國立交通大學

多媒體工程研究所

碩士論文

惡意軟體之測試與攻擊方法探討

On the Study of Fuzzing and Attacking Malware Methods

研 究 生：吳介豪

指導教授：陳登吉　教授

黃世昆　教授

中 華 民 國 102 年 6 月

惡 意 軟 體 之 測 試 與 攻 擊 方 法 探 討

On the Study of Fuzzing and Attacking Malware Methods

研 究 生：吳介豪　　Student：Chieh-Hao Wu

指導教授：陳登吉　　Advisor：Deng-Jyi Chen

黃世昆　　Shin-Kun Huang

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2013

Hsinchu, Taiwan, Republic of China

中華民國一百零二年六月

惡意軟體之測試與攻擊方法探討

學生：吳介豪　　指導教授：陳登吉博士
黃世昆博士

國立交通大學多媒體工程研究所碩士班

摘要

惡意軟體在網際網路大量流竄，造成大量電腦系統的損壞，更因為殭屍網路(Botnet)的盛行而產生實體產業如金融業的災害。雖然許多測試工具已能協助找出一般軟體的漏洞，以提升軟體品質與穩定性，而惡意軟體雖然同屬軟體的一員，但由於其中常使用加密演算法來保護內部的資訊與外部伺服器的溝通，因此一般測試工具很難有效地對惡意軟體進行測試。對於惡意軟體研究大致可以分為兩種，第一是惡意軟體的惡意行為分析，第二是分析惡意軟體的弱點，現今多數研究為前者。

因此本研究將探討對惡意軟體中使用的加密演算法進行處理的文獻，以及目前對惡意軟體的測試、分析方法，了解找尋漏洞的過程，最後蒐集已被發現漏洞的惡意軟體，分析這些漏洞發生的原因和相關資訊，以提供進一步的研究資料。

關鍵字：惡意軟體、殭屍網路、漏洞、加密演算法、測試工具

On the Study of Fuzzing and Attacking Malware Methods

Student：Chieh-Hao Wu

Advisor：Dr. Deng-Jyi Chen
Dr. Shin-Kun Huang

Department of Multimedia Engineering

National Chiao Tung University

Abstract

Malware has been spread in the network for a long time and damaged many application systems. Many commercial activities like the financial sector have been affected by the botnet in the real world. Although many fuzzing tools have been used to find related vulnerabilities in the software to improve the quality and reliability, but the malware usually implements the encryption functions to protect inner information and the message communicating with C&C servers. A general fuzzing tool can't test malware with satisfactory results. The research for malware can be divided into two types, (1) the analysis of malware's malicious behavior and (2) the analysis of malware's vulnerability. Nowadays the former research is paid much more attention than the latter.

Therefore, this thesis will study how we can identify the encryption functions in the malware, and attack and analyze malware with general methods. We have collected and organized the vulnerabilities of various kinds of malware, with possible attacks.

Keywords: Malware, Botnet, Vulnerability, Fuzzing tools, Encryption function.

致謝

光陰似箭，時光飛梭，碩士兩年的生活很快的就結束了，雖然這段期間只專注在自己的課業上，並沒有參與太多的活動，對交大的許多事情仍然不夠了解，不過只要身為交大人那就夠了。

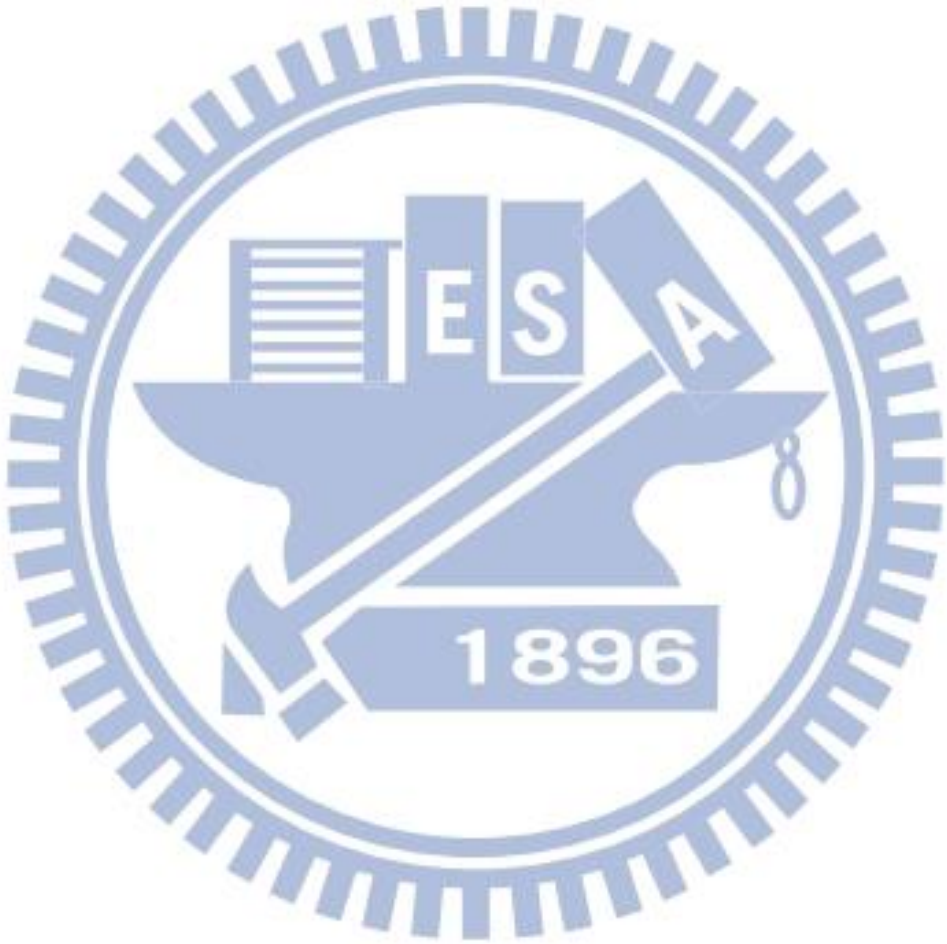
在交大的這些日子受到許多人的幫助。首先是已經去世的指導教授的陳登吉老師，在一年級時陳老師便不斷的教導我們無論是課業或是生活上的知識，直到最後陳老師仍然心繫著我們的畢業論文，現在沒有辜負陳老師的期望順利的完成論文，謝謝陳老師一年來的指導以及鼓勵。接著謝謝在二年級時接納我的黃世昆老師，無論是進度落後或是中途遇到困難，黃老師仍然細心的指引我方向以及給予建議，最後才能完成畢業的論文。也同樣感謝孔崇旭老師時常給予不少的意見，讓我受益良多。感謝實驗室的學長們以及同學，由於我對這個領域的不熟悉，在他們的講解下才能更快的了解這方面的內容。最後感謝家人一直以來在背後的支持以及鼓勵。

沒有各個老師、同學以及家人的支持，憑我自己或許會花費更多的時間才能完成，所以感謝所有人。

目錄

表目錄

圖目錄



一、 緒論

1.1 研究背景

在軟體工程流程中，軟體測試[1]的目的為找出軟體中可能存在的弱點、漏洞，以避免軟體品質的下降。而有心人士也可能藉由這些漏洞製造出非程式本身以外的功能，例如執行病毒程式、修改電腦設定等等。

軟體測試的方法有很多種，其中 Fuzzing[2, 3]是一種自動產生測試資料的技術，Fuzzing 能夠根據受測程式需求產生非法的測試資料去測試程式，檢查是否有漏洞存在。而 Fuzzing 有幾個簡單的步驟：

1. 根據測試程式所需產生測試資料，例如某部分程式需要輸入數字，則會產生溢位的數值或是字串作為輸入的資料。
2. 將產生的測試資料送給測試程式執行。
3. 由 2，若是測試程式崩潰，則將此筆測試資料記錄下來以做後續分析，若沒有則直接回到第一個步驟。

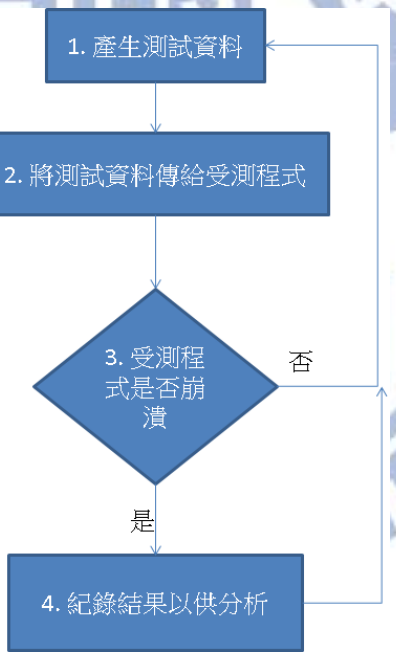


圖 1. Fuzzing 簡單流程

但是實際上 Fuzzing 時會遇到一些困難，加密演算法(Encryption algorithm)時常運用在軟體當中，以保護內部的資訊，最常見的為一般的壓縮軟體、網路傳輸的 Checksum，這些都會造成 Fuzzing 時需要額外耗費多餘的成本去處理。而加密演算法也有其對應的解密演算法(Decryption algorithm)，有些系統[4, 5]便是使用解密演算法來加快處理這些問題。

惡意軟體(Malware, Malicious Software)現今常出沒在各個地方，主要為竊取使用者的私密資料，例如銀行、信用卡資訊以及帳號密碼等等[6]，或是造成系統崩潰損壞。而常見的惡意軟體除了一般電腦病毒之外，還有 bots、spyware、worm 以及木馬等等，而像是殭屍網路(Botnet)[6-8]是由潛伏在系統中的 Bots 與控制這些 Bots 的 C&C (Command and Control)Server 所組成，由於被 Botnet 所掌握的系統的規模都相當龐大，因此後續章節會將 Botnet 作為主要的惡意軟體目標進行討論。

1.2 研究動機

由於軟體內可能會有使程式崩潰的漏洞，為了穩定以及可靠性，軟體測試是非常重要的一環。同樣也是軟體的惡意軟體，當然也可能存在著漏洞，而惡意軟體的作用是對目標產生攻擊性的行為，比起一般軟體是否更不注重安全性，更容易存在著漏洞。

目前已經有許多對惡意軟體的研究，其中分為以防禦、分析角度[9, 10] [11] [12-15]以及測試、攻擊角度。前者像是防毒軟體等等，觀察軟體是否具有攻擊性的行為來辨識是否為惡意軟體，而後者則是藉由測試的技術，例如 Fuzzing[5, 16, 17]，去測試這個軟體是否具有弱點。

圖 2. 惡意軟體的惡意行為分析 VS 惡意軟體的弱點分析

而像是 botnet 這類的惡意軟體，它們掌握的受感染電腦規模、災情都非常的龐大[6, 18-21]，因此許多組織機關都持續的發佈某個 botnet 的消息，包括其類型、災情、規模以及如何檢測有無被感染的解決方法，但是 botnet 變種成長的速度依舊讓整個網路有著隱藏的威脅。

1.3 研究目的

雖然 Fuzzing 已經廣泛應用在許多軟體上面，協助找出這些軟體的漏洞[22, 23]，由於惡意軟體中常使用加密演算法來保護內部資訊以及與 C&C server 溝通

的訊息，所以在惡意軟體的測試上若是沒有特別針對加密演算法做處理，則整個過程會變得非常耗時。一些文獻[4, 5, 16, 24]也以辨識、解決在測試惡意軟體時遇到加密演算法做為其系統目的，作為分析測試惡意軟體的首要步驟，因此了解如何處理加密演算法為對惡意軟體做測試的重要議題之一。

惡意軟體有許多不同的類型，其中 botnet 的所造成的危害非常的大，再加上被 botnet 所掌控的系統數量都非常的多[18, 25]，因此 botnet 為本研究主要針對的惡意軟體，在得知惡意軟體的弱點後，可以藉由這些弱點取得我們所感興趣的資訊，例如說這個 botnet 的規模，影響範圍以及受感染的系統資訊等等。

本研究目的著重在以下四點：

1. 惡意軟體中加密演算法的分析以及辨認
   因為加密演算法會將內部資訊做加密，讓 fuzzing tools 無法直接產生符合的資料格式，在探討相關文章後，整理如何辨認目標軟體中所使用的加密演算法並做一個比較。
2. 對惡意軟體進行測試以及攻擊的方法案例
   針對 botnet 去探討攻擊的方式，以及介紹幾個測試以及攻擊的方法案例，最後再做一個比較。
3. 整理屬於惡意軟體的漏洞資訊
   本研究會收集已知屬於惡意軟體的漏洞，分析發生的原因、型態，並整理一些較常見的 botnet 資訊作為之後研究方向的參考。
4. 使用 CRAX 產生弱點以及 exploit
   利用實驗室所製作的測試工具，嘗試對第三點所收集的幾個惡意軟體進行測試以重現其弱點，以及能否找出新的漏洞。

1.4 章節介紹

第一章節說明本研究的背景、動機以及目的

第二章節會介紹本研究主要探討的 botnet 以及針對惡意軟體內部加密演算法處理的相關研究以及工具。

第三章節會探討一些如何攻擊以及分析惡意軟體的案例。

第四章節為對屬於惡意軟體的漏洞進行分析以及整理，還有使用實驗室的工具進行的實驗結果。

第五章節為本研究的結論以及未來展望。

二、 文獻探討、相關研究

在 1.2 的圖 2 提到針對惡意軟體的相關研究可分為兩個角度，第一為惡意軟體的惡意行為分析，目前許多研究都是偏於這類，例如[9, 26] [11] [12, 15]，不過本研究主要針對的為第二角度所提的分析惡意軟體的弱點，而無論是一般軟體或是惡意軟體，它們內部可能都會用一些加密演算法來保護內部資訊，因此針對惡意軟體內所使用的加密演算法的分析以及處理也是需要探討的內容之一。

2.1 Botnet 架構

在探討攻擊惡意軟體的方式前，首先要先了解常見的惡意軟體架構，像是殭屍網路(Botnet)[27-29]，它一般是由命令以及控制伺服器(Command & Control Server, C&C Server)搭配潛藏於受感染主機中的 Bot 所組成，簡易的運作模式如圖 3，botnet 的擁有者會透過 C&C server 所提供的介面來監視所有的 bots，或是對它們發送惡意的命令。

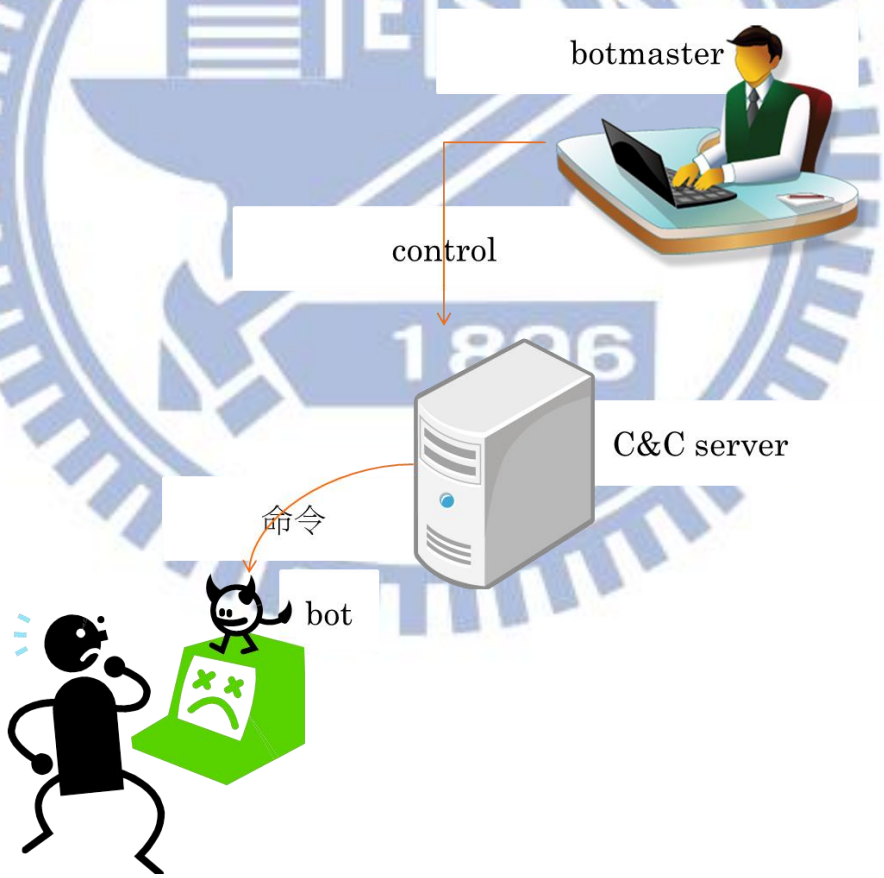


圖 3. Botnet 基本運作模式

而一般 botnet 可以分為以下三種架構：

1. Centralized

   由一個 botnet C&C server 與所有 victim 溝通，好處是能夠直接的掌控所有 bot 所在的主機以及能夠同步的發送命令，但是缺點為這個 C&C server 若是發生問題則整個 botnet 可能就會直接瓦解。

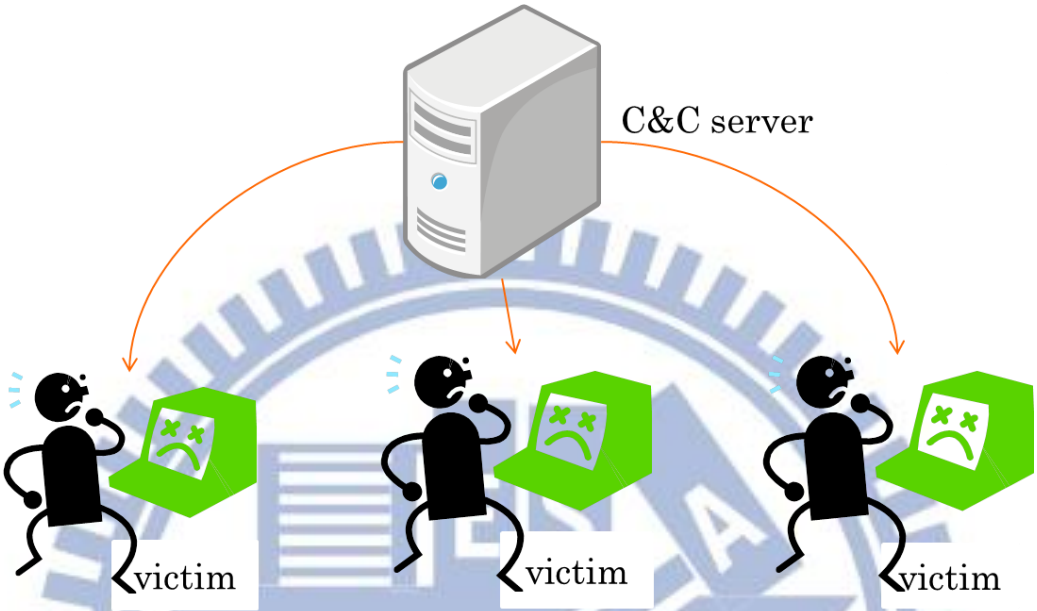


圖 4. Centralized 類型 botnet

2. Decentralized

像是 peer-to-peer 架構，有多個 C&C server 能互相溝通，比起 Centralized，多個 C&C server 能夠避免壞一台則整個 botnet 瓦解的狀況，但是缺點為命令沒辦法直接同步到所有的 bot 上。

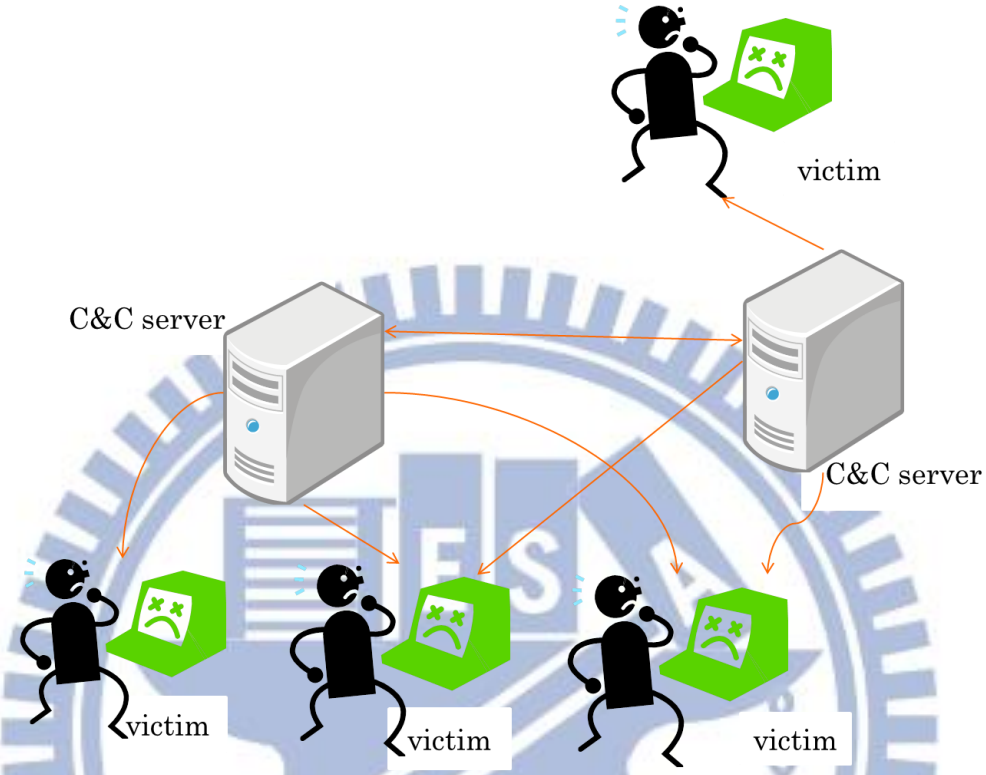


圖 5. Decentralized 類型 botnet

3. Locomotive

比起上述兩種更加複雜，除了本身可能為上述兩種架構之外，domain name 可能會隨著時間改變，例如 Torpig botnet[30]、Conficker[31]。而這種類型也沒有一個固定的架構型態。

2.2 Malware Fuzzer

在了解 botnet 的架構後，由於惡意軟體中時常使用加密演算法來保護內部的資訊而不容易被分析，接著要介紹一些與惡意軟體內相關的解加密演算法的文獻。Fuzzing 是一種軟體測試的技術，能夠自動產生測試資料對目標程式進行測試，而使用這種技術的工具便稱為 fuzzing tools 或是 fuzzer。本論文收集了兩篇相關的文章，一篇為專對惡意軟體所實作的系統，另一篇則是專對一般軟體中使用 Checksum 加密的系統。

2.2.1 BitFuzz

BitFuzz[5]為一個以 BitBlaze[32, 33]為框架的 malware fuzzer，這系統使用了 Dynamic Symbolic Execution[34]技術追蹤程式流程，加上為了辨認惡意軟體內部的加密演算法而可以重複的產生不同測試資料所提出 Stitching 技術，主要想法分為以下三點：

1. 1. Identify the Potential Encoding Function(加密演算法)
   在追蹤程式流程時，藉由 Path Constraint 系統會找出其中的加密演算法以及其反函數。

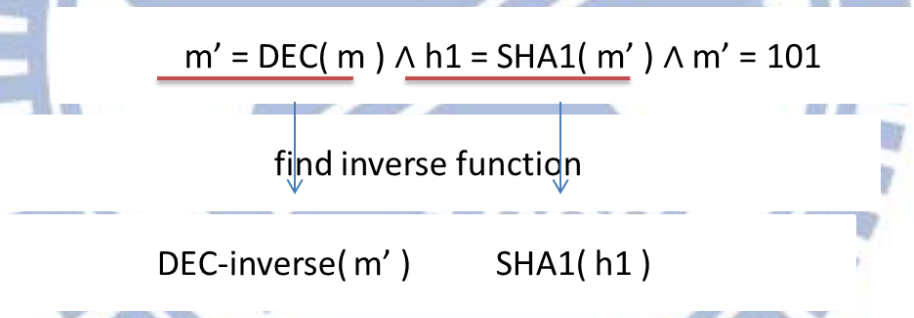


圖 6. BitFuzz 找出 path constraint 中的加密演算法

2. Decompose the Path Constraint
   在找出使用了哪一些加密演算法後，由於要解這一些演算法較為困難，因此系統會從 Path Constraint 中選出與加密演算法無關的部分，並且著手去解這一些地方。

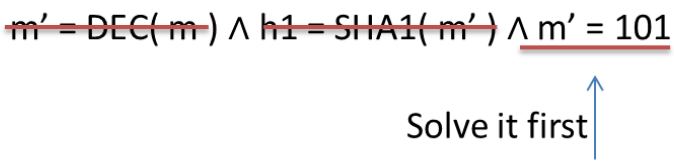


圖 7. BitFuzz 解 path constraint 中較為容易的部分

3. Re-stitching

在解出與加密演算法無關的部分之後，若是這些值與加密演算法有關連，則最後系統會將這些解出的值，藉由各個加密演算法的反函數取得各種不同的輸入值。

$$m' = X$$
$$\rightarrow m = \text{DEC-inverse}(X)$$

圖 8. BitFuzz 使用 Re-stitching 經由不同 X 解出不同的輸入值

如此一來 BitFuzz 能夠藉由解出較簡易的部分，搭配解密演算法來加速整個流程，當然若是遇到無法找出解密演算法，或是 Path Constraint 沒有較簡易部分可以利用，那麼整個程式仍然要耗費許多時間。

2.2.2 TaintScope

TaintScope[16]同樣為一個 Fuzzer，它主要針對的是使用 Checksum 做加密的軟體。Checksum 常用於網路傳輸的資料上，雖然此篇文章實作的系統只以一般軟體作為目標，不過也可以運用在同樣為軟體一員的惡意軟體上面。

有些 Fuzzer 產生測試資料時是隨意的更動已知的正常輸入資料，但是這樣並不能保證每筆測試資料都是有用的，可能很早就被驗證的機制給阻擋下來，例如先前所提到的 Checksum，若是任意更動數值，那麼 checksum 的檢查機制會立即判斷出來。因此為了不重複製造出沒有價值的測試資料，TaintScope 主要的核心技術為只修改測試資料中較重要的部分。

TaintScope 主要專注於改動測試資料中的兩個部分，如圖：

1. Hot Bytes
   由於 TaintScope 有興趣的是 API 中的參數部分，因此他們將與 API 有接觸的資料部分稱為 hot bytes。
2. Checksum information
   第二則是與 jump instruction 有關，也就是可能為判斷 checksum 的地方。

藉由這兩個部分 TaintScope 能夠快速的產生有效的測試資料。

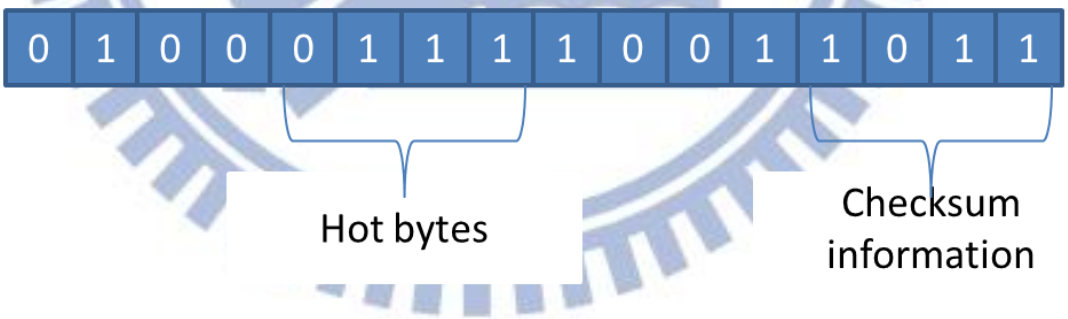


圖 9. Hot bytes and checksum information

2.3 2.3 Malware specific function identification and handling

許多軟體撰寫時，常使用加密演算法來保護內部的資訊，尤其是會與外界溝通、傳輸資料的軟體，例如網路的封包、資料壓縮等等，而惡意軟體為了規避防毒軟體的檢查，像是 Bots 同樣也用了加密演算法來保護與 C&C server 溝通的訊息。因此這邊將列出一些針對惡意軟體中加密演算法的保護機制，進行處理的文獻。

在對加密演算法作處理之前，系統必須要能夠辨識出目標程式使用了哪種加密演算法，這一小節要介紹三篇[4, 24, 35]能夠自動的辨識出惡意軟體所使用的加密演算法，以及重要的相關參數。

2.3.1 Identification Cryptographic Primitives

第一篇所實作的系統，主要分為兩部分，第一追蹤整個程式的流程，將其所有的指令以及使用到的資料還有狀態都記錄下來，這部份他們使用 DBI(dynamic binary instrumentation)技術。第二部分使用 Heuristic 去辨識前一部分所收集的資訊，最後輸出使用那些加密演算法，以及相關的參數資訊。

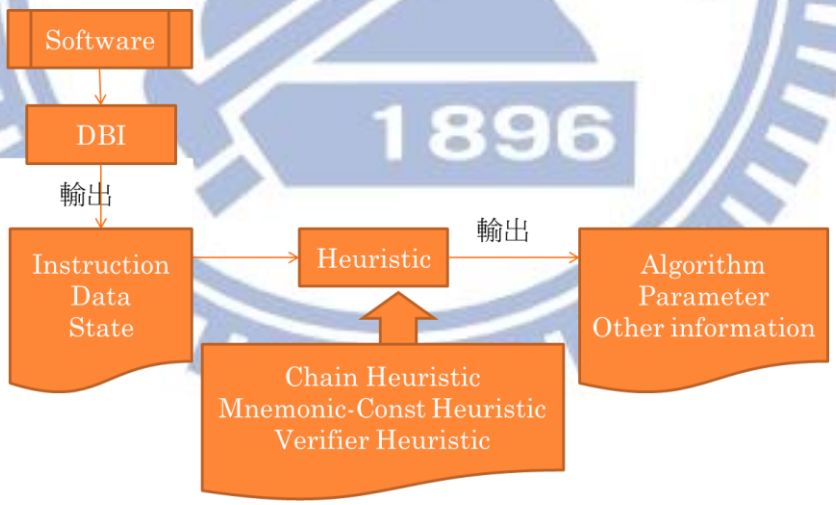


圖 10. 主要流程

2.3.2 Aligot

第二篇則是實作了一個 Aligot[35]系統，它主要想法是以 I/O parameter 來判斷屬於哪種加密演算法，也就是說如果有個 Input 以及 Output，假設它是使用演算法 F，那麼要是 Output = F(Input)就能得知所使用的加密演算法即為 F。而與上篇類似，Aligot 也是動態收集 Execution Trace，將這些 trace 以 Loop Data Flow 表示，接著以 I/O parameters 來與事先所準備好的演算法做比較，它是選擇 Tiny Encryption、RC4 algorithm、Advanced Encryption Standard、MD5 algorithm 以及 RSA。主要流程如圖 13。

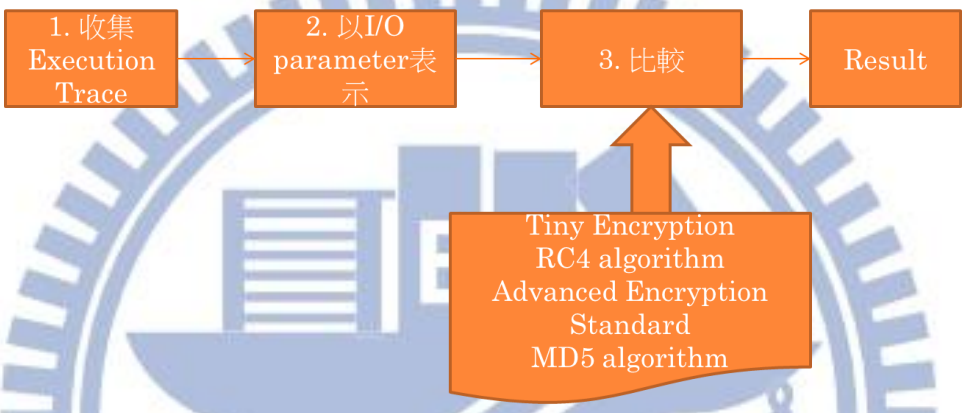


圖 11. Aligot 主要流程

2.3.3 CIS

CIS[4]也是收集了的程式執行流程，利用不同指令的組合代表使用不同加密演算法來判斷。而他們在挑選這些 heuristic 時則是從總計八種中選擇了四種來做為系統的 heuristic function。

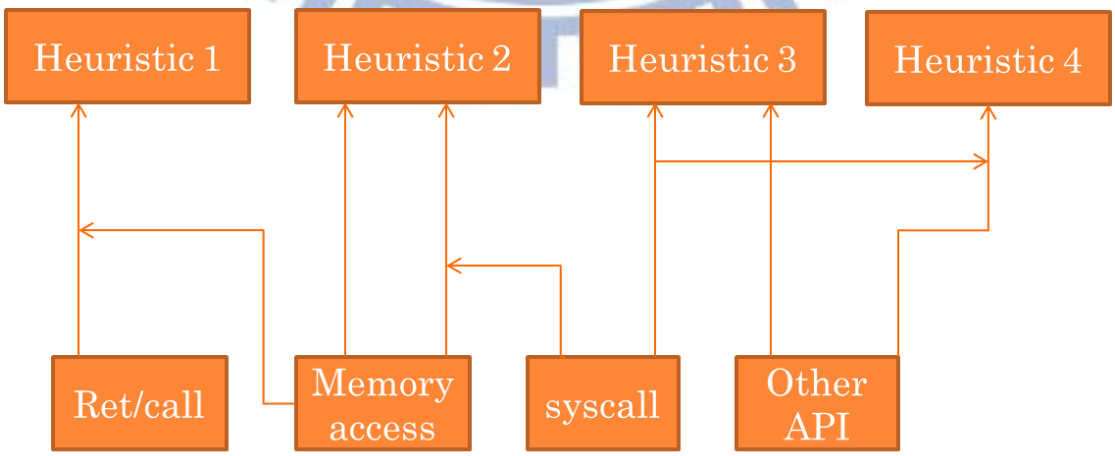


圖 12. CIS 判斷的簡易架構圖

接著比較以上所介紹的系統中，他們所使用的辨識方法、用了哪些 Heruistic 以及能夠辨認出那些加密演算法。

由表所示，這三個系統都使用了三至四種的 heuristic 來判斷，而辨識的內容主要以 instruction 以及 input-output 為主。最後能夠辨認出來的加密演算法也不完全相同。

表 1. 各個系統針對加密演算法辨認的分析比較

探討的系統	主要辨識方法	所使用的 Heuristic 數量	測試結果能辨認出來的加密演算法
Identification Cryptographic Primitives	Instruction (instruction, const) input-out	3	AES DES MD5 RC4 RSA
Aligot	Input-output	4	TEA RTEA RC4 AES MD5
CIS	instruction	4	(程式) Curl HTTPS Aescrypt File Encrypter Aphex Crypter AES File Crypter

三、 惡意軟體攻擊方法分析探討

3.1 攻擊 botnet 的方法

在第二章節中介紹了基本的 botnet 架構，接著要探討如何去攻擊，在[27]中則提出了三個對策：

1. The command and control (C&C) server
   以 C&C server 為目標，直接去關閉他，這樣一來 bots 因為接收不到命令而不會有任何行動產生。
2. The botnet traffic
   將從 C&C server 傳過來的命令導入到 sinkholes 中，除了能夠分析這些命令格式、內容之外，也能避免 bots 接收命令而開始行動。
3. The infected computers
   直接將受感染的系統清除。

而本論文目標是攻擊惡意軟體，所以只針對第一種方法做討論，而他們也提出了一些限制：

1. Botnet 的架構必須為 centralized。
   目前許多 C&C server 都使用非 centralized 的架構，像是 peer-to-peer，就算只關閉一個 server 仍有其他 server 可以運作。
2. 必須事先知道 C&C server 在哪。
   這部分可以由先攻擊或是追蹤 bot 內部[36]來取得 server 的位置。
3. the provider cooperates
   這裡的 provider 指的是 host provider。通常 botnets 的 C&C server 會架設在一些難以被追蹤的 hoster 上，因此需要這些 host provider 幫忙。

但是由於一些法令的緣故，並沒有辦法直接對這些 botnet 做出行動，例如直接對其攻擊等等，因此主要的目的以找出弱點，再藉此弱點對這些惡意軟體進行監控，來瞭解惡意軟體的行動以及資訊。

3.2 對惡意軟體做攻擊的方法案例

接著這邊要介紹幾種一般對惡意軟體做攻擊的方法，目前大多以追蹤程式碼的方式觀察內部是否有漏洞，以及找出較感興趣的資訊，分別以 Fuzz bots、Analyze Malware、Attack C&C server 以及 Attack RAT 做介紹。

3.2.1 Fuzz Bots

首先是使用 Fuzzing 來對惡意軟體做測試的系統，在第二章節提到 BitFuzz 為 malware fuzzer，它以 Stitching 技術解決了加密演算法造成產生測試資料的困難，並藉此不斷產生測試資料對幾個惡意軟體進行攻擊，成功的找出了幾個漏洞，接著簡單介紹 BitFuzz 的流程。

他們所用來測試的四個惡意軟體皆為 bots，分別是 Zbot、MegaD、Gheg 以及 Cutwail，由於 bots 會與 C&C server 建立連線來傳遞訊息，因此他們架設了一個假的 DNS server 以及 replay server，負責與 bots 做溝通並發送測試資料。在 bots 開始嘗試與 C&C server 做溝通時，一開始利用 Dynamic symbolic execution 收集程式流程(圖 13)，然後使用 Stitching 技術不斷的解出不同的測試資料並送回給這些 bots，並且觀察是否有崩潰的情形發生，圖 14、15 分別代表兩次不同的測試。

$$m' = DEC(m) \wedge h1 = SHA1(m') \wedge m' = 101$$

圖 13. path constraints

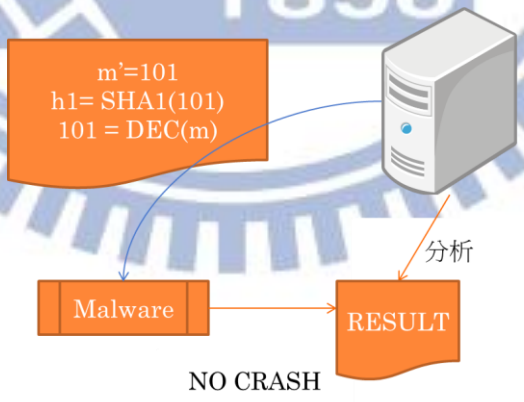


圖 14. BitFuzz 第一次 fuzzing

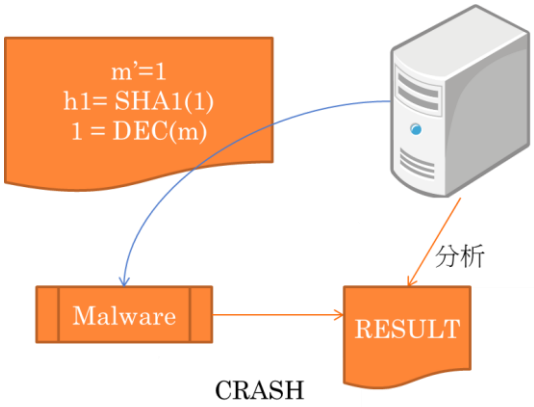


圖 15. BitFuzz 第二次 fuzzing

使用 Fuzzing 技術的好處在於能夠自動的重複整個測試步驟，但是前提為必須先了解如何與 bot 做溝通才能開始，而且還得先建立起整個測試平台，包括與 bot 溝通的 server 等等，這都是要個別去設定的，因此前置的動作還是得花上不少時間。

另外 BitFuzz 中也提及了如果 bot 中的加密演算法有被修改過用來混淆分析過程，那麼可能會產生出錯誤的資訊，而拖慢測試的時間，這部分與前一段相同，如果有這種情況仍然需要手動的去檢查。

3.2.2 Analyze Bot and Attack C&C Server

接著是藉由 DisAssembler 相關工具直接去追蹤程式碼，這種方式能夠了解程式內部做了什麼事情，例如 malware-lu[36]是針對 HerpesNet botnet 的 bot 進行追蹤。在追蹤程式碼過程中發現了這個 bot 做了一些動作，像是為了確保開機時會執行這程式，每 100ms 就會去設定 register key，以及他們感興趣的 C&C server 的資訊，這個惡意軟體每 15 秒就會與 C&C server 溝通一次。

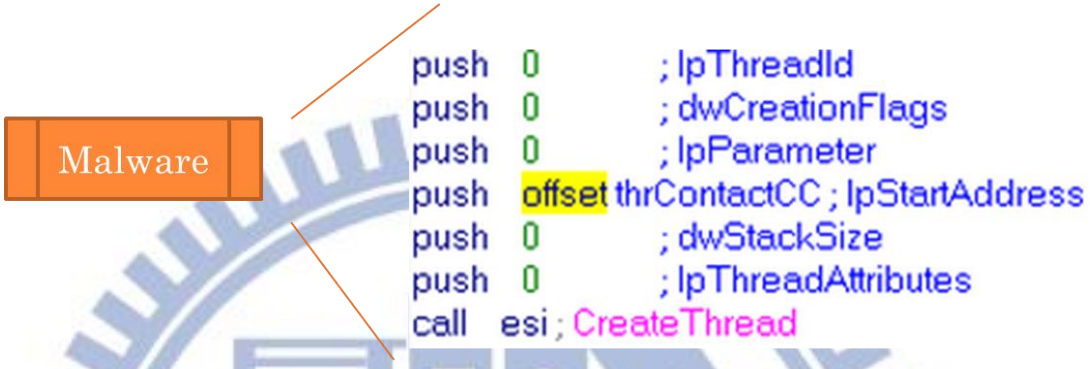


圖 16. 惡意軟體的 Assembly code

接著他們試著了解這個 bot 與 C&C server 溝通的訊息格式，雖然訊息是有被加密過的，但是經由直接觀察 assembly code 能夠得知加密方法而可以直接找出解碼方法，在解出訊息內容之後便得知了這個 C&C server 的位置以及相關資訊。

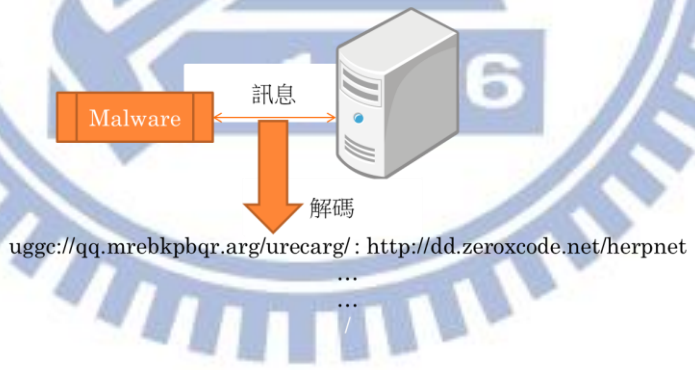


圖 17. 對溝通的訊息做解碼

最後他們使用 sqlmap 送出自己重組的訊息給 C&C server，嘗試製造出 SQL injection 的漏洞，也成功產生了 exploit。藉此 exploit 他們由這個 botnet 中取得了這個 botnet 的相關檔案以及資訊，還有 botnet 擁有者的個人資訊，最後讓這個擁有者關閉了它的 botnet。

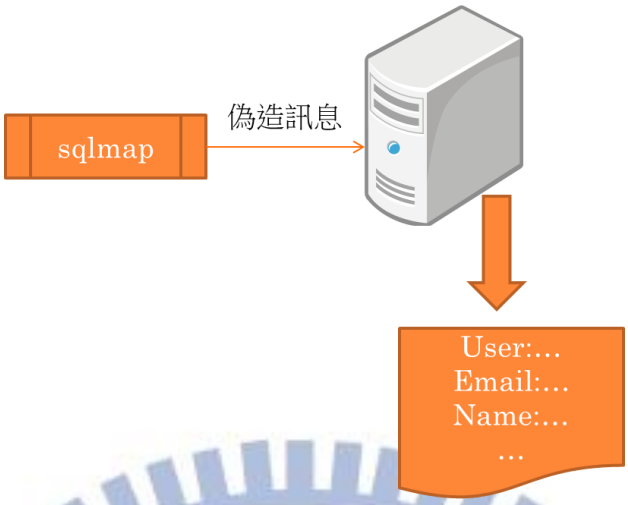


圖 18. SQL injection

在此案例中最為困難的地方，在於必須追蹤惡意軟體的 assembly code，從中了解這個 bot 所使用的加密演算法、與 server 溝通的訊息格式以及位置，這是十分耗費時間以及心力的，而且此案例的目標是背後的 C&C server，並不是這個 bot 本身，因此這些只是前置動作而已，接下來對 C&C server 送訊息才是主要的目的。

但是作者並沒有提及這個 C&C server 的漏洞在哪，而是直接嘗試送偽造的訊息來嘗試產生 SQL injection，當然最後是成功的產生漏洞以及 exploit。

3.2.3 Attack Botnet C&C Server

至於在 Web 部分，由於 botnet 的原始檔案很容易地在網路上取得，因此大部分研究都是直接追蹤原始碼來找是否有弱點存在，像是[37, 38]這兩個案例，兩者都是以 Zeus Botnet C&C server 作為目標，這邊則選後者作為例子。

在[38]中，作者直接由 C&C server 的 PHP 原始檔案去檢查，找到一個可能有漏洞的地方。在 zeus 的 bot 與 C&C server 溝通時，是經由 gate.php 這個頁面，而溝通的內容包括了能夠上傳 log file，而作者發現 C&C server 雖然有特別檢查這個上傳的檔案，但是做得並不完整。

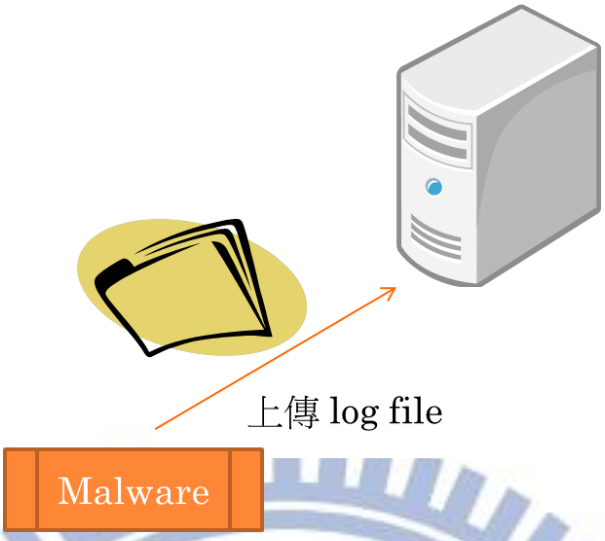


圖 19. Bot 會將 log 檔上傳至 C&C server

但是這個 C&C server 並沒有對 php extension 做特別處理，因此作者能夠自由的上傳自定義的 php 檔案 Server 所在的位置，這個 php 檔包含了讀入 C&C server 的設定檔，並取得一些資料例如 database 的帳號密碼等等，接著在 php interpreter 去處理這個網頁檔時便能執行以上功能，藉此摧毀掉這個 botnet。

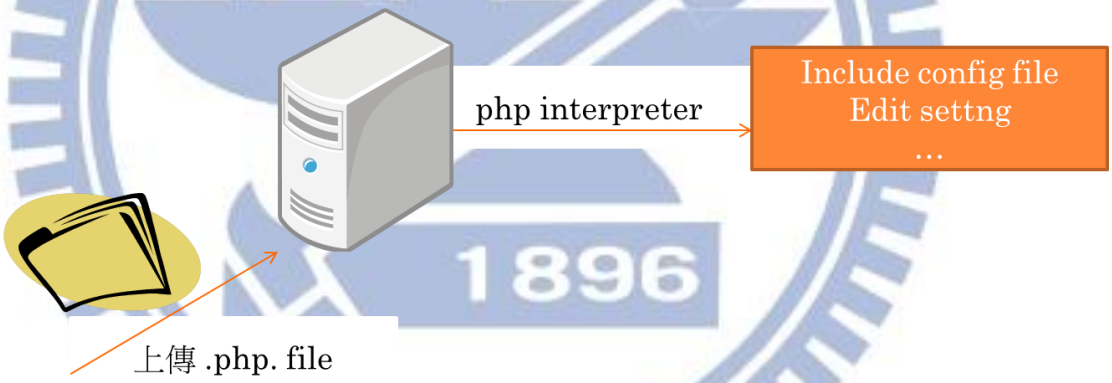


圖 20. 上傳自定義的 php file 來執行想要的動作

比起一般的程式，網頁是相對容易使用追蹤的方式來找尋弱點，在這案例中所找到的是 code injection 的漏洞，作者藉此漏洞上傳自己撰寫的 php 檔案，經由 php interpreter 去執行指令，除了能夠直接摧毀這個 botnet 之外，也能從旁監控這個 botnet 整個活動的訊息。

3.2.4 Attack RAT

3.2.3 所提的為一般的 web server，接著這裡要介紹的為 RAT(Remote Administrator Tools)，這類的軟體架構與 botnet 相同，都是由 C&C server 與 bots 所組成，只是 RAT 為視窗程式。

RAT 會開啟一個 port 與外界溝通，在事先了解這個軟體是以 Camelia 做加密之後，作者先以 Wireshark 追蹤程式傳遞訊息的大小，以及 Client 會傳送加密訊息，而這訊息代表了 Client 的資訊。

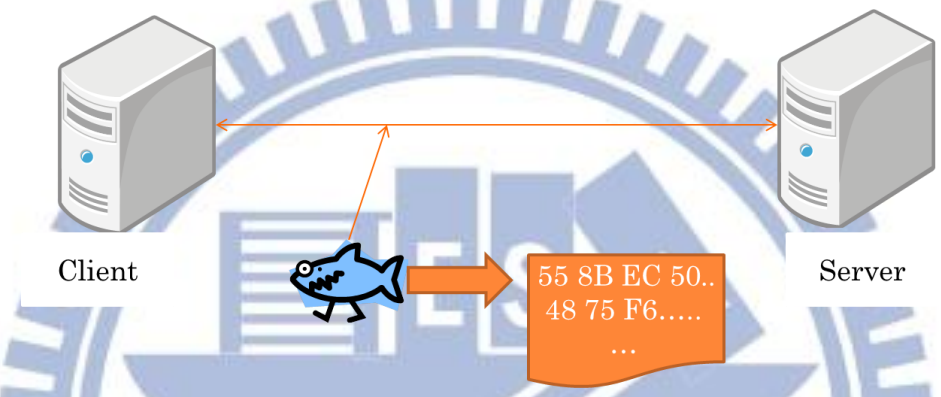


圖 21. 使用 Wireshark 監控之間的傳遞訊息

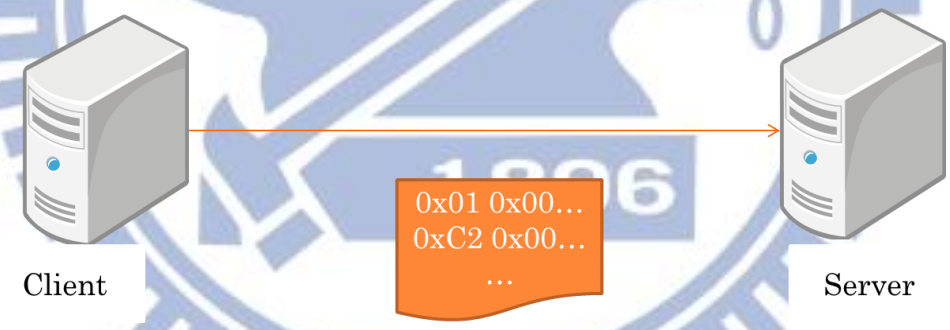


圖 22. Client 傳送加密的訊息至 server，這訊息代表了 client 的資訊

在得到這個訊息時，server 會開啟一個 thread 去處理，而這個建立 thread 的函數可能有 overflow 的弱點，因此最後在重組訊息並且嘗試加入 exploit 之後，建立連線並直接發送這個訊息，成功的產生攻擊。

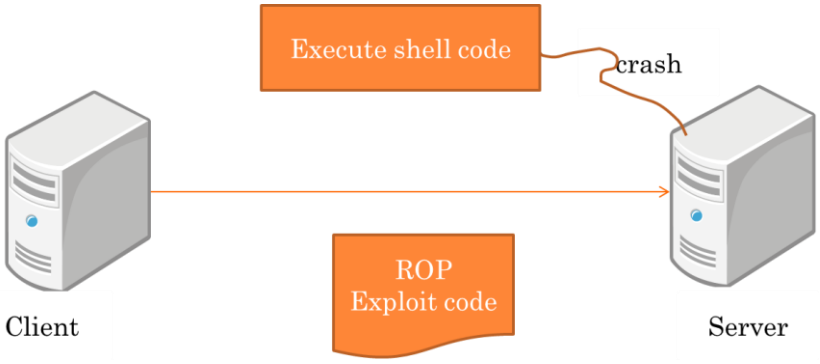


圖 23. Attack RAT

與第二個案例相同，都是直接追蹤程式的 assembly code，同樣也是非常耗費心力以及時間，雖然作者指出其中有個漏洞，但是並沒有詳細說明為何會有這個漏洞。另外這個攻擊是對 Poison Ivy 預設的 port 進行，若是他的 port 有更動那麼就必須事先再得知所使用的 port 才行。

3.3 攻擊惡意軟體的方法案例比較

在 3.2 中介紹了目前幾個攻擊惡意軟體的方式，3.2.1 是使用 fuzzing 來自動產生測試資料，而他們的目標為 bots。3.2.2 利用像是 DisAssembler 工具來追蹤 bots 的程式流程，藉此取得其 C&C server 位置，接著嘗試產生 SQL injection 等等的攻擊。與 3.2.2 不同，3.2.3 是以得知 C&C server 的原始碼為前提直接進行追蹤，觀察可能會有 XSS 或是 SQL injection 漏洞的部分，接著再嘗試做出攻擊。3.2.4 也是類似 3.2.2 作法，只是目標是視窗程式。

表 2. 攻擊惡意軟體案例比較

案例	針對的惡意軟體	弱點型態	自動化
Fuzz Bots	四種 Bots	Overflow Infinite loop Null reference	O
Analyze Bot and Attack C&C server	HerpesNet botnet	SQL injection	X
Attack C&C server	ZeuS botnet	Code injection	X
Attack RAT	Poison Ivy	overflow	X

以結果上來看，以上四種方法都有成功的找出漏洞，而這些漏洞可以用來了解這些惡意軟體運作的情況，例如存在於 C&C server 的 SQL injection，能夠藉此得知整個 botnet 的規模以及危害的系統資訊，code injection 以及其餘類型的弱點則可以藉著 exploit 來執行我們想要的操作，像是植入一些軟體來觀察這些系統的動作。

但是除了 Fuzz bots 的案例之外，其餘三種都是以人工方式來尋找漏洞可能的位置，很顯然的這個方式非常耗時，在 Analyze Bot and Attack C&C server 跟 Attack RAT 的案例中，兩者都是必須追蹤惡意軟體的 Assembly code，必須花費大量的時間去研究那些指令組合是在做什麼事情，接著再去找尋有無漏洞的發生，至於網頁介面的 C&C server 比起上述較為簡單，但仍需要耗費相當多的心力。

因此 Fuzz bots 以 Fuzzing 的技術取代人工檢查的方法，但是也有其缺點，由第二章所提到的加密演算法，BitFuzz 有提及若是惡意軟體中所使用的加密演算法為了混淆檢測的工具，而重新編寫演算法內容導致檢測工具產生錯誤的測試資料，那麼他們就不能順利的進行測試的流程。而手動追蹤程式碼就比較沒有此疑慮，能夠直接了解內部運作的流程進行後續的動作。

四、 惡意軟體弱點型態整理與探討結果

在探討完一些針對惡意軟體做測試以及內部加密演算法的文獻之後，接著收集已知弱點的惡意軟體，並對其做歸納整理。

目前已經有許多網站收集各個軟體的弱點，例如 Open Source Vulnerability Database[39]、National Vulnerability Database[40]提供給其他人分析，其中也有屬於惡意軟體的弱點。Exploit Database[30]中同樣也有收錄一些惡意軟體的 exploit。而 Open Malware[41]、Malware Domain List[42]則分別存有許多惡意軟體的樣本以及網站。至於一些文獻、研究中也有提及惡意軟體的漏洞[5, 6, 36-38]，因此本研究先從這些地方整理三十五個弱點，分別屬於十七種不同類型的惡意軟體以進行分析。

為了瞭解不同種類的惡意軟體通常擁有什麼弱點，這裡將惡意軟體型態分為 Server 端以及 Client 端。Server 端即為 botnet C&C server，而 client 端則為一般存於受感染電腦中的 bot 或是病毒以及視窗程式。首先將有漏洞的惡意軟體以 CWE[43]所分類的弱點型態來做一個整理。

Botnet 最早之前是以 IRC[44]類型為主，後來才衍生出經由 HTTP 的方式，發展出網頁介面的類型，藉由 HTTP 的 80 PORT 也比較不容易被防火牆阻止，而也因為利用網頁的緣故而使其可能會有 XSS 以及 SQL injection 的漏洞。

表 3. 惡意軟體弱點型態整理

弱點型態	發生原因	惡意軟體名稱	惡意軟體型態
Overflow	由 C&C message 發生，當 bot 從 message 中取出資料時，因為沒有做額外判斷讓取得的數值遠大於資料結構本身最大值。	Cutwail Bot、Zbot、Sasser Worm、Worm Web Server、Poison Ivy、Fakem RAT	Client、Server
Null Reference	由 C&C message 發生，bot 因為沒有做額外判斷而會存取到 Null Reference，例如使用 RtlAllocateHeap 時超過可用記憶體的最大空間。	Gheg Bot、Zbot	Client
Infinite Loop	在使用 C&C message 中的數值當作迴圈的間隔時，若是這數值為負數，則會造成重複回到原頭做相同的事情。	Zbot	Client
XSS(Cross-side scripting)	PHP XSS	VOlk botnet、Zeus botnet、Corpse botnet、Adrenalin botnet、EOF-0×01 botnet	Server、Client
SQL injection	SQL injection	VOlk、HerpesNet、Dirt Jumper DDoS Toolkit、BlackEnergy、Corpse、Zunkerbot	Server
Access control	藉由軟體密碼被得知而能直接控制這個軟體或是系統。	Poison Ivy	Server

Code injection	執行經由攻擊者所上傳檔案中的惡意指令。	PHP IRC Bot、Zeus botnet	Server

將所蒐集的資訊整理後，大致可以將弱點型態分為七項：overflow、null reference、infinite loop、XSS、SQL injection、access control 以及 code injection。由表 1 可知，bot 所潛在的弱點幾乎是與 C&C message 有關，也就是說只要能夠知道目標 bot 的 C&C message 格式，就能嘗試對 bot 做攻擊。而本研究所找到的 C&C server 是由 PHP 所撰寫，其中也存在著許多 XSS 以及 SQL injection 的漏洞。

接著以惡意軟體所被找出漏洞以及發佈年代還有規模來做分類，由下表可知這些惡意軟體都有一至二種類型的漏洞，由本章節開頭所述，由於現金 botnet 都是使用 HTTP base 的架構，因此 botnet 的 C&C server 中常可以找出 XSS 以及 SQL injection 的弱點。而在規模部分除了較有名的 Zeus 以及 Cutwail 之外，其餘能夠找到的都座落在數萬左右，有些並沒有被找出弱點的 botnet 會在後續稍微做個整理。

表 4. 惡意軟體與其弱點型態和規模

惡意軟體	弱點型態	弱點發佈年代	規模
ZeuS botnet	XSS Code injection	2011 2010	1300 萬
zBot(ZeuS botnet)	Overflow Infinite Loop Null Reference	2010 2010 2010	1300 萬
Cutwail Bot	Overflow	2010	150 萬
Gheg bot	NullReference	2010	10 萬
Sasser Worm	Overflow	2004	數萬至數十萬
Worm Web Server	Overflow	2000	數萬至數十萬
Zunkerbot botnet	SQL injection	2011	數萬
BlackEnergy botnet	SQL injection	2011	數千至數萬
HerpesNet botnet	SQL injection	2012	數千
Poison Ivy	Overflow Access control	2010 2012	--
Fakem	Overflow	2013	--
Volk botnet	XSS SQL injection	2012 2012	--
Corpse botnet	XSS SQL injection	2011 2011	--
Adrenalin botnet	XSS	2011	--
EOF-0×01 botnet	XSS	2011	--
Dirt Jumper DDoS Toolkit	SQL injection	2012	--
PHP IRC bot	Code injection	2012	--

接著整理這些弱點所對應的惡意軟體以及其 OSVDB、Exploit DB 的 ID，還有其回報的年份。

表 5. 惡意軟體的弱點以及其 OSVDB、Exploit DB 的 ID 跟回報年份

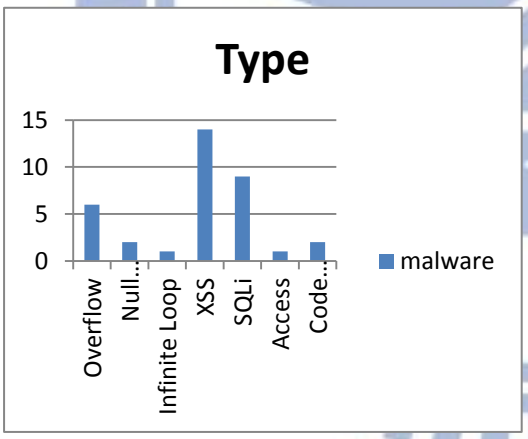
弱點型態	惡意軟體名稱	OSVDB ID	Exploit DB ID	回報年份
Overflow	Cutwail Bot	OSVDB-66497		2010
	Zbot	OSVDB-66501		2010
	Sasser Worm	OSVDB-6197		2004
	Worm Web Server	OSVDB-58524		2000
	Poison Ivy	OSVDB-83774	EDB-19613	2010
	Fakem RAT	OSVDB-92673		2013
Null Reference	Gheg Bot、	OSVDB-66498		2010
	Zbot	OSVDB-66499		2010
Infinite Loop	Zbot	OSVDB-66500		2010
XSS	VOlk botnet	OSVDB-86832	EDB-21891*	2012
		OSVDB-86833		2012
		OSVDB-86834		2012
		OSVDB-86835		2012
	Zeus botnet	OSVDB-82465		2011
		OSVDB-80275		2011
		OSVDB-80276		2011
	Corpse botnet	OSVDB-80269		2011
	Adrenalin botnet	OSVDB-80278		2011
	EOF-0×01 botnet	OSVDB-80272		2011
		OSVDB-80273		2011
		OSVDB-80274		2011
SQL injection	VOlk botnet	OSVDB-86830	EDB-21891*	2012
		OSVDB-86831		2012
	HerpesNet botnet	OSVDB-82464		2012
	Dirt Jumper DDoS Toolkit	OSVDB-85045		2012
		OSVDB-85046		2012
		OSVDB-85047		2012
	BlackEnergy botnet	OSVDB-80280		2011
		OSVDB-80281		2011
	Corpse botnet	OSVDB-80279		2011
		OSVDB-80271		2011

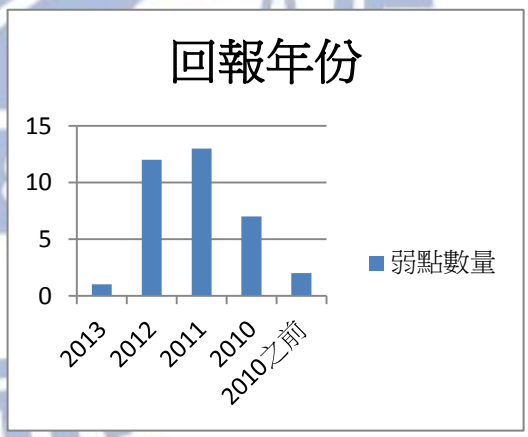
	Zunkerbot botnet	OSVDB-80277		2011
Access Control	Poison Ivy	OSVDB-83681		2012
Code Injection	PHP IRC Bot	OSVDB-84913	EDB-20168 EDB-20354	2012
	Zeus botnet	OSVDB-72651		2010

*註：皆為同一個 exploit

接著分別以型態、發現年代等等整理列表。由於本研究所找到的 C&C server 幾乎都是由 PHP 所撰寫，因此有不少相關研究者專門去接收這些樣本，接著再去嘗試尋找其中的 XSS 以及 SQLi 漏洞。而存在於受感染主機上的 bot 因為會與 C&C server 做溝通，因此中間傳遞的訊息 bot 處理時可能會產生一些 overflow、null reference 的狀況。

表 6、7. 弱點型態、回報年份數量





接著以回報的年份來看，2012、2011 年所找到的弱點主要都以 botnet C&C server 上為主，因此數量遽增，而 2010 主要為[5]這篇文獻中回報的 bot 弱點。

而惡意軟體像是 Zeus、Volk 這些 botnet 由於架設簡單，因此時常出現變種的型態潛伏在許多主機中，而文獻中[45]也提及這些 botnet 的大小以及影響範圍，至於在這篇中[30]提及了有些 botnet 是藉著併吞掉其他的 botnet 來壯大自己的勢力。

Botnet 災情主要是濫發垃圾郵件以及竊取銀行帳戶等等的個人資訊，在本研究所蒐集屬於前者的惡意軟體有 Cutwail、Gheg、Zunker Botnet、BlackEnergy Botnet，其中 Cutwail、Gheg 在 2010 的十大 spam botnets 中[19]排名分別為第三以及第十，Cutwail 能夠在一天中發送大約 190 億個垃圾郵件訊息，參與的 bots 數量則有 100 萬，目前也出現在 Android 系統上[46]。至於 Gheg 一天則是能夠散發 4 億封垃圾郵件，參與的 bots 數量則是 6 萬。

後者則有 ZeuS、VOlk、Zunker、BlackEnergy botnet，ZeuS 以及 vOlk 這類的 botnet 以竊取使用者銀行帳戶資訊為目標，ZeuS botnet[8]為目前最為龐大的 botnet 之一，根據統計[18, 20]在 2011 年從歐洲的銀行中竊取了 4700 萬美元，而微軟也公佈了竊取超過一億美元的兩個 ZeuS botnets，受到感染的電腦大約有 1300 萬部。FBI[47]也有提及感染 ZeuS malware 的受害者大多位於美國。而 vOlk botnet[6]主要影響區域在拉丁美洲，當然也有些遍佈在中國。至於 Dirt Jumper DDoS Toolkit 在 2012 年時也被發現了一些用來關閉整個 C&C server 的漏洞[48]，而這漏洞可以追溯到 2008 時的版本。

接著將一些上述提及的惡意軟體規模做個整理，並且也加入一些由[25, 46, 49]所整理的 botnet 規模。

表 8. 大規模的 Botnet 資訊

惡意軟體名稱	受感染的系統數量	資料發布時間	主要災情	備註
Pushdo/Cutwail	100 萬	2013	垃圾訊息	也出現在 Android 上，濫發 SMS
Gheg	6 萬	2010	垃圾訊息	
Conficker	1500 萬	2009	破壞使用者的系統	Windows 的 MS08-067 更新即為防堵此惡意軟體
ZeuS	1300 萬 360 萬	2012 2009	銀行帳戶密碼竊取	
Koopface	290 萬	2009	控制使用者的系統	主要由 facebook 以及 myspace 流傳。
TidServ	150 萬	2009	控制使用者的系統	

Pushdo/Cutwail 在 2010 時被評為第三大的 spam botnet，前者為 downloader，在感染之後會下載後者，接著由它來發送垃圾訊息、郵件。在 2013 年時有報導指出目前感染的數量仍在 100 萬左右，而且也在 Android 系統上發現其蹤跡。

ZeuS、Koopface 以及 TidServ 在 2009 年分別為美國 10 大 botnet 的前三名，當時 ZeuS 在美國感染的系統數量有 360 萬，直至現在數量更加的龐大，Conficker[50]則是微軟修補 Windows 的 MS08-067 安全性漏洞主要防堵的惡意軟體。而另外兩個由於並沒有相關的弱點訊息，因此並沒有做深入的探討。

由此得知遭受 botnet 所危害的總系統數量相當龐大，即便這些系統並非同一時間都會在線上，有些可能都是不再使用的個體，或者同時成為兩個 botnet 的一員，但是仍可以看的出來災害很大，若是能夠藉著這些已被找到的漏洞進行攻擊，便可能將整個 botnet 瓦解，或是從旁觀察它們的一舉一動，觀察整個被危害的系統的資訊，例如屬於何種作業系統、所在的區域以及在何種狀況下中毒，這些資訊都可以用來協助改善系統的安全性，藉此降低危害。

當然這些漏洞並非永遠存在，像是 ZeuS botnet 其中一個漏洞[37]已經被修正，因此開發一個自動化的測試工具來協助找尋新的漏洞，無論是對付 botnet 或是其他惡意程式都是很有用的，當然也需要考量該對這些惡意軟體做出什麼行動，像是直接摧毀或是觀察來收集資訊。

最後本研究嘗試以實驗室做提出的 CRAX [51]來嘗試重現上述整理的惡意軟體已知漏洞，CRAXweb 目前主要是以 XSS 以及 SQL injection 為主要目標，因此挑選 ZeuS、vOlk 以及 blackenergy C&C server 來做測試。至於 Bot，由[5]所述，必須要架設一個偽造的 Server 與其溝通，因此只選用 Poison Ivy 做為測試。

表 9. 實驗結果

測試的惡意軟體	找到的漏洞數量	應有的漏洞數量(*1)	漏洞型態	備註
ZeuS C&C server	-	4	Code injection	-
vOlk C&C server	1(*2)	5	SQL injection	-
Blackenergy C&C server	3	2	SQL injection	-
Poison Ivy	-	2	Stack Overflow	目前正在測試中

*註 1：這邊的原有漏洞數量為 OSVDB 的 ID 數量
*註 2：這個漏洞為可能的漏洞，目前正在嘗試產生 Exploit

由表 5 得知目前找出了 3 個屬於 Blackenergy 的漏洞，而這些找出的漏洞皆能產生其 Exploit 加以利用。

由於本研究所找到屬於 ZeuS C&C server 漏洞，其中兩個為舊版本，目前無法取得，而另外的與 I/O 有關，目前實驗室的工具並沒有處理這部分的判斷，因此並沒有找到 ZeuS 的漏洞。

在 volk C&C server 部分，所收集的漏洞有五個，其中包括了兩個 SQL injection 以及三個 XSS，測試的結果目前有發現一個疑似是漏洞的部分，只是還未能產生出結果。至於原有的漏洞經過追蹤原始碼後，確定其中兩個漏洞已被修復。

接著在 BlackEnergy C&C server 上，雖然表 5 標示所蒐集的漏洞為兩個，但其中一個根據在 OSVDB 上的說明，實際上是多個 SQL injection，而實驗室開發的工具在這個 server 上則找出了三個漏洞。

五、 總結

結論

現今惡意軟體的危害日趨嚴重，網路上許多連結常夾帶惡意行為可能讓使用者電腦淪為 botnet 的一員，也造成了現實社會中的災害損失，而惡意軟體中常使用加密演算法來保護內部資訊以及惡意指令，規避防毒軟體以及一般分析軟體的檢查，botnet 的 C&C server 也因為架構問題而難以瓦解。雖然目前以防禦角度的方法例如防毒軟體等能夠減少危害，但時常因為惡意軟體版本更動而使其失效，所謂治標不治本，因此本研究以直接針對惡意軟體進行測試與攻擊為目的，包括背後的 botnet C&C server，收集並了解這方面的研究。

所以本研究整理了以下幾個內容：

1. 加密演算法的辨認以及處理
   由前述所提，一般分析測試軟體會因為加密演算法的關係而沒有辦法有效的對惡意軟體產生測試資料，因此在探討了目前有針對惡意軟體這類問題的文獻後，了解他們大多是利用 heuristic function 去判斷惡意軟體執行時的指令以及資料，然後辨認是哪種加密演算法。

2. 惡意軟體的攻擊、分析方法案例
   目前已有許多針對惡意軟體的研究，因此本研究整理了幾項方法作為例子，探討如何對惡意軟體做測試分析，找出其中的漏洞，其中包含了一個自動化的測試工具。

3. 已被發現屬於惡意軟體的弱點分析
   惡意軟體也與一般軟體一樣，可能潛在著漏洞，而這些漏洞可以用來產生 Exploit 藉此產生一些原本沒有的行為，例如 botnet C&C server 的 Exploit 便能使之關閉整個 botnet，因此蒐集這些漏洞並分析其發生原因。

從這些內容中知道，測試過程中加密演算法會造成很大的阻礙，因此許多研究都有提及解決加密演算法為測試惡意軟體的先決條件，而在解決這部分後惡意軟體便與未使用加密的軟體無異，能夠使用測試工具進行測試，而後續所整理的惡意軟體常見的漏洞也能改進測試工具的效率。

在第三章節介紹了四種尋找惡意軟體弱點的案例，由於加密演算法的緣故，目前許多方式都是直接去追蹤程式碼來觀察是否有漏洞存在，因此相當的耗費時間，自動化的方式雖然能夠解決這問題，但若是這些惡意軟體試圖改變加密演算法來混淆，反而沒辦法達到原有自動化的效果，在這種情況下以手動方式追蹤其

內部的程序反而是個好辦法。

而 C&C Server 的部分也有許多 XSS 以及 SQL injection 存在，雖然由於架構關係就算能夠成功攻陷其中一個 Server，但仍可能有其餘 Server 存在著，即使如此這些收集的漏洞仍可以讓測試工具專注於其目標。最後利用實驗室所開發的工具嘗試重現所蒐集到的漏洞，由實驗結果得知確實能夠重現部分已知的漏洞外，並且這些漏洞也都能產生出其 Exploit。

未來展望

在第四章節介紹了已知屬於惡意軟體的漏洞後，嘗試以實驗室所開發的工具重現這些漏洞，除了成功重現部分已知的之外，也產生其 Exploit。因此做為後續的發展，接著將更進一步的測試其他的惡意軟體，例如必須架設偽裝與其溝通的 bot，而由於有些漏洞會在惡意軟體更新後消失，因此也會蒐集不同版本加以測試。

由於這邊主要是探討 botnet 相關的漏洞，因此後續也會以其他類型的惡意軟體例如 spyware、rootkits 進行研究跟分析。

本研究以探討攻擊與測試惡意軟體的方法為主要目標，整理相關方法以及分析已知漏洞的發生狀況，作為後續研究的參考資料，藉此用以開發自動化的測試工具來瓦解 botnet，而能夠降低其危害。

參考文獻

1. McGraw, G., *Software security.* Security & Privacy, IEEE, 2004. **2**(2): p. 80-83.
2. Oehlert, P., *Violating assumptions with fuzzing.* Security & Privacy, IEEE, 2005. **3**(2): p. 58-62.
3. Ganesh, V., T. Leek, and M. Rinard. *Taint-based directed whitebox fuzzing*. in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. 2009. IEEE.
4. Matenaar, F., et al. *CIS: The Crypto Intelligence System for automatic detection and localization of cryptographic functions in current malware*. in *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*. 2012. IEEE.
5. Caballero, J., et al. *Input generation via decomposition and re-stitching: Finding bugs in malware*. in *Proceedings of the 17th ACM conference on Computer and communications security*. 2010. ACM.
6. Jorge Mieres, K.L.E. *Latin American banks under fire from the Mexican VOlk-Botnet*. 2012; Available from: http://www.securelist.com/en/blog/208193160/Latin_American_banks_under_fire_from_the_Mexican_VOlk_Botnet2012-10-11.
7. Zhu, Z., et al. *Botnet research survey*. in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*. 2008. IEEE.
8. Falliere, N. and E. Chien, *Zeus: King of the Bots.* Retrieved from Security Response Whitepapers Symantec Corp. website: http://www. symantec. com/content/en/us/enterprise/media/security_response/whitepapers/zeus_king_of_bots. pdf, 2009.
9. Wu, Y., et al. *Malware network behavior extraction based on dynamic binary analysis*. in *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on*. 2012. IEEE.
10. Wang, P., et al., *Honeypot detection in advanced botnet attacks.* International Journal of Information and Computer Security, 2010. **4**(1): p. 30-51.
11. Dinaburg, A., et al. *Ether: malware analysis via hardware virtualization extensions*. in *Proceedings of the 15th ACM conference on Computer and communications security*. 2008. ACM.
12. Moser, A., C. Kruegel, and E. Kirda. *Exploring multiple execution paths*

for malware analysis. in *Security and Privacy, 2007. SP'07. IEEE Symposium on*. 2007. IEEE.

13. Godefroid, P., N. Klarlund, and K. Sen. *DART: directed automated random testing*. in *ACM Sigplan Notices*. 2005. ACM.
14. Egele, M., et al., *A survey on automated dynamic malware-analysis techniques and tools.* ACM Computing Surveys (CSUR), 2012. **44**(2): p. 6.
15. Yoshioka, K., et al. *Vulnerability in public malware sandbox analysis systems*. in *Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on*. 2010. IEEE.
16. Wang, T., et al. *TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection*. in *Security and Privacy (SP), 2010 IEEE Symposium on*. 2010. IEEE.
17. Weaver, A. and P. OWASP, *Breaking Botnets: Finding App Vulnerabilities in Botnet Command and Control Servers.*, 2011.
18. *Microsoft identifies two Zeus botnet crime ring suspects*. 2012; Available from: http://news.cnet.com/8301-1009_3-57465470-83/microsoft-identifies-two-zeus-botnet-crime-ring-suspects/.
19. *The top 10 spam botnets: New and improved*. 2010; Available from: http://www.techrepublic.com/blog/10things/the-top-10-spam-botnets-new-and-improved/1373.
20. *Oracle Security Alert for CVE-2012-4681*. 2012; Available from: http://www.oracle.com/technetwork/topics/security/alert-cve-2012-4681-1835715.html.
21. Stone-Gross, B., et al., *The underground economy of fake antivirus software*, in *Economics of Information Security and Privacy III*. 2013, Springer. p. 55-78.
22. Clarke, T., *Fuzzing for software vulnerability discovery.* Department of Mathematic, Royal Holloway, University of London, Tech. Rep. RHUL-MA-2009-4, 2009.
23. Bekrar, S., et al. *Finding software vulnerabilities by smart fuzzing*. in *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*. 2011. IEEE.
24. Gröbert, F., C. Willems, and T. Holz. *Automated identification of cryptographic primitives in binary programs*. in *Recent Advances in Intrusion Detection*. 2011. Springer.
25. *America's 10 most wanted botnets*. 2009; Available from:

http://www.networkworld.com/news/2009/072209-botnets.html.
26. Willems, C., T. Holz, and F. Freiling, *Toward automated dynamic malware analysis using cwsandbox.* Security & Privacy, IEEE, 2007. **5**(2): p. 32-39.
27. Leder, F., T. Werner, and P. Martini, *Proactive botnet countermeasures: an offensive approach.* The Virtual Battlefield: Perspectives on Cyber Warfare, 2009. **3**: p. 211-225.
28. Feily, M., A. Shahrestani, and S. Ramadass. *A survey of botnet and botnet detection*. in *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*. 2009. IEEE.
29. Rodrigues, N.G., A. Nogueira, and P. Salvador, *Fighting botnets-a systematic approach.* 2012.
30. Stone-Gross, B., et al. *Your botnet is my botnet: analysis of a botnet takeover*. in *Proceedings of the 16th ACM conference on Computer and communications security*. 2009. ACM.
31. Bächer, P., et al. *Know your Enemy: Tracking Botnets*. 2008; Available from: http://www.honeynet.org/papers/bots/.
32. Song, D., et al., *BitBlaze: A new approach to computer security via binary analysis*, in *Information systems security*. 2008, Springer. p. 1-25.
33. Miller, C., et al., *Crash analysis with BitBlaze.* at BlackHat USA, 2010.
34. Chipounov, V., et al. *Selective symbolic execution*. in *Workshop on Hot Topics in Dependable Systems*. 2009.
35. Calvet, J., J.M. Fernandez, and J.-Y. Marion. *Aligot: cryptographic function identification in obfuscated binary programs*. in *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012. ACM.
36. Rascagneres, P., et al. *Analysis & pownage of herpesnet botnet*. 2012; Available from: https://code.google.com/p/malware-lu/wiki/en_analyse_herpnet.
37. Hardin, B. and B. Rios. *Imagination - XSS and XSRF*. 2011; Available from: http://spotthevuln.com/2011/07/imagination-xss-and-xsrf/.
38. Rios, B.B. *Turning the Tables*. 2010; Available from: http://xs-sniper.com/blog/2010/09/27/turning-the-tables/.
39. *Open Source Vulnerability Database (OSVDB)*. 2002 - 2013; Available from: http://www.osvdb.org/.
40. *National Vulnerability Database*. Available from: http://nvd.nist.gov/.

41. *Open Malware*. Available from: http://www.offensivecomputing.net/.
42. *Malware Domain List*. 2009; Available from: http://www.malwaredomainlist.com/.
43. *CWE - Common Weakness Enumeration*. Available from: http://nvd.nist.gov/cwe.cfm.
44. Puri, R., *Bots & botnet: An overview.* SANS Institute 2003, 2003.
45. Fabian, M.A.R.J.Z. and M.A. Terzis. *My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging*. in *Proceedings of the 1st USENIX Workshop on Hot Topics in Understanding Botnets, Cambridge, USA*. 2007.
46. Stone-Gross, B. *Cutwail Spam Botnet Targeting Android Users*. 2013; Available from: http://www.f-secure.com/weblog/archives/00002537.html.
47. FBI. *Cyber Banking Fraud*. 2010-2013; Available from: http://www.fbi.gov/news/stories/2010/october/cyber-banking-fraud.
48. Dunn, J.E. *Popular Dirt Jumper DDoS toolkit riddled with security flaws, research finds*. 2012; Available from: http://news.techworld.com/security/3376047/popular-dirt-jumper-ddos-toolkit-riddled-with-security-flaws-research-finds/.
49. *Kaspersky Lab*. Available from: http://www.securelist.com/en/.
50. Microsoft. *Help protect yourself from the Conficker worm*. 2009; Available from: http://www.microsoft.com/security/pc-security/conficker.aspx.
51. Huang, S.-K., et al. *CRAX: Software Crash Analysis for Automatic Exploit Generation by Modeling Attacks as Symbolic Continuations*. in *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*. 2012. IEEE.