request behaviors during the application executing. Although function calls of higher layers also provide the similar functionality to form up call sequences, it would lead ambiguous because of its multi-interfaces. Hence, compared with other methods [15-19], the proposed mechanism used system calls not only simply indicate the running behavior of applications, but also record the behavior of malicious codes completely than using high level function call.

Therefore, for detecting malicious repackaged applications, we initially collect a set of same type repackaged applications M. Let M_i denotes the i-th repackaged application in the set M. Then, for each M_i, we record the corresponding S_i as a set of system call sequences. After that, we can extract the common system call subsequences S_{sign} from the set $\{S_1, S_2, S_3, \ldots, S_{|M|}\}$ to derive the common behavior patterns. Finally, for those undiscovered repackaged applications E_{mal} with the same malicious codes, we can detect them from a set of applications to be inspected E by S_{sign}. Figure 2 shows the overview of our approach. For extracting the common behavior patterns, three phases, including recording phase, extraction phase, and evaluation phase, are involved in the proposed approach. In the following, we first describe these phases, respectively.

Recording Phase

In this phase, for all of applications in M, we sequentially select an application M_i from M, and then execute the M_i individually to record the system call requests during the Android system runtime. Next, we extract S_i from all of system call data by the process ID of M_i and the corresponding child thread IDs. After all of S_i extracting, we can obtain S in the recording phase.

Extraction Phase

The extraction phase aims to extract a set of common system call subsequences C from all of S_i. Since the same malicious code can trigger same system call sequences, we extract C by *Longest Common Substring* (LCSs) algorithm for comparing system call sequences of different repackaged applications. In addition, to improve the efficiency, we also present a mechanism for reducing the time cost of extracting within multi-thread system call sequences. Finally, C is obtained and each common system call subsequence in C is denoted as C_j, where j is the index of subsequences in C. It needs to address clearly that we extract substring in the system call sequences. The substring is a consecutive part of system call sequences, but we still denote it as subsequences since a substring is always a subsequence and most works use subsequences to call their system call combinations.

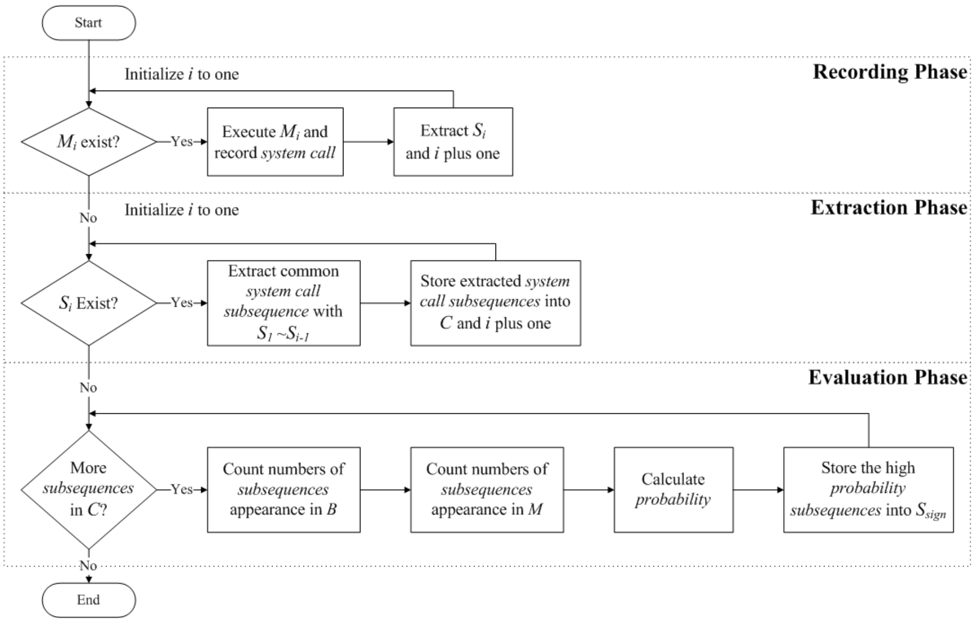

Start
Recording Phase
Initialize i to one
M_i exist?
Yes
Execute M_i and record system call
Extract S_i and i plus one
No
Extraction Phase
Initialize i to one
S_i Exist?
Yes
Extract common system call subsequence with S_1 ~S_i-1
Store extracted system call subsequences into C and i plus one
No
Evaluation Phase
More subsequences in C?
Yes
Count numbers of subsequences appearance in B
Count numbers of subsequences appearance in M
Calculate probability
Store the high probability subsequences into S_sign
No
End

Figure 2. Flowchart of System Call Sequence Processing

Evaluation Phase

However, it is impossible for all of the extracted subsequence to be used as the detection patterns. This is because some system call subsequences not only appear in repackaged applications, but also in benign applications. To overcome this problem, we leverage the Bayes probabilistic model to calculate the probability of C_j by counting the number of each subsequence appearance in a set of benign applications B and M. After that, we can indicate those non-discriminating subsequences in C and filter out them because they have the lower probability. After filtering, we can get S_{sign}, which has the higher probability, appearance in malware applications but have a lower probability appear in benign applications at the end of this phase. If an application matches the same system call subsequence pattern of S_{sign} in execution time, we can claim that the application is repackaged with the same malicious code.

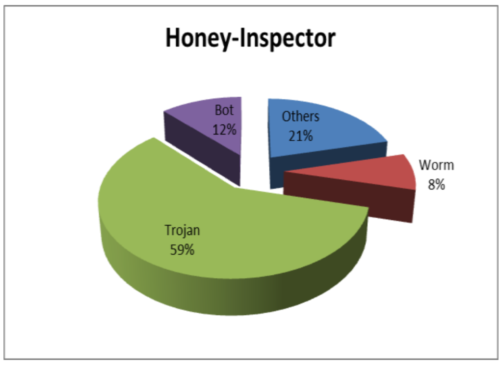
We have to evaluate the extracted subsequences in terms of the probability of appearance under the Bayes probability model. The probability formula is calculated as

$$P(M \mid C_j) = \frac{P(C_j \mid M) \cdot P(M)}{P(C_j \mid M) \cdot P(M) + P(C_j \mid B) \cdot P(B)}, \tag{1}$$

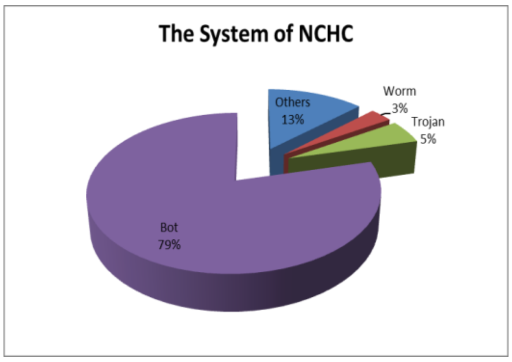
where the C_j is the system call subsequence that is used to evaluate. $P(B)$ denotes the probability that the given applications are benign applications. $P(M)$ denotes the probability that the given applications is a repackaged application. $P(C_j|B)$ denotes the probability that the subsequences appear in benign applications, and $P(C_j|M)$ denotes the probability that the subsequences appear in repackaged applications. Finally, we can get the probability $P(M|C_j)$ that an application is a repackaged application, detected by the specific system call subsequence. In the first step of evaluation phase, we count the number of sequences occurrence in benign applications and malicious applications. Then, in the second step, we can calculate the probability $P(M|C_j)$ by the formula (1). We can obtain the subsequences that appeared in the malicious application has the higher probability than that of benign applications. And this information helps us to find S_{sign} which achieves the higher accuracy for detecting repackaged applications.

3. Results and Discussions

The distributions of captured malware for Honey-Inspector and the NCHC system are shown in Figure 3 (a) and (b). Here, we analyze the distribution of captured malware from the passive system and Honey-Inspector. For the passive NCHC system, 79% of the captured malware are bots and 21% are worms, Trojan horses, and other malware. Because bots are able to perform active attacks, they can be easily captured by the passive system. However, 59% of the malware captured by Honey-Inspector are Trojan horses, and 41% are bots, worms, and other malware. Trojan horses usually hide inside files [20]. As a result, Honey-Inspector will have a higher chance capturing Trojans as it actively seeks for potential malware binary files from multiple sources. In comparison, the passive NCHC honeypot system will have to wait till a Trojan initiating a remote attack to have the chance of capturing the Trojan binary.



(a) The distribution of captured malware for Honey-Inspector



(b) The distribution of captured malware for the NCHC's system

Figure 3. The distribution of captured malware for Honey-Inspector and the NCHC system

Table 1 shows the behavior analysis results of captured malware. The notations in Table 1 are shown in the following:

A: (*high activeness*) malware that exhibits both host behavior and network behavior
B: (*medium activeness*) malware that exhibits network behavior only
C: (*low activeness*) malware that exhibits host behavior only
D: (*no activeness*) malware that exhibits no action

In Table 1 (a), we analyze the behavior of the captured malware by Honey-Inspector and the NCHC system. Honey-Inspector can collect malware with strong activeness, including malware that has both host behavior and network behavior, or only network behavior such as 77% of captured malware in class A and class B. Furthermore, Honey-Inspector can also capture malware with weak activeness, such as 23% of captured malware in class C and D. However, the NCHC system collects 2% of captured malware in class C and D. Therefore, compared to the NCHC system, our mechanism can collect malware with strong or weak activeness.

In Table 1 (b), the percentages of captured bots in class A and B of Honey-Inspector are the same with those of the NCHC system. A possible reason is that most bots have strong activeness.

Due to the property difference of active and passive malware collection methods, the activeness of captured malware in the proposed mechanism is different from that in the NCHC system.

Table 1: The behavior analysis of captured malware (bots)

(a) The behavior analysis of captured malware

System / Item	Honey-Inspector	The system of NCHC
A	48%	67%
B	29%	31%
C	12%	1%
D	11%	1%

(b) The behavior analysis of captured bots

System / Item	Honey-Inspector	The system of NCHC
A	35%	67%
B	63%	31%
C	0	1%
D	2%	1%

For evaluating our approach, we prepare five different repackaged application types: *Kmin* [21], *Geinimi* [22], *DroidDream* [23], *BaseBridge* [24] and *DroidDream Light* [25]. We also collect two sets of benign applications. The first set is used to evaluate the appearance probability of common system call subsequences for excluding useless subsequences, and the other set to evaluate the false positive of detection rate.

After the extraction, we count the number of subsequences appearance in the benign application set and the repackaged application set from which the subsequences are extracted for the training of Bayes probability. At first, we prepare 300 benign applications as the benign application set. After the calculation by Bayes probability model, Figure 5 shows the probability distribution of subsequences. We calculate the probability in five different types of repackaged applications, and the number of subsequences is the sum of five results. Most of subsequences are distributed at the interval 10%~15% and 100%. The higher probability of subsequences means if the subsequences are discovered in an application, it has higher probability that the application is repackaged. We select higher probability subsequences as the significant common system call subsequences.

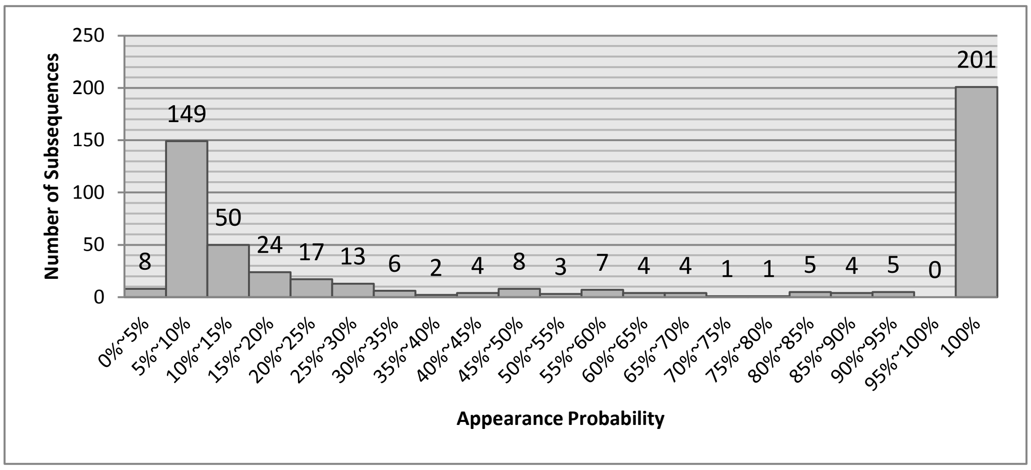


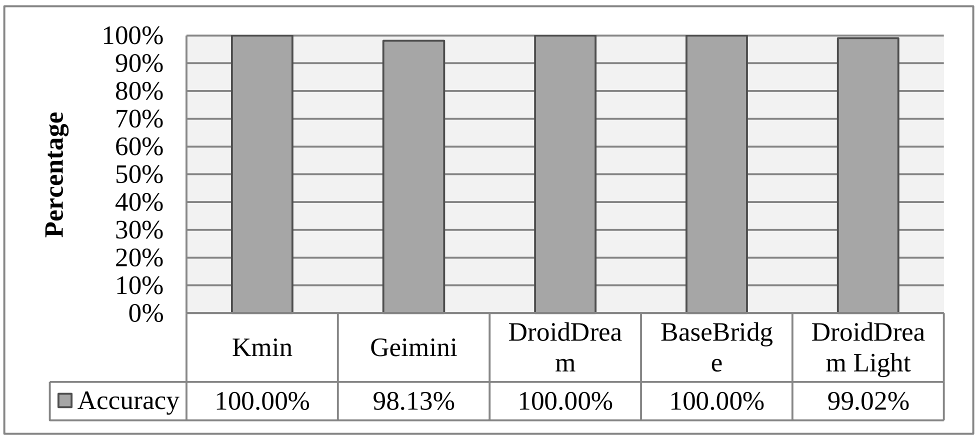
Figure 5. Probability Distribution of Subsequences

With regarding the detection accuracy, we also prepare 100 benign applications to evaluate the false positive of detection rate. We put the significant common system call subsequences which are extracted from different types independently of the five type sets. Then, we take those subsequences as the pattern to match in the system call sequences of the same type of repackaged applications. An application is detected as a repackaged application if it has the same subsequence in its system call sequences. Table 5 shows the detected result. Most of repackaged applications can be detected by our approach, except the application *DDL – 1*.

Suppose that true positive (TP) denotes the percentage of repackaged applications detected correctly, and false negative (FN) denotes the percentage of repackaged applications detected incorrectly; on the contrary true negative (TN) denotes the percentage of benign applications which are not detected by any excluded subsequences, and false positive (FP) be the percentage of benign applications which are incorrectly detected as repackaged applications. B denotes the number of benign applications. M denotes the number of repackaged applications. Table 2 demonstrates the true positive and true negative of detection rate that we evaluated with prepared samples. And the *Accuracy* can be calculated as

$$Accuracy = \frac{TP \cdot M + TN \cdot B}{(TP + FN) \cdot M + (TN + FP) \cdot B} \times 100\%. \qquad (2)$$

Our approach has a good accuracy rate to detect repackaged applications. It is worth noting that, our approach has a very low false positive rate and a false negative rate. For all of five type samples, we only miss one evaluated target in 25 repackaged applications. For the 100 benign evaluation samples, our approach only has 2 false positive of benign samples. The accuracy is 97%. Compared with other experimental results, only the true positive rate of *DroidDream Light* is significantly lower than that of our results, but we only miss one target. And this could be explained by an insufficient training and evaluation samples of *DroidDream Light*.



	Kmin	Geimini	DroidDream	BaseBridge	DroidDream Light
▨ Accuracy	100.00%	98.13%	100.00%	100.00%	99.02%

Figure 6. Accuracy of Detection

Table 2. True Positive and True Negative

Malware Type	*Kmin*	*Geinimi*	*DroidDream*	*BaseBridge*	*DroidDream Light*
True Positive	100%	100%	100%	100%	50%
True Negative	100%	98%	100%	100%	100%

4. Conclusion

In this work, we propose an open-source malware tool chain, Honey-Inspector, which performs active collection, detection, and analysis of malware. In our experimental results, we find that our system can capture more malware than the passive NCHC system. We also analyzed the distribution and activeness of captured malware. Bots reflect the highest percentage of captured malware in the passive system, whereas most of the captured malware in the proposed mechanism are Trojan horses. Therefore, the proposed mechanism could collect them easily. In addition, we present an approach with the concept on extracting the common behaviors of repackaged applications in system call sequences. The detection only requires a few repackaged applications with the same type to extract the common system call subsequences. In addition, our approach does not need to collect and compare the original benign applications with the repackages applications. We take those extracted common system call subsequences as the behavior patterns to detect repackaged applications. In our experiment, we use five different types of repackaged applications to evaluate the accuracy rate. Our approach extracts 238 common system call subsequences from training samples, and the detection result demonstrates that our approach has higher true positive rate in detecting most repackaged applications. We evaluate 25 repackaged applications and only miss one evaluated target. Our approach also has higher true negative rate in verifying benign applications. The accuracy of detection rate is 97.6% in all of evaluation applications.

In future works, we shall extend the period of malware collection and find out the possibly identical malware from NCHC's system and Honey-Inspector. We also plan to improve the disadvantages of a high-interaction honeypot. In addition, we shall add new sources of malware collection and new tools for analyzing malware behavior in order to capture more and newer malware programs. Simultaneously, we would study application behaviors analysis. We need to design a tool to capture the complete behaviors in the applications when they are triggered. Moreover, we also hope to develop an on-device detector of system call sequences detection which can work like the anti-virus software that directly detects repackaged applications on mobile devices.

References

[1] "A Malware Tool Chain: Honey-Inspector," http://honeyinspector.sourceforge.net/.

[2] Y. M. Wang, "Strider HoneyMonkeys: Active Client-side Honeypots for Finding Web Sites that Exploit Browser Vulnerabilities," *Proceedings of Part of Works in Progress at the 14th USENIX Security Symposium*, 2007.

[3] K. Wang, HoneyClient, Version 0.1.1, http://www.honeyclient.org/trac/.

[4] "Capture-HPC", https://projects.honeynet.org/capture-hpc/.

[5] Y. Alosefer and O. Rana, "Honeyware: A Web-based Low Interaction Client Honeypot," *Proceedings of the 3rd International Conferences on Software Testing, Verification, and Validation Workshops* (ICSTW), IEEE, 2010, pp. 410-417.

[6] W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android security," *IEEE Security & Privacy Magazine*, vol. 7, no. 1, pp. 10–17, 2009.

[7] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in Network Security: A Survey," *Proceedings of International Conference on Communication, Computing & Security* (ICCCS 2011), ACM, pp. 600-605, 2011.

[8] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. C. Freiling, "The Nepenthes Platform: An Efficient Approach to Collect Malware," *Proceedings of Recent Advances in Intrusion Detection* (RAID), Springer, Vol. 4219, pp. 165–184, 2006.

[9] The Honeynet Project, http://www.honeynet.org/node/157.

[10] N. Idika and A. P. Mathur, "A Survey of Malware Detection Techniques," 2007, http://www.serc.net/system/files/SERC-TR-286.pdf.

[11] M. Ramilli, M. Bishop, and S. Sun, "Multiprocess Malware", *International Conference on Malicious and Unwanted Software*, pp. 8-13, 2011.

[12] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting Repackaged Smartphone Applications in Third-party Android Marketplaces," *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy,* San Antonio, TX, USA, February 2012.

[13] Y. Zhou, and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," *Proceedings of the 33rd IEEE Symposium on Security and Privacy,* San Francisco, CA, May 2012.

[14] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel, "Anomalous System Call Detection," *ACM Transactions on Information and System Security,* vol. 9, no. 1, pp. 61–93, February 2006.

[15] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N.

Sheth, "TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones," *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, Vancouver, BC, Canada, pp. 393–407, October 2010.

[16] W. Enck, M. Ongtang, and P. McDaniel, "On Lightweight Mobile Phone Application Certification," *Proceedings of the 16th ACM conference on Computer and communications security*, Chicago, IL, USA, pp. 235–245, November 2009.

[17] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "SCanDroid: Automated Security Certification of Android Applications," Technical report, University of Maryland, 2009.

[18] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, "An Android Application Sandbox System for Suspicious Software Detection," *Proceedings of the 5th International Conference on Malicious and Unwanted Software (Malware 2010)*, Nancy, France, pp. 55–62, 2010.

[19] I. Burguera, U. Zurutuza, and N. T. Simin, "Crowdroid: Behavior-based Malware Detection System for Android," *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, Chicago, IL, USA, pp. 15–25, October 2011.

[20] "How to Identify Trojan Malware," http://www.spamlaws.com/identify-trojan-malware.html.

[21] "Encyclopedia entry: Trojan:AndroidOS/Kmin.A," available at: http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Trojan%3AAndroidOS%2FKmin.A .

[22] T. Strazzere, and T. Wyatt, "Geinimi Trojan Technical Teardown," *Lookout Mobile Security*, 2011.

[23] "Lookout Mobile Security Technical Tear Down," Lookout Mobile Security.

[24] "Android.Basebridge," available at: http://www.symantec.com/security_response /writeup.jsp?docid=2011-060915-4938-99 .

[25] "Security Alert: New DroidDream Light Variant Published to Android Market," available at: http://blog.mylookout.com/blog/2011/07/08/ security-alert-new-droiddream-light-variant-published-to-android-market/.

國科會補助專題研究計畫項下出席國際學術會議心得報告

日期：101_年7月16_日

計畫編號	NSC 100－2218－E－009－024-		
計畫名稱	惡意程式偵測與樣本資料庫之建置與研究		
出國人員姓名	林盈達	服務機構及職稱	國立交通大學資工系教授
會議時間	101 年 6 月 2 日至 101 年 6 月 3 日	會議地點	香港
會議名稱	(中文)國際先進資訊科技會議 (英文) ICAIT (International Conference on Advanced Information Technology)		
發表論文題目	(中文）以入侵偵測系統萃取漏擋與誤擋之真實流量 (英文) False Positives and Negatives from Real Traffic with Intrusion Detection/Prevention Systems		

一、參加會議經過

6/1 中午 台北→香港
　　下午 拜訪香港科技大學(HKUST) Prof. Qian Zhang
6/2-3 9AM-5PM 研討會
6/3 下午 香港→澳門
6/4 下午 澳門→台北

二、與會心得

此次參與 ICAIT (International Conference on Advanced Information Technology)報告一篇論文:

Cheng-Yuan Ho, Ying-Dar Lin, Yuan-Cheng Lai, I-Wei Chen, Fu-Yu Wang and Wei-Hsuan Tai, "False Positives and Negatives from Real Traffic with Intrusion Detection/Prevention Systems," ICAIT (International Conference on Advanced Information Technology, Hong Kong, June 2012.

該篇論文的較完整版本已出現在 IEEE Communications Magazine – Network Testing Series，由於本人是該 Series 之 Editor，IEEE Communications Society 規定 Series Editor 不能將自己的論文放在自己的 Series 中，所以本人忍痛割愛將自己的名字從該論文中拿掉，並將有放本人名字的精簡版投稿至 ICAIT 會議。之所以會放我們的研究論文在這個 Series 上，是因為該 Series 需要設立幾個有代表性可參考的文章，讓投稿者知道要投稿至 Network Testing Series 應該具備如何的品質、內容與方向，與另一位 Series Editor UNH/IOL 的 Director Erica Johnson 討論後同意由 NBL 及 IOL 各貢獻一篇論文當做範例，但有了這次割愛的痛苦，僅此一次下不為例。至於 ICAIT 研討會本身是很普通的研討會，大部分文章的品質普通。

三、考察參觀活動(無是項活動者略)

因為被提名擔任 2013 年 IEEE Globecom Next-Generation Network Symposium 之 Co-Chair，因此也趁此會議之便拜訪同時被提名擔任 Co-Chair 的香港科技大學(Hong Kong University of Science and Technology) Prof. Qian Zhang，張教授之背景較特別，她是中國土產的學士、碩士、博士，都是在武漢大學，但因為在博士期間以及之後任職都在微軟亞洲研究院，所以有得到相當不錯的國際化洗禮，是極少數中國製造有國際競爭力的學者，這都是要歸功微軟的投資，我跟他談了許多共同認識的人、研究的方向與方式、香港科大的成功之道。在學校方面，香港科大的大學部學生有 8 成是香港本地生、1 成是中國內地生(Mainlanders)、1 成是國際生，但研究生就完全倒過來，香港本地生不到 1 成、國際生 1 成、中國內地生 8 成，這個邏輯是大學教育是地區性的責任，要照顧納稅人的子弟，研究就是全球性的議題，不用再照顧本地人，而是去收最好且願意來的人，顯然科大對中國內地生很有號召力，它可以輕易搶到北大清華的學生，因為科大才有國際化(1/3 外籍師資、全英文授課)，而科大大學部中的本地生根本無法跟中國內地生競爭念研究所，可以想見香港在擁抱全中國資源的同時，也大開門戶，讓香港本地跟中國內地廉價勞工與知識菁英都進來跟各階層競爭，因此好的越來越好，不夠好的就很苦了。

除了收到中國最好的學生到科大當研究生，張教授還跟中國頂尖大學合作研究，只是這個合作模式很有趣，表面上是跟中國教授合作，實質上是用他們優秀的學生做她想做的題目，而科大學生不夠多來做這些題目，因此跟中國教授只要一開始搭好關係，論文給他掛名就好，其他開會討論就跟研究生聯繫就好，科大等於在招生掃過一次後，進去插吸管再榨一次，這是非常聰明的作法，值得學習。

科大的師資約 1/3 是外籍、1/3 香港本地背景、1/3 中國內地背景，外籍師資的研究表現較乏善可陳，主要作為國際化的表象，內地背景師資比香港背景師資研究表現好，因為較有在香港外地生根的壓力、且有龐大中國內地資源可以運用，整體而言科大有 1/3 到 1/2 的教授具有相當優異的國際競爭力，這要歸功於成立才二十年的新學校有較多師資可以挑，也還沒開始出現包袱，另外就是要歸功以上總總的策略運用。

四、建議

1. 整合各大學資源吸收掛計畫的學生
2. 提供更好福利與資源聘請優秀學者,提升各大學師資

五、攜回資料名稱及內容

論文集光碟一片

六、其他

國科會補助專題研究計畫項下出席國際學術會議心得報告

日期：101_年 7 月 16 日

計畫編號	NSC 100－2218－E－009－024-		
計畫名稱	惡意程式偵測與樣本資料庫之建置與研究		
出國人員姓名	林盈達	服務機構及職稱	國立交通大學資工系教授
會議時間	101 年 7 月 9 日至 101 年 7 月 11 日	會議地點	義大利熱內亞
會議名稱	（中文）國際電腦與通訊系統效能分析會議 （英文） SPECTS (International Symposium on Performance Evaluation of Computer and Telecommunication Systems)		
發表論文題目	（中文）快速收斂的視窗平均流速控制方法 （英文）A Fast-Converging TCP-Equivalent Window-Averaging Rate Control Scheme		

一、參加會議經過

6/27 台北→米蘭
6/28-7/7 拜訪米蘭理工大學 Prof. Luigi Fratta
7/8-11 9AM-5PM SPECTS 研討會
7/12-14 義大利旅遊
7/15 米蘭→台北

二、與會心得

此次參與 SPECTS 2012 報告一篇論文:

Shih-Chiang Tsao, Yuan-Cheng Lai, Ying-Dar Lin, "A Fast-Converging TCP-Equivalent Window-Averaging Rate Control Scheme," SPECTS (International Symposium on Performance Evaluation of Computer and Telecommunication Systems), July 2012.

該篇論文獲頒該會議之最佳論文獎(Best Paper Award)，事實上該論文兩三年前就應發表，但在投稿期刊 Computer Networks 時被 Reviewer(韓國 KAIST 教授轉交其博士生 review)整篇 PDF 檔抄襲(僅改作者，且將該投稿以零分之評等 reject)投稿至另一會議與期刊發表，此事經本人已畢業之博士生(該論文第一作者)在網路上發現，本人對已發表之會議論文向 IEEE Explore 請求下架(IEEE 已將該文下架並貼出抄襲情事)，對已接受但尚未發表之期刊(Journal of Communications and Networks)論文則及時通知不予發表，並向 KAIST 校方及系上提出抗議，KAIST 原本想大事化小，但本人正告將向韓國媒體批露此事，KAIST 才將該博士生退學，對該教授扣月薪 10%。此時獲得最佳論文獎，也可算是這個 Society 對此研究遲來的公道，雖然審查者與主辦單位並不知道此事。本人也從此事獲得兩個經驗: 遇到不平一定要自己據理力爭，以及即使遭遇不平也要繼續努力鍥而不捨。

在該會議也認識幾位歐洲的學者，包括義大利、西班牙語美國的與會者，並共進午餐與咖啡，其中一位美國研究者更對本人論文提出許多問題，他雖然在工業界工作，但仍對學術研究的效能數學分析仍保持涉略，精神值得學習。

三、考察參觀活動(無是項活動者略)

在會議前本人花了約一周的時間訪問米蘭理工大學的 Prof. Luigi Fratta，同時間本人以前在 UCLA 指導教授 Prof. Mario Gerla 也回到義大利米蘭故鄉一齊訪問，我們一齊參加他們的研究會議、聽取研究生及博士後之報告，也共進數次午晚餐，兩位都是年約六十的學者，仍很規律積極的投入研究工作，值得當作模範。由四個米蘭大學 ECE 教授合作的實驗室之研究生與博士後共約 15 人，分別來自義大利、西班牙、東歐的捷克與波蘭，組成相當多元，Fratta 仍有自己的實驗室，所以他有一半的學生在個別實驗室一半在聯合實驗室，配置相當靈活。

該校有兩個做法很值得我們學習: (1) 數個教授聯合組成實驗室真的聯合運作，共同申請計畫、合作指導學生與共同發表論文，空間不夠時甚至一齊透過學校租借校外之空間，在台灣似乎較多是共同申請計畫的假性合作，(2) 該校研究所一律英文授課，大學部則大部分是義大利文授課，此舉是為了增加國際研究生之比率(已達 15%)，這個研究所全面英文授課的決定是來自校方，而非系所投票決定，在台灣我們似乎把治校不需民主的事也民主化了，導致國際化進程緩慢，國際生的比率還在 3-4%掙扎。

四、建議

1. 鼓勵教授們聯合組成實驗室聯合運作, 共同申請計畫、合作指導學生與共同發表論文, 激發更多

創意
2. 強制要求研究所全面英文授課,加快國際化的腳步

五、攜回資料名稱及內容

論文集光碟一片

六、其他

國科會補助計畫衍生研發成果推廣資料表

日期:2012/09/07

國科會補助計畫	計畫名稱: 惡意程式偵測與樣本資料庫之建置與研究
	計畫主持人: 林盈達
	計畫編號: 100-2218-E-009-024- 學門領域: 資訊安全

無研發成果推廣資料

100 年度專題研究計畫研究成果彙整表

計畫主持人：林盈達	計畫編號：100-2218-E-009-024-
計畫名稱：惡意程式偵測與樣本資料庫之建置與研究	

成果項目			量化			單位	備註（質化說明：如數個計畫共同成果、成果列為該期刊之封面故事...等）
			實際已達成數（被接受或已發表）	預期總達成數(含實際已達成數)	本計畫實際貢獻百分比		
國內	論文著作	期刊論文	0	0	100%	篇	
		研究報告/技術報告	0	0	100%		
		研討會論文	0	0	100%		
		專書	0	0	100%		
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（本國籍）	碩士生	0	0	100%	人次	
		博士生	0	0	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		
國外	論文著作	期刊論文	12	4	60%	篇	多年延續性計畫之成果
		研究報告/技術報告	0	0	100%		
		研討會論文	5	2	50%		有一篇獲得 SPECTS 2012 之最佳論文獎
		專書	1	0	20%	章/本	由 McGraw-Hill 出版之英文教科書，八章中有一章是網路安全
	專利	申請中件數	4	1	50%	件	
		已獲得件數	2	1	50%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（外國籍）	碩士生	3	3	100%	人次	
		博士生	1	1	100%		
		博士後研究員	1	1	100%		
		專任助理	1	1	100%		

其他成果 (無法以量化表達之成果如辦理學術活動、獲得獎項、重要國際合作、研究成果國際影響力及其他協助產業技術發展之具體效益事項等，請以文字敘述填列。)	1.獲得 SPECTS 2012 最佳論文獎。。 2.由 McGraw-Hill 出版英文教科書 3.接受 NCC 委託制訂資通訊產品安全認證規範，已公告 8 項產品規範，並建立符合 ISO/7025 之檢驗實驗室。 4.擔任九個期刊之總編，其中六個為 IEEE 期刊。

🟥	成果項目	量化	名稱或內容性質簡述
科教處計畫加填項目	測驗工具(含質性與量性)	0	
	課程/模組	0	
	電腦及網路系統或工具	0	
	教材	0	
	舉辦之活動/競賽	0	
	研討會/工作坊	0	
	電子報、網站	0	
	計畫成果推廣之參與（閱聽）人數	0	

國科會補助專題研究計畫成果報告自評表

請就研究內容與原計畫相符程度、達成預期目標情況、研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）、是否適合在學術期刊發表或申請專利、主要發現或其他有關價值等，作一綜合評估。

1. 請就研究內容與原計畫相符程度、達成預期目標情況作一綜合評估

   ■達成目標

   □未達成目標（請說明，以 100 字為限）

   　　□實驗失敗

   　　□因故實驗中斷

   　　□其他原因

   說明：

2. 研究成果在學術期刊發表或申請專利等情形：

   論文：□已發表 ■未發表之文稿 □撰寫中 □無

   專利：□已獲得 ■申請中 □無

   技轉：□已技轉 □洽談中 ■無

   其他：（以 100 字為限）

   研究成果已撰寫成論文，同時，亦送到國際學術期刊 IEEE Security & Privacy 及 Computer Communications 審查中.

   另外，本研究成果也已申請美國專利中。

3. 請依學術成就、技術創新、社會影響等方面，評估研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）（以 500 字為限）

   資訊與網路設備之研發及製造是我國資通訊產業的發展主流，國內自行研發或製造之資通安全產品種類眾多，運作模式與偵測技術更是五花八門，為了確保資通安全設備之防禦成效如其規格書所述，有必要針對各家資通安全產品，進行安全功能及防禦能力之檢測。

   有鑑於國內缺乏大型且具代表性之檢測資料庫供實機測試時使用，也由於 CVE 或 Wild List 等國外單位公佈之惡意流量或攻擊程式取得不易，本研究擬針對惡意程式樣本，發展出有效的偵測、分析及收集機制，用以廣泛收集樣本，最終建置台灣代表性之惡意程式樣本檢測資料庫。另外，由於無線通訊技術之發展，行動裝置已逐漸取代傳統有線之桌上型裝置，行動裝置的安全議題亦越來越受到重視與廣泛討論，未來安全規範檢測技術規範亦可能涵蓋行動通訊裝置，進行相關的安全測試。因此，本惡意程式檢測資料庫同時亦包含了部分行動裝置之安全威脅樣本。此一代表性之惡意程式檢測資料庫，不僅可供安全檢測技術規範中實機測試用，亦可提供相關安全技術之發展與研究使用.未來，更可透過此惡意程式資料庫來了解網路攻擊、病毒感染等發生頻率，並進一步佈署對應的防禦機制。