

# 銘傳大學

資訊工程學系碩士班

碩士論文

OPC-UA、NetBIOS 及 DHCPv4 模糊測試  
的研究與實作



研 究 生：高子翔

指導教授：徐武孝 博士

中 華 民 國 107 年 7 月

# 銘傳大學

## 碩士班

### 論文口試委員會審定書

本校 資訊工程 研究所 高子翔 君

所提論文 OPC-UA、NetBIOS 及 DHCPv4 模糊測

試的研究與實作 合於碩士資格水準，業經本委員會

評審認可。

口試委員：

(召集人) 閻宏燦

謝育平  
徐武厚

指導教授：

徐武厚

系主任：

李嘯嘯 教授

中華民國 107 年 7 月 20 日

# OPC-UA、NetBIOS 及 DHCPv4 模糊測試 的研究與實作

研究生：高子翔

指導教授：徐武孝

銘傳大學資訊工程學系

## 摘 要

本論文以模糊測試的技術來找出 OPC-UA、NetBIOS 以及 DHCPv4 三種協定的潛在威脅，進而提高網路協定的安全性。模糊測試的過程分為四個步驟：(1) 研讀網路協定的封包格式及傳輸方式。(2) 在用戶端將測試的格式及測試資料輸入所撰寫的程式。(3) 架設伺服器並以我們所撰寫的程式對伺服器進行模糊測試。(4) 使用 Wireshark 觀察是否有出現不正常封包。本研究用戶端以 Linux 作為測試平台，而伺服器端的平台則是依據網路協定的特性而安裝在不同的平台。考量到需要處理大量的數據，因此在撰寫程式時以 python 為撰寫程式的語言，以便我們能更有效率的做模糊測試。實作成果顯示三種網路協定都可以使用此模糊測試程式來判斷是否有潛在威脅，藉此提高網路安全性。

**關鍵字：**模糊測試、網路安全、OPC-UA、NetBIOS、DHCPv4

# The Design and Implementation of Fuzz testing for OPC-UA, NetBIOS and DHCPv4

Student: KAO,TZU-HSIANG

Advisor: SHYU,WU-HSIAO

Department of Computer Science Information Management  
Ming Chuan University

## ABSTRACT

This thesis uses fuzz testing techniques to identify potential threats for the three protocols: OPC-UA, NetBIOS, and DHCPv4 in order to improve the security of these network protocols. The process of fuzzy testing is divided into four steps: (1) To study the packet format and transmission method of the network protocol. (2) To input the test formant and data into the developed program on the user side. (3) To install the server and fuzz the server with the developed program. (4) To use Wireshark to check if there are any abnormal packets. Client uses Linux as a platform in this research and the server is installed on different platforms according to the characteristics of the network protocol. Python is used to develop our program, so that we can do fuzzing more efficiently. The results show that the three network protocols can use this fuzzing program to determine whether there is a potential threat, thereby improving network security.

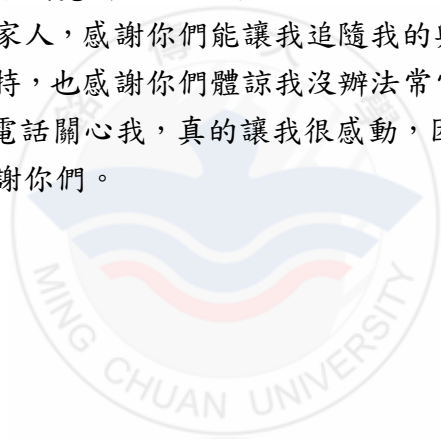
**Keywords:** fuzz testing, network security, OPC-UA, NetBIOS, DHCPv4

## 誌 謝

時光飛逝，從兩年前剛進研究所到現在已經快要結束，其中有辛苦也有徬徨，但也都跟研究所的老師及成員們一起度過了，我們一起走過艱難、一起走過困苦，在想要放棄的時候互相勉勵，終於堅持到了這一刻，現在回想起來心中只有滿滿的不捨及感謝。首先我想先感謝我的指導教授 徐武孝老師，老師他真的幫助了我很多，從剛入研究所的照顧，作論文的寶貴建議以及人生的經歷，也時常督促我的缺點，讓我能有機會改正我的缺陷，在此表示深摯的感謝。

而在研究所的過程中同學及學長也都給了我很大的幫助。感謝王驊及愷林學長在我剛進研究所時非常的照顧我，有問題時也都有問必答，也在我讀研究所的過程中多了許多歡笑。感謝邱彥銘及劉庭瑩一路上的陪伴以及支持，研究所的路上有你們互相鼓勵也是我的福氣。感謝高偉軒學弟在我做計劃時常常幫助我許多，讓我在枯燥乏味的計畫中感覺到一絲溫暖。

最後我想感謝我的家人，感謝你們能讓我追隨我的興趣，在我身心靈都快接近崩潰的時候帶給我支持，也感謝你們體諒我沒辦法常常回家陪你們，不但沒有怪我沒回家，還時常打電話關心我，真的讓我很感動，因此我在這求學快要結束的階段我只想說聲，謝謝你們。



# 目錄

中文摘要.....	i
英文摘要.....	ii
誌謝.....	iii
目錄.....	iv
圖目錄.....	vi
第一章 緒論.....	1
1.1 研究背景 .....	1
1.2 研究動機 .....	1
1.3 論文架構 .....	1
第二章 文獻探討.....	2
2.1 模糊測試 .....	2
2.2 OSS-Fuzz.....	2
第三章 研究方法及實作成果.....	3
3.1 OPC-UA .....	4
3.1.1 OPC-UA 封包傳遞方式 .....	4
3.1.2 OPC-UA 所有封包類型格式 .....	5
3.1.3 OPC-UA 伺服器架設 .....	7
3.1.4 OPC-UA 模糊測試成果 .....	8
3.2 NetBIOS .....	10
3.2.1 NetBIOS 封包傳遞方式 .....	10
3.2.2 NetBIOS 所有封包類型格式 .....	11
3.2.3 NetBIOS 伺服器架設 .....	15
3.2.4 NetBIOS 模糊測試成果 .....	15
3.3 DHCPv4.....	17
3.3.1 DHCPv4 封包傳遞方式 .....	18
3.3.2 DHCPv4 所有封包類型格式 .....	19
3.3.3 DHCPv4 伺服器架設 .....	20
3.3.4 DHCPv4 模糊測試成果 .....	21
第四章 結論.....	23
參考文件.....	24

# 圖目錄

圖 1、模糊測試進行流程.....	3
圖 2、確認伺服器是否存活.....	4
圖 3、建立安全通道.....	4
圖 4、傳輸資料及數據.....	5
圖 5、關閉安全通道.....	5
圖 6、HEL 封包格式.....	6
圖 7、ACK 封包格式.....	6
圖 8、OPN 封包格式.....	6
圖 9、MSG 封包格式.....	6
圖 10、CLO 封包格式.....	7
圖 11、ERR 封包格式.....	7
圖 12、進行 OPC-UA 模糊測試時小黑窗的過程.....	8
圖 13、使用 Wireshark 觀察 OPC-UA 封包.....	9
圖 14、NetBIOS 可以用於電腦之間的封包傳遞.....	10
圖 15、Name Service 封包格式總覽.....	13
圖 16、HEADER 欄位的詳細格式.....	13
圖 17、QUESTION ENTRIES 欄位的詳細格式.....	13
圖 18、ANSWER RESOURCE RECORDS 欄位的詳細格式.....	13
圖 19、Datagram Service 服務的詳細格式.....	14
圖 20、Session Service 服務的詳細格式.....	14
圖 21、進行 NetBIOS 模糊測試時小黑窗的過程.....	16
圖 22、使用 Wireshark 觀察 NetBIOS 封包.....	16
圖 23、DHCPv4 4-ways 架構.....	18
圖 24、DHCPv4 2-ways 架構.....	19
圖 25、DHCPv4 詳細封包格式.....	20
圖 26、進行 DHCPv4 模糊測試時小黑窗的過程.....	22
圖 27、使用 Wireshark 觀察 DHCPv4 封包.....	22

# 第一章 緒論

## 1.1 研究背景

隨著科技越來越發達，網路已經成為大家每天會使用的必需品，而網路的安全性變的更重要，如同買車會多買一道鎖、寫信會黏貼信封紙一樣，都是為了提升安全性所做的措施，那在現代人人都會使用網路的情況下網路安全就顯得更加重要，因此在現代已經有非常多人致力於提升網路安全性，而模糊測試就是其中一種方法。

## 1.2 研究動機

正如 1.1 小節所提到的提升網路安全性是非常重要的，只要是網路協定就一定會有被攻擊的可能性，因此在本研究中的想法為比起被攻擊後才修補漏洞，不如提前找出網路協定中的潛在威脅，即可在被攻擊前提前預防，而模糊測試就是藉由測試網路協定是否存在著潛在的威脅的一種技術，進而提升網路安全性。

## 1.3 論文架構

本論文架構如下所述：第一節先敘述跟對各種網路協定進行模糊測試的背景及動機；第二節討論和本論文有相關的文獻；第三節詳細描述如何進行模糊測試；第四節說明本論文的結論。



## 第二章 文獻探討

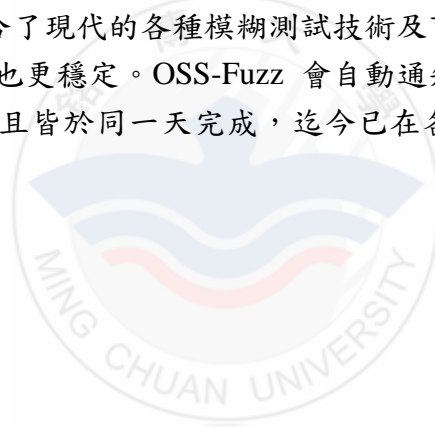
### 2.1 模糊測試[1]

模糊測試的概念最早由美國威斯康辛大學(University of Wisconsin)的巴頓·米勒(Barton Miller)在1988年提出，是一種自動化或半自動化的軟體測試技術，藉由發送”無效”、”預期外”或”隨機”的資料到測試目標，然後監測例如當機，程序崩潰或緩存區溢出，被普遍用於發掘軟體系統的未知安全漏洞。模糊測試並不限定於測試網絡協議，亦可測試對象文件格式、系統 IPC 資料交換、環境變數、鍵盤和滑鼠事件、API 調用等。

### 2.2 OSS-Fuzz[2]

Google 為了提升其程式、網站、或服務的骨幹的穩定性、安全與可靠性，因此開發出了 OSS-Fuzz 來找出程式中的漏洞，於軟體中輸入無效或隨機資料來快速及全面監控程式的反應，用來檢測軟體或電腦系統的安全漏洞。

OSS-Fuzz 即是結合了現代的各種模糊測試技術及可延展的分散式執行來讓通用的軟體架構更安全也更穩定。OSS-Fuzz 會自動通知維護程式的開發人員，並自動進行修補確認，且皆於同一天完成，迄今已在各種開放源碼專案中找到 150 個臭蟲。



### 第三章 研究方法及實作成果

本論文所研究及實作是為了找出未來使用各種網路協定時可能會造成威脅的網路封包，運作流程如圖一所示。目前模糊測試的伺服器是架設於實驗室中，然後由另一台電腦做為客戶端發送各種網路協定的封包給伺服器，再經由 Wireshark [3] 去觀察客戶端發送了哪種封包以及伺服器如何回應，在本研究中選擇了三種網路協定進行模糊測試，分別為 OPC-UA、NetBIOS 及 DHCPv4，三種網路協定詳細介紹會分別於第 3.1 小節、第 3.2 小節及第 3.3 小節詳細說明。

發送模糊測試的封包則是使用 Python 撰寫程式讓他自動發送大量指定網路協定的封包至伺服器，發送哪種封包也是可以經由預先設定好的指令進行調整，

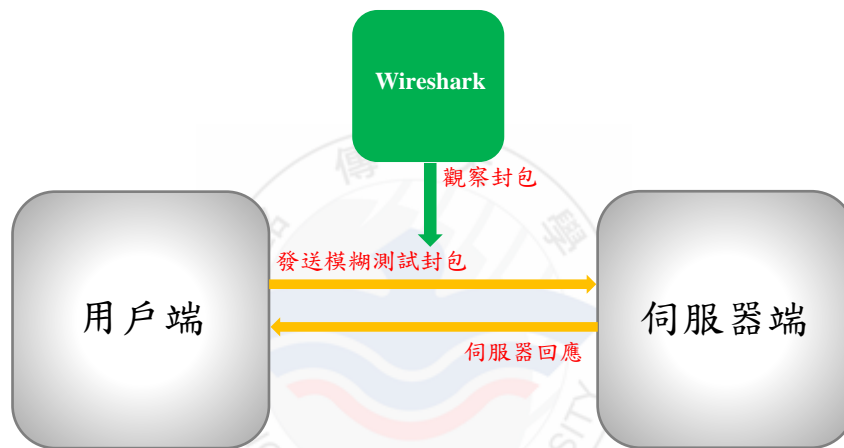


圖 1、模糊測試進行流程

### 3.1 OPC-UA

OPC-UA [4][5] 是一種應用於工業4.0自動化工業中機器與機器之間的網路協定，雖然現在還沒有全世界統一的通用標準，但是工業4.0是未來工業的趨勢，因此其安全性也是非常重要。

#### 3.1.1 OPC-UA 封包傳遞方式

為了知道要對 OPC-UA 的哪個類型的封包做模糊測試，因此必須先去知道 OPC-UA 的傳遞方式[6]，才可以決定要對哪個類型做模糊測試。

OPC-UA 的傳遞是建立在 TCP 之下的要進行通訊首先要先進行 TCP 三向交握，再來要確認 OPC-UA 的伺服器端是否還存活，因此客戶端會發送 HEL 類型封包給伺服器端如圖 2 所示。確認伺服器端存活後為了傳送資料及數據，需要傳送 OPN 類型封包建立雙方的資料交換安全通道如圖 3 所示。建立完通道即可開始傳輸所需的資料及數據，在其中交換包含資料的 MSG 類型封包如圖 4 所示。傳輸完資料後要關閉安全通道則需要傳送 CLO 類型封包告知伺服器端要關閉安全通道如圖 5 所示。

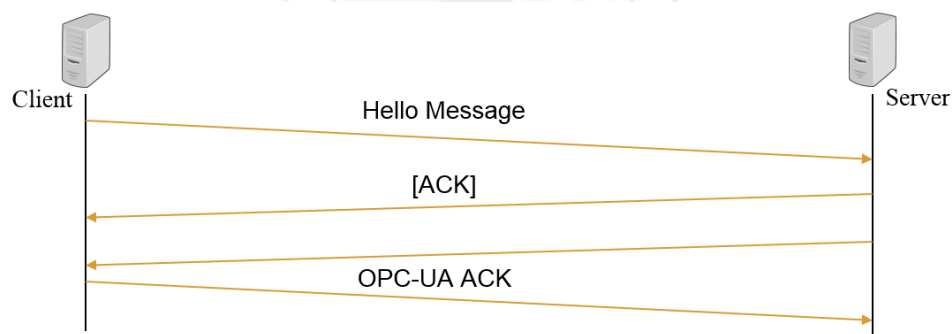


圖 2、確認伺服器是否存活

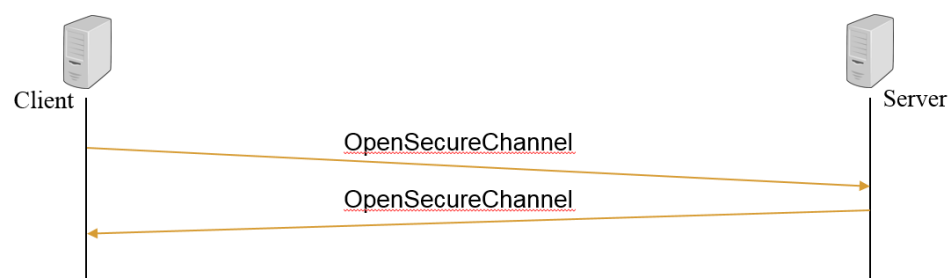


圖 3、建立安全通道



圖 4 、傳輸資料及數據

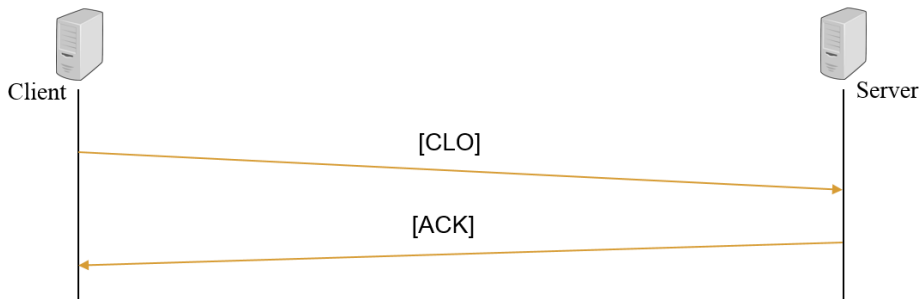


圖 5 、關閉安全通道

### 3.1.2 OPC-UA 所有封包類型格式

決定要對哪個類型的封包進行模糊測試後，為進行模糊測試需要針對不同類型的封包做數據修改，因此必須要了解 OPC-UA 每種類型的封包格式[7]，封包格式共有六種，分別為 HEL、ACK、OPN、MSG、CLO、ERR，每個類型詳細的封包格式如圖 6 到圖 11 所示：

MessageType：辨識此 OPC-UA 封包是何種類型的封包

ChunkType：1bit，為補長 MessageType 成為 1byte 的欄位。

ReceiveBufferSize：用戶端可以接收的最大封包長度。

SendBufferSize：用戶端將發送的最大封包長度。

MaxMessageSize：任何回應封包的最大大小

MaxChunkCount：任何回應封包中的單個欄位的最大大小

EndPointUrl：用戶端希望連接端點的 URL

SecureChannelID：建立安全通道後所使用的密碼，之後的訊息封包 ID 必須跟安全通道的密碼一樣

SecurityPolicyUrl：OPC-UA 每個安全政策的 URL

SenderCertificate：用於標識用戶端應用程序的應用程序實例證書

ReceiverCertificateThumbprint：伺服器端的證書指紋。

0	15	31
Message Type (HEL)	Chunk Type	Message Size
32	Version	Receive Buffer Size
64	Send Buffer Size	Max Message Size
96	Max Chunk Count	End Point Url
128		

圖 6 、HEL 封包格式

0	15	31
Message Type (ACK)	Chunk Type	Message Size
32	Version	Receive Buffer Size
64	Send Buffer Size	Max Message Size
96	Max Chunk Count	

圖 7 、ACK 封包格式

0	15	31
Message Type (OPN)	Chunk Type	Message Size
32	Secure Channel ID	Security Policy Url
64	Sender Certificate	Receiver Certificate Thumbprint
96	Sequence Number	Request ID
128	DATA	

圖 8 、OPN 封包格式

0	15	31
Message Type (MSG)	Chunk Type	Message Size
32	Secure Channel ID	Security Token ID
64	Security Sequence Number	Security Request ID
96	DATA	

圖 9 、MSG 封包格式

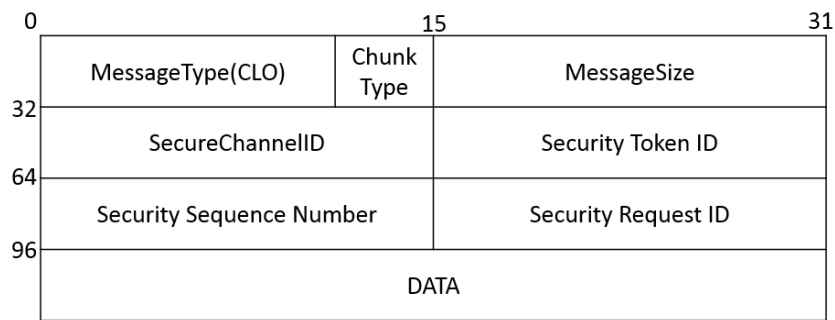


圖 10 、CLO封包格式

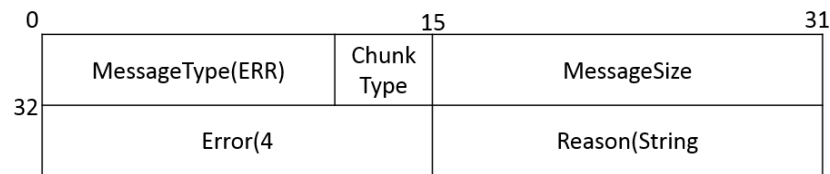


圖 11 、ERR封包格式

### 3.1.3 OPC-UA 伺服器架設

本測試環境使用一個 ubuntu14.04 虛擬機(VM)作為模糊測試系統平台，設定伺服器端以及用戶端，其步驟如下：

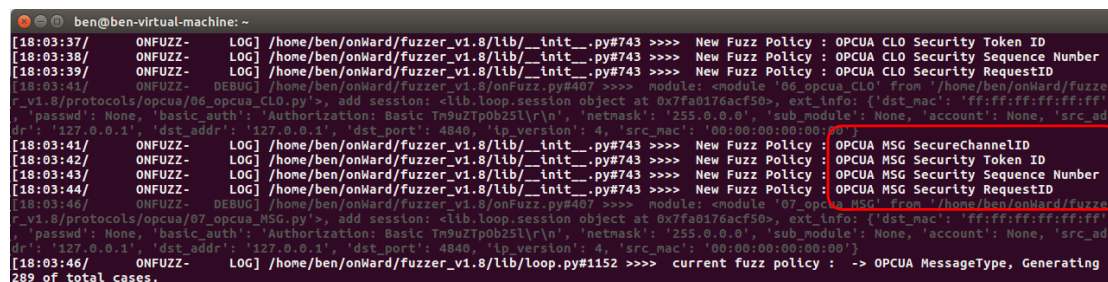
1. 至檢測端虛擬機開啟Terminal應用程式
2. 執行 `sudo apt-get install python-pip3`
3. 執行 `sudo apt-get install python-pyqt5`
4. 更新套件
5. 執行 `sudo pip3 install opcua-client`
6. 執行 `uaserver`
7. 建置完成

### 3.1.4 OPC-UA 模糊測試成果

OPC-UA 的測試環境建置完成之後即可開始進行模糊測試，在程式設計方面要先選擇如何製作測試封包，本研究中製作測試封包的方法為將相同格式封包的每個封包欄位分別進行改動成為錯誤封包的測試封包，目的是將測試封包發送至伺服器端之後若是伺服器有錯誤反應就可以知道此封包是否是可能成為威脅的封包，每改動一次就送一次測試封包給伺服器端。

以 OPC-UA 來說，由於 MessageType 欄位是判斷此 OPC-UA 是哪種封包，所以首先挑選 MessageType 欄位根據我們所輸入的測資做改動，其他欄位都固定，MessageType 欄位的值每改動一次就是一個測試封包，將 MessageType 欄位做完測試封包後再將 MessageType 欄位分別固定為 HEL、ACK、OPN、MSG、CLO 及 ERR 進行其他欄位測試封包的製作，其中要注意的是 MessageType 欄位若固定成 HEL 以外的封包值的話，在發送測試封包前都必須預先發送一次正確的 HEL 封包才會合乎 OPC-UA 的封包傳遞方式，將所有封包欄位的測試封包都設定完成之後即可開始進行模糊測試。

製作完測試封包後開始進行模糊測試，輸入預先設定好的指令 `sudo python onFuzz.py opcua 127.0.0.1 -p 4840`，設定好的指令中 `opcua` 是這次模糊測試的網路協定，`127.0.0.1` 是目的端的 IP，`-p` 則是將目的端的 port 指定成 4840，輸入指令後，用戶端就會向伺服器端發送大量 OPC-UA 格式的封包，其過程的一部分如圖 12 所示，從中可以看到圖 12 中正在進行的是將 MessageType 欄位固定為 CLO 及 MSG 後對其他封包欄位分別做改動，而在發封包的過程中我們就可以利用 Wireshark 來觀察是否有不正常的封包，成果如圖 13 所示，可以清楚的看到 OPC-UA 在進行測試 MessageType 欄位固定成 ACK 時測試封包的流程，由於 OPC-UA 是建立於 TCP 連線基礎上的網路協定，因此第 623 項到第 625 項可以看到正在進行三向交握，第 626 項及第 627 項則為 OPC-UA 會在進行一次封包 ACK 交換，之後第 664 項及第 665 項則是因為在發送 MessageType 欄位固定成 ACK 時的測試封包必須先發送一次 HEL 封包才合乎 OPC-UA 的封包傳遞方式，第 666 項則是我們所製作的測試封包，由於測試封包是我們故意製作出來的錯誤封包，因此伺服器端不回應我們所發送的測試封包，表示這次發送的測試封包並不是潛在威脅的封包。



```
ben@ben-virtual-machine: ~
[18:03:37/ ONFUZZ- LOG] /home/ben/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : OPCUA CLO Security Token ID
[18:03:38/ ONFUZZ- LOG] /home/ben/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : OPCUA CLO Security Sequence Number
[18:03:39/ ONFUZZ- LOG] /home/ben/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : OPCUA CLO Security RequestID
[18:03:41/ ONFUZZ- DEBUG] /home/ben/onWard/fuzzer_v1.8/onFuzz.py#407 >>> module: <module '06_opcua_CLO' from '/home/ben/onWard/fuzzer_v1.8/protocols/opcua/06_opcua_CLO.py'>, add session: <lib.loop.session object at 0x7fa0176acf50>, ext_info: {'dst_mac': 'ff:ff:ff:ff:ff:ff', 'passwd': None, 'basic_auth': 'Authorization: Basic Tm9uZTp0b251\r\n', 'netmask': '255.0.0.0', 'sub_module': None, 'account': None, 'src_addr': '127.0.0.1', 'dst_addr': '127.0.0.1', 'dst_port': 4840, 'ip_version': 4, 'src_mac': '00:00:00:00:00:00'}
[18:03:41/ ONFUZZ- LOG] /home/ben/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : OPCUA MSG SecureChannelID
[18:03:42/ ONFUZZ- LOG] /home/ben/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : OPCUA MSG Security Token ID
[18:03:43/ ONFUZZ- LOG] /home/ben/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : OPCUA MSG Security Sequence Number
[18:03:44/ ONFUZZ- LOG] /home/ben/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : OPCUA MSG Security RequestID
[18:03:46/ ONFUZZ- DEBUG] /home/ben/onWard/fuzzer_v1.8/onFuzz.py#407 >>> module: <module '07_opcua_MSG' from '/home/ben/onWard/fuzzer_v1.8/protocols/opcua/07_opcua_MSG.py'>, add session: <lib.loop.session object at 0x7fa0176acf50>, ext_info: {'dst_mac': 'ff:ff:ff:ff:ff:ff', 'passwd': None, 'basic_auth': 'Authorization: Basic Tm9uZTp0b251\r\n', 'netmask': '255.0.0.0', 'sub_module': None, 'account': None, 'src_addr': '127.0.0.1', 'dst_addr': '127.0.0.1', 'dst_port': 4840, 'ip_version': 4, 'src_mac': '00:00:00:00:00:00'}
[18:03:46/ ONFUZZ- LOG] /home/ben/onWard/fuzzer_v1.8/lib/loop.py#1152 >>> current fuzz policy : -> OPCUA MessageType, Generating 289 of total cases.
```

模糊測試 MSG

圖 12、進行 OPC-UA 模糊測試時小黑窗的過程

Filter: **tcp.port == 4840** Expression... Clear Apply 儲存

No.	Time	Source	Destination	Protocol	Length	Info
623	46.466376232	127.0.0.1	127.0.0.1	TCP	76	48024 → 4840 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=816169168 TSecr=0 WS=1
624	46.466413931	127.0.0.1	127.0.0.1	TCP	76	4840 → 48024 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=816169168 TSecr=0 WS=1
625	46.466423045	127.0.0.1	127.0.0.1	TCP	68	48024 → 4840 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=816169168 TSecr=816169168
626	46.467030148	127.0.0.1	127.0.0.1	TCP	123	[TCP segment of a reassembled PDU]
627	46.467036994	127.0.0.1	127.0.0.1	TCP	68	4840 → 48024 [ACK] Seq=1 Ack=56 Win=43776 Len=0 TSval=816169168 TSecr=816169168
664	51.078063733	127.0.0.1	127.0.0.1	OpCua	97	Hello message [TCP segment of a reassembled PDU]
665	51.078070325	127.0.0.1	127.0.0.1	TCP	68	4840 → 48024 [ACK] Seq=1 Ack=85 Win=43776 Len=0 TSval=816170321 TSecr=816170321
666	51.078257967	127.0.0.1	127.0.0.1	OpCua	96	Acknowledge message
667	51.078304278	127.0.0.1	127.0.0.1	TCP	68	48024 → 4840 [ACK] Seq=85 Ack=29 Win=43776 Len=0 TSval=816170321 TSecr=816170321
668	51.078351757	127.0.0.1	127.0.0.1	TCP	68	48024 → 4840 [FIN, ACK] Seq=85 Ack=29 Win=43776 Len=0 TSval=816170321 TSecr=816170321
669	51.078475905	127.0.0.1	127.0.0.1	TCP	68	4840 → 48024 [FIN, ACK] Seq=29 Ack=86 Win=43776 Len=0 TSval=816170321 TSecr=816170321
670	51.078480019	127.0.0.1	127.0.0.1	TCP	68	48024 → 4840 [ACK] Seq=86 Ack=30 Win=43776 Len=0 TSval=816170321 TSecr=816170321

▶ Frame 761: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0  
 ▶ Linux cooked capture  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 48142, Dst Port: 4840, Seq: 0, Len: 0

```

0000  00 00 03 04 00 06 00 00 00 00 00 00 00 00 00 00  .....
0010  45 00 00 3c b3 a5 40 00 40 06 09 14 7f 00 00 01  E...@.@.....
0020  7f 00 00 01 bc 0e 12 e8 10 b0 12 c2 00 00 00 00  .....
0030  a0 02 aa aa fe 30 00 00 02 04 ff d7 04 02 08 0a  ....0.....
0040  30 a5 d0 47 00 00 00 00 01 03 03 07              0..G....
  
```

any: <live capture in progress>... Packets: 16130 · Displayed: 3028 (18.8%) Profile: Default

三向交握  
ACK 交換  
發送HEL封包  
ACK 測試封包

圖 13 、使用 Wireshark 觀察 OPC-UA 封包



## 3.2 NetBIOS

NetBIOS [8] 的目的是讓不同電腦的不同程式可以互相連線及分享資料，不管是否為同個子網路都可以互相做連接，現代作業系統多數使用 TCP/IP 協定，因此就產生了 NetBIOS over TCP/IP 協定，NetBIOS over TCP/IP [9][10] 的標準協定為 RFC1001 及 RFC1002，通常 NetBIOS 佔有 port 137.138.139 三個端口。

### 3.2.1 NetBIOS 封包傳遞方式

NetBIOS 大部分封包的傳遞方式皆是以廣播的方式傳遞，除了一些特定功能是以 TCP 傳遞以外，其他封包傳遞都是以 UDP 的方式傳遞，因此在一個環境底下可能的連線方式如圖 14 所示。

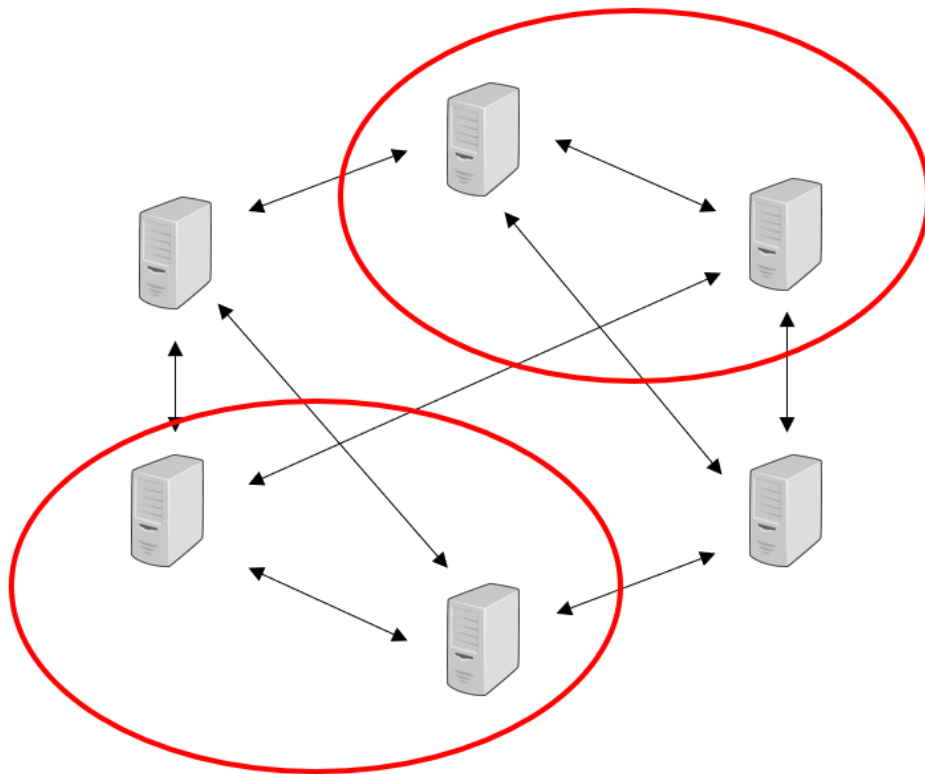


圖 14 、NetBIOS 可以用於電腦之間的封包傳遞

### 3.2.2 NetBIOS 所有封包類型格式

NetBIOS 的封包類型依照服務的不同可分為三大類，分別為 Name Service、Datagram Service 及 Session Service，三種服務所使用到的封包欄位也會不同，接下來會按照三種不同的服務分別介紹其封包格式。

Name Service 的封包格式如圖 15 所示，Name Service 的封包格式欄位可分為三大項，分別為 HEADER、QUESTION ENTRIES 及 ANSWER RESOURCE RECORDS，而其餘兩個欄位則為特殊欄位，只有特定的功能才會用到，因此在此不詳述。HEADER 的詳細封包格式如圖 16 所示：

NAME\_TRN\_ID：名稱服務交易的交易 ID。

OPCODE：數據包類型代碼。

NM\_FLAGS：操作標誌。

RCODE：請求的結果代碼。

QDCOUNT：無符號16位整數，指定名稱服務數據包的問題部分中的條目數。始終為零（0）以進行響應。對於所有NetBIOS名稱請求，必須為非零。

ANCOUNT：無符號16位整數，指定名稱服務數據包的答案部分中的資源記錄數。

NSCOUNT：無符號16位整數，指定名稱服務數據包的授權部分中的資源記錄數。

ARCOUNT：無符號16位整數，指定名稱服務數據包的附加記錄部分中的資源記錄數。

請求類型的 Name Service 封包一定會帶有 QUESTION ENTRIES 的欄位，QUESTION ENTRIES 的詳細封包格式如圖 17 所示：

QUESTION\_NAME：請求的NetBIOS名稱的壓縮名稱表示。

QUESTION\_TYPE：請求的類型。 這個字段的值是為每個請求指定的。

QUESTION\_CLASS：這個字段的值是為每個請求指定的。

回應類型的 Name Service 封包一定會帶有 ANSWER RESOURCE RECORDS 的欄位，ANSWER RESOURCE RECORDS 的詳細封包格式如圖 18 所示。

NM\_FLAGS 內則包含了幾個控制項的 bit 欄位，其中包含，

B：廣播旗幟。

= 1：數據包是廣播或組播

= 0：單播

RA：遞歸可用標誌。

僅在來自 NetBIOS 名稱服務器的響應中有效 - 在所有其他響應中必須為零。

如果為一（1）則 NBNS 支持遞歸查詢，註冊和發布。

如果為零（0），那末端節點必須迭代查詢和挑戰註冊。

RD：遞歸期望的標誌。

只能在對 NetBIOS 名稱服務器的請求中設置。

NBNS 會將其狀態複製到響應數據包中。

如果一個（1）NBNS 將迭代查詢，註冊或發布。

TC：截斷標誌。

如果由於攜帶它的數據報的長度大於576個字節而導致此消息被截斷，請設置該值。

使用 TCP 從 NetBIOS 名稱服務器獲取信息。

AA：權威答案標誌。

如果 OPCODE 的 R 標誌為零（0），則必須為零（0）。

如果 R 標誌是1（1），那麼如果AA是1（1），則響應的節點是域名的權限。

響應查詢的終端節點始終在響應中設置此位。

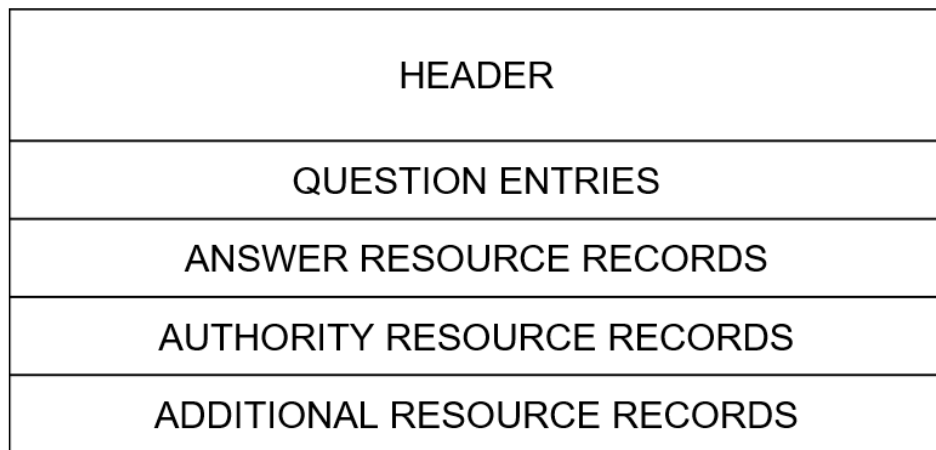


圖 15 、Name Service 封包格式總覽

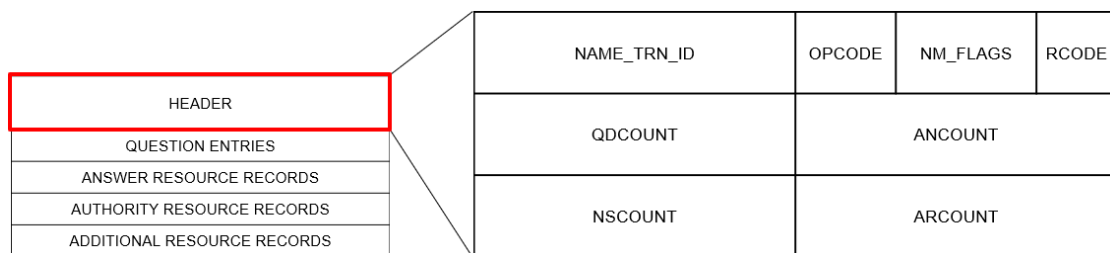


圖 16 、HEADER 欄位的詳細格式

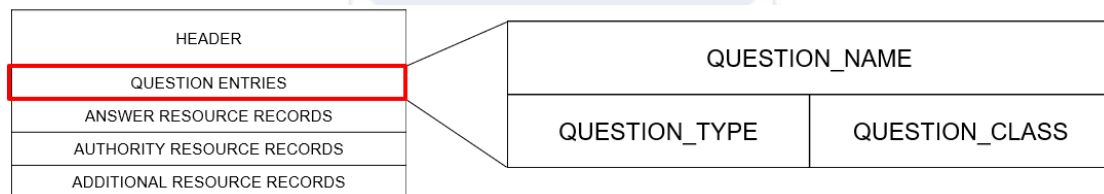


圖 17 、QUESTION ENTRIES 欄位的詳細格式

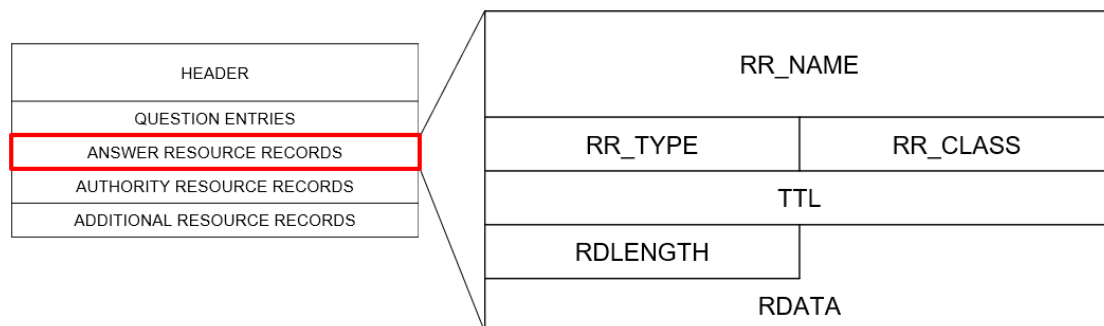


圖 18 、ANSWER RESOURCE RECORDS 欄位的詳細格式

Datagram Service 服務的封包格式如圖 19 所示，Datagram Service 服務的 OFFSET 為其他網路協定，故此欄位不在這次模糊測試的範圍中。

MSG_TYPE	FLAGS	DGM_ID
SOURCE_IP		
SOURCE_PORT	DGM_LENGTH	
PACKET_OFFSET		

圖 19 、Datagram Service 服務的詳細格式

Session Service 服務的封包格式如圖 20 所示，Session Service 服務通常是為其他網路協定做預先的溝通，Session Service 的重點欄位 TRAILER 為其他網路協定，故此欄位不在這次模糊測試的範圍中。

TYPE	FLAGS	LENGTH
TRAILER (Packet Type Dependent)		

圖 20 、Session Service 服務的詳細格式

### 3.2.3 NetBIOS 伺服器架設

本測試環境使用一個架設 Windows Server 2012 R2 的虛擬機(VM) 來做為伺服器端，其中開啟 DNS 服務，其步驟如下，

1. 先新增角色及功能選取新增DNS伺服器
2. 設定 DNS 的正向對應區域
3. 設定 DNS 的區域名稱
4. 建置完成

### 3.2.4 NetBIOS 模糊測試成果

當 NetBIOS 的測試環境建置完成之後即可開始進行模糊測試，在程式設計方面要先選擇如何製作測試封包，本研究中製作測試封包的方法為將相同格式封包的每個封包欄位分別進行改動成為錯誤封包的測試封包，目的是將測試封包發送至伺服器端之後若是伺服器有錯誤反應就可以知道此封包是否是可能成為威脅的封包，每改動一次就送一次測試封包給伺服器端。

NetBIOS 的封包在進行模糊測試時由於服務不同時會發送到不同的目的端 port，在進行模糊測試的時會影響到我們所要輸入的指令，因此在程式設計時將 NetBIOS 按照服務分為三類，分別為 Name Service、Datagram Service 及 Session Service，Name Service 是使用 TCP 傳輸，其目的端的 port 為 137；Datagram Service 是使用 UDP 傳輸，其目的端的 port 為 138；Session Service 是使用 TCP 傳輸，其目的端的 port 為139，分類完後分別製作三種服務的測試封包，Name Service 為組合封包，因此在製作 Name Service 測試封包時要依照功能的不同來產生測試封包，而 Datagram Service 及 Session Service 則因為這兩種服務不會因為功能的不同而改變封包格式，所以直接按原來的封包格式分別改動封包欄位的值製作測試封包即可，而需要其他網路協定支援的封包欄位則是以 String 欄位隨機輸入值即可。

製作完 NetBIOS 的測試封包後即可開始進行模糊測試，輸入指令 `sudo python onFuzz.py netbios 192.168.10.1 -p 137`，netbios 為當前模糊測試的網路協定，192.168.10.1 為目的端 IP，其部分過程如圖 21 所示，從中可以看到正在進行 Name Service 按照不同功能 NAME QUERY REQUEST 及 MODE STATUS REQUEST 所進行的模糊測試。在進行測試的過程中我們即可利用 Wireshark 來觀察我們所發送出去的測試封包，成果如圖 22 所示，可以觀察到從第5348項到第7787項我們所發送的測試封包都能夠被 Wireshark 正確的判斷出是非正常的 NetBIOS 封包，即為 Unknown message type 的封包，而伺服器端也並沒有對非正常的封包進行回應，在模糊測試結束後伺服器也正常在運作，代表在進行 NetBIOS 的模糊測試時並未發現潛在威脅的封包。

```
ouk@ouk-virtual-machine: ~/onWard/fuzzer_v1.8
[15:37:55/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : NAME QUERY REQUEST FLAGS
[15:37:56/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : NAME QUERY REQUEST QDCOUNT
[15:37:58/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : NAME QUERY REQUEST ANCOUNT
[15:37:59/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : NAME QUERY REQUEST NSCOUNT
[15:38:00/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : NAME QUERY REQUEST ARCOUNT
[15:38:02/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : NAME QUERY REQUEST QUESTION_NAME
[15:38:03/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : NAME QUERY REQUEST QUESTION_TYPE
[15:38:04/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : NAME QUERY REQUEST QUESTION_CLASS
[15:38:05/ ONFUZZ- DEBUG] onFuzz.py#407 >>>> module: <module '06_netbios_name_query_request' from '/home/ouk/onWard/fuzzer_v1.8/protocols/netbios_name_tcp/06_netbios_name_query_request.pyc'>, add session: <lib.loop.session object at 0x7f98746aa7d0>, ext
info: {'dst_mac': 'ff:ff:ff:ff:ff:ff', 'passwd': None, 'basic_auth': 'Authorization: Basic Tm9uZTp0b251\r\n', 'netmask': u'255.255.255.192', '
sub_module': None, 'account': None, 'src_addr': u'120.125.85.149', 'dst_addr': '120.125.85.138', 'dst_port': 137, 'ip_version': 4, 'src_mac':
u'00:0c:29:24:bb:1b'}
[15:38:05/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST NAME TRN_ID
[15:38:07/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST FLAGS
[15:38:08/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST QDCOUNT
[15:38:09/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST ANCOUNT
[15:38:11/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST NSCOUNT
[15:38:12/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST ARCOUNT
[15:38:13/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST QUESTION_NAME
[15:38:15/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST QUESTION_TYPE
[15:38:16/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/___init___py#743 >>> New Fuzz Policy : MODE STATUS REQUEST QUESTION_CLASS
[15:38:17/ ONFUZZ- DEBUG] onFuzz.py#407 >>>> module: <module '07_netbios_mode_status_request' from '/home/ouk/onWard/fuzzer_v1.8/protocols/netbios_mode_status_request.py'>, add session: <lib.loop.session object at 0x7f98746aa7d0>, ext
info: {'dst_mac': 'ff:ff:ff:ff:ff:ff', 'passwd': None, 'basic_auth': 'Authorization: Basic Tm9uZTp0b251\r\n', 'netmask': u'255.255.255.192',
'sub_module': None, 'account': None, 'src_addr': u'120.125.85.149', 'dst_addr': '120.125.85.138', 'dst_port': 137, 'ip_version': 4, 'src_mac':
u'00:0c:29:24:bb:1b'}
[15:38:17/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/loop.py#1205 >>> current fuzz policy : -> NAME REGISTRATION REQUEST NAME_
TRN_ID, Generating 492 of total cases.
```

NAME QUERY  
REQUEST  
測試封包

MODE STATUS  
REQUEST  
測試封包

圖 21 、進行 NetBIOS 模糊測試時小黑窗的過程

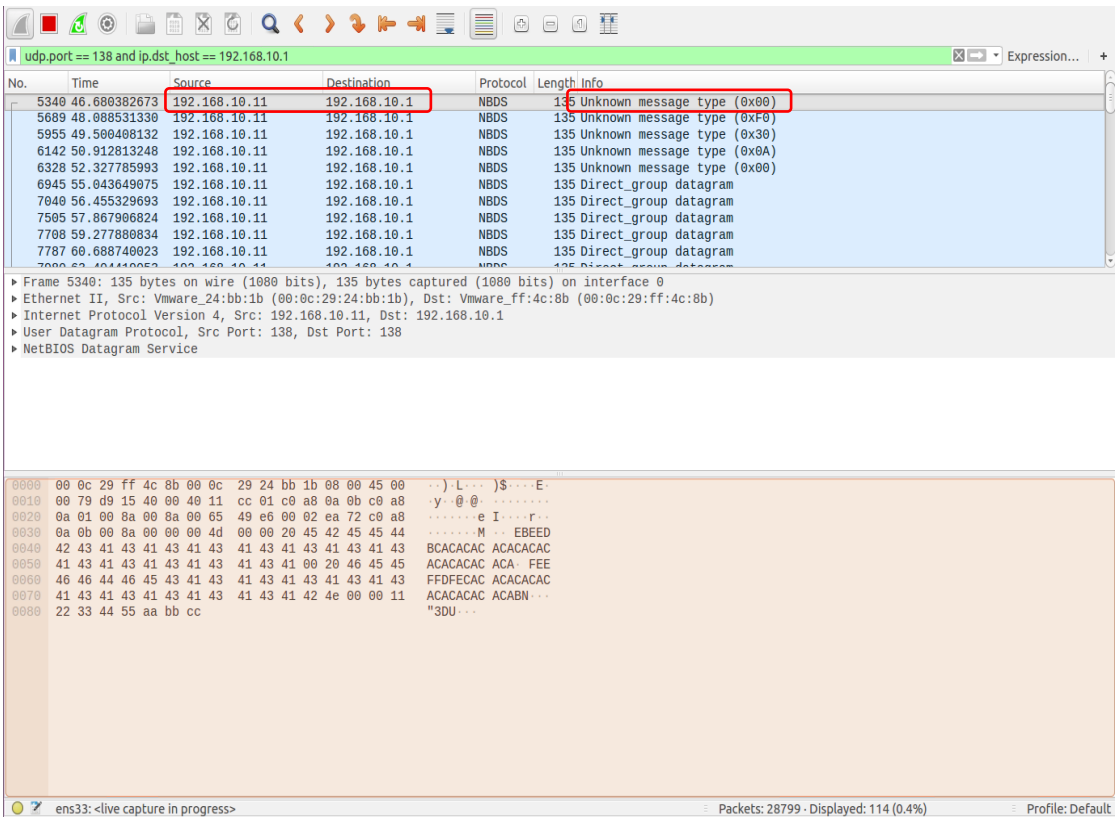


圖 22 、使用 Wireshark 觀察 NetBIOS 封包



### 3.3 DHCPv4

DHCP 是一種區域網路的網路協定，使用 UDP 協定，主要工作有兩個：

- a) 自動分配IP給用戶。
- b) 對區網內所有電腦做中央管理的手段。

DHCP 分配 IP 的方式有兩種，分為靜態分配及自動分配，靜態分配指的是 DHCP 伺服器會根據已經經由人手動設定好的 MAC 位址及 IP 位址的對照表進行分配，只有 MAC 位址對應到的用戶端才可以取得 IP。動態分配則是 DHCP 伺服器可以根據已經設定好的 IP 來分配給用戶端，自動分配 IP 是現在多數使用的方式，因此在本研究中就是以自動分配方式的 DHCP 伺服器做為測試目標。

用戶端要取得 IP 的時候需要與 DHCP 伺服器進行封包交換，而除了取得 IP 位址需要跟 DHCP 伺服器交涉之外，取得 IP 後的更新也需要與 DHCP 伺服器進行封包交換。在此先針對取得 IP 的過程進行簡單介紹，取得 IP 須經由四個步驟進行，

1. Discovery (尋找或是請求)

尋求 DHCP 伺服器位置

2. Offer (提供)

一旦 DHCP 伺服器收到 Discovery 之後，DHCP 伺服器會回傳一個 MAC 位址、IP 位址資料以及相關網路設定資料

3. Request (選擇需求)

因為用戶端也必須要讓其他 DHCP 伺服器知道有 DHCP 伺服器已經在與用戶端做連線，因此必須發送 Request 給其他 DHCP 伺服器知道，這樣一來，如果有準備要分發 IP 的 DHCP 伺服器收到這個封包就會知道此用戶端不需要再分發 IP 給它，就會把嘗試要丟出去的IP位址與資料再度收回來，保留給其他用戶端使用。

4. Acknowledge (最後確認)

確認時效以及所有其他設定資料。



### 3.3.1 DHCPv4 封包傳遞方式

DHCPv4 [11] 是一種能讓網路管理者能集中管理以及自動分配 IP 網路位址的通訊協定，因此當用戶端需要網路 IP 位址的時候就會開啟 port 68 當作 DHCPv4 的端口然後向伺服器端請求網路 IP 位址，伺服器端則是會以 port 67 作為 DHCPv4 的端口回應用戶端的請求，如圖 23 所示，用戶端先向子網路內所有 DHCPv4 伺服器發送請求廣播尋找可用的伺服器，當 DHCPv4 伺服器收到請求後會回應一個 IP 的租約給用戶端，用戶端收到 IP 租約之後必須告知子網路內所有的 DHCPv4 伺服器已經接受了一個租約提供，而 DHCPv4 伺服器收到用戶端所發送的告知封包之後會再回應一個封包給用戶端確認租約確定生效，這時候 DHCPv4 的配置就已經完成。封包交換的流程中，用戶端皆是以 port 68 做為 DHCPv4 的端口，伺服器端則是以 port 67 做為 DHCPv4 的端口。

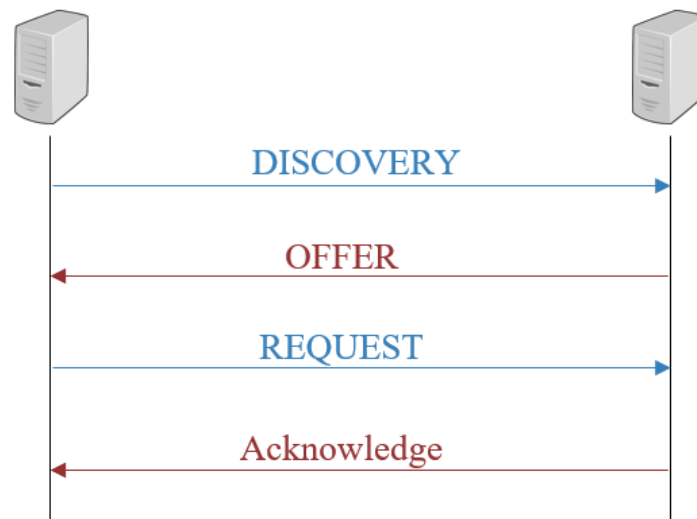


圖 23 、DHCPv4 4-ways架構

在完成 DHCPv4 配置之後，用戶端每隔一段時間會跟 DHCPv4 伺服器做租約的延續跟更新，如圖 24 所示，因此會再跟 DHCPv4 伺服器做一次封包數據交換，而多久更新一次由 DHCPv4 伺服器端的設定所決定。



圖 24 、DHCPv4 2-ways 架構

### 3.3.2 DHCPv4 所有封包類型格式

DHCPv4 的封包種類[12]有四種，但是四種其實都是同一種封包格式，因此在此只介紹這種封包格式總覽，DHCPv4 封包格式詳細規格如圖 25 所示：

Op：訊息操作代碼。

Htype：硬體位址類型。

Hlen：硬體位址長度。

Xid：處理 ID。

Secs：從取得到 IP 位址或者續約過程開始到現在所消耗的時間。

Flags：標記。

Ciaddr：客戶機 IP 位址。

Yiaddr：用戶端的 IP 位址。

Siaddr：在 bootstrap 中使用的下一台伺服器的 IP 位址。

Giaddr：用於匯入的接替代理 IP 位址。

Chaddr：客戶機硬體。

Sname：任意伺服器主機名稱，空終止符，封包長為 64 bytes。

File：DHCP 發現協定中的開機檔案名，封包長為 128 bytes。

Options：可選參數欄位。參考定義選擇列表中的選擇檔案。

Op	Htype	Hlen	Hops
Xid			
Secs		Flags	
Ciaddr			
Yiaddr			
Siaddr			
Giaddr			
Chaddr (16 bytes) ...			
Sname (64 bytes) ...			
File (128 bytes) ...			
Option (variable) ...			

圖 25 、DHCPv4 詳細封包格式

### 3.3.3 DHCPv4 伺服器架設

本測試環境使用一個架設 Windows Server 2012 R2 的虛擬機(VM) 來做為伺服器端，其中開啟 DHCPv4 服務，其步驟如下，

1. 新增角色及功能選取新增 DHCP 伺服器
2. 設定 DHCPv4 伺服器所能分發的 IP 位址
3. 設定 DHCPv4 伺服器的網域名稱
4. 設定 DHCPv4 伺服器的起始 IP 到結束 IP
5. 設定 DHCPv4 伺服器的預設閘道
6. 設定 DHCPv4 伺服器的 DNS 伺服器為 8.8.8.8
7. 建置完成

### 3.3.4 DHCPv4 模糊測試成果

建立完 DHCPv4 伺服器後即可開始進行模糊測試，在程式設計方面要先選擇如何製作測試封包，本研究中製作測試封包的方法為將相同格式封包的每個封包欄位分別進行改動成為錯誤封包的測試封包，目的是將測試封包發送至伺服器端之後若是伺服器有錯誤反應就可以知道此封包是否是可能成為威脅的封包，每改動一次就送一次測試封包給伺服器端。

在製作測試封包的程式設計上，製作 DHCPv4 測試封包時的壞處為 DHCPv4 單個封包的封包欄位較多，因此在建立模糊測試的封包時會有較多的欄位需要分別做模糊測試，則好處則是在於 DHCPv4 的封包格式只有一種，因此在程式撰寫時不需要分為非常多的封包格式來製作測試封包，但由於不同功能會影響到發送的目的端 IP，因此即使是只有一種封包格式但還是需要依照不同功能來進行模糊測試。DHCPv4 的測試封包類型依照功能的不同分為三類，分別為 DISCOVERY、REQUEST 及 2-ways REQUEST，但在製作的過程中發現，由於在進行模糊測試時用戶端必須自行要帶有 IP 才能進行模糊測試，而 DHCPv4 實際上在進行時 REQUEST 卻是不能帶有 IP 的，因此執行 REQUEST 模糊測試的成果會跟執行 2-ways REQUEST 模糊測試的成果近乎一樣。另一點要注意的是在測試 REQUEST 封包時必須先進行 DISCOVERY 的封包交換，因此在撰寫程式的過程中要進行 REQUEST 的模糊測試時都必須都必須預先發送 DISCOVERY 封包。

製作完 DHCPv4 的測試封包之後即可開始進行模糊測試，輸入指令 `sudo python onFuzz.py dhcpv4 192.168.10.1 -p 67`，dhcpv4 為當前模糊測試的網路協定，192.168.10.1 為目的端的 IP，-p 則為目的端的 port，部分過程如圖 26 所示，從中可以看到程式正在對 2-ways REQUEST 進行模糊測試。測試 DHCPv4 時輸入的測資為一萬筆，因此要跑完所有的資料需要耗費的時間大約為半小時。在進行模糊測試的同時可以在 Wireshark 上觀察到我們所製作的測試封包，成果如圖 27 所示，可以看到每個 Info 為 DHCP REQUEST 的項目都是程式正在對功能為 2-ways REQUEST 進行模糊測試時所發送的測試封包，伺服器並沒有回應我們所發送的非正常封包，表示這次的模糊測試並未找到潛在威脅的封包。

```

ouk@ouk-virtual-machine: ~/onWard/fuzzer_v1.8
[16:12:32/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS Gladdr
[16:12:34/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS Chaddr
[16:12:35/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS Sname
[16:12:36/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS File
[16:12:38/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS Magic Cookie
[16:12:39/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS option53
[16:12:40/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS option57
[16:12:42/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS option61
[16:12:43/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS option51
[16:12:44/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS option12
[16:12:45/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS option55
[16:12:47/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/_init_.py#743 >>> New Fuzz Policy : DHCPv4 REQUEST 2WAYS option255
[16:12:48/ ONFUZZ- DEBUG] onFuzz.py#407 >>>> module: <module '03_dhcpv4_request_2ways' from '/home/ouk/onWard/fuzzer_v1.8/protocols/dhcpv4/03_dhcpv4_request_2ways.pyc'>, add session: <lib.loop.session object at 0x7fcd07c4d150>, ext_info: {'dst_mac': 'f
f:ff:ff:ff:ff:ff', 'passwd': None, 'basic_auth': 'Authorization: Basic Tm9uZTp0b25l\r\n', 'netmask': 'u'255.255.255.0', 'sub_module': None, 'a
ccount': None, 'src_addr': 'u'192.168.10.11', 'dst_addr': '192.168.10.1', 'dst_port': 67, 'ip_version': 4, 'src_mac': 'u'00:0c:29:24:bb:1b'
[16:12:48/ ONFUZZ- LOG] /home/ouk/onWard/fuzzer_v1.8/lib/loop.py#1205 >>> current fuzz policy : -> DHCPv4 DISCOVERY Op, Generatin
g 664 of total cases.

```

DHCPv4 REQUEST  
2-ways  
測試封包

圖 26 、進行 DHCPv4 模糊測試時小黑窗的過程

udp.port == 67 and ip.src\_host == 192.168.10.11

No.	Time	Source	Destination	Protocol	Length	Info
95482	1266.9210811..	192.168.10.11	255.255.255.255	DHCP	338	DHCP Request - Transaction ID 0x155c
95511	1266.9729498..	192.168.10.11	255.255.255.255	DHCP	618	DHCP Discover - Transaction ID 0x155c
95610	1269.6360832..	192.168.10.11	255.255.255.255	DHCP	338	DHCP Request - Transaction ID 0x155c
95612	1269.6879426..	192.168.10.11	255.255.255.255	DHCP	618	DHCP Discover - Transaction ID 0x155c
95718	1272.3516090..	192.168.10.11	255.255.255.255	DHCP	338	DHCP Request - Transaction ID 0x155c
95720	1272.4054833..	192.168.10.11	255.255.255.255	DHCP	618	DHCP Discover - Transaction ID 0x155c
95860	1275.0686367..	192.168.10.11	255.255.255.255	DHCP	338	DHCP Request - Transaction ID 0x155c
95862	1275.1204448..	192.168.10.11	255.255.255.255	DHCP	618	DHCP Discover - Transaction ID 0x155c
95864	1275.1731297..	192.168.10.11	255.255.255.255	DHCP	339	DHCP Request - Transaction ID 0x155c
95866	1275.2254093..	192.168.10.11	255.255.255.255	DHCP	618	DHCP Discover - Transaction ID 0x155c

Frame 21039: 618 bytes on wire (4944 bits), 618 bytes captured (4944 bits) on interface 0  
 Ethernet II, Src: Vmware\_24:bb:1b (08:0c:29:24:bb:1b), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
 Internet Protocol Version 4, Src: 192.168.10.11, Dst: 255.255.255.255  
 User Datagram Protocol, Src Port: 68, Dst Port: 67  
 Bootstrap Protocol (Discover)

0000 ff ff ff ff ff ff 00 0c 29 24 bb 1b 08 00 45 00 .....J\$...E  
 0010 02 5c 1b 08 40 00 40 11 52 d6 c0 a8 0a 0b ff ff ...\.0.0.R.....  
 0020 ff ff 00 44 00 43 02 48 39 8f 01 01 06 00 00 00 ...D.C.H.9.....  
 0030 15 5c 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 0040 00 00 00 00 00 00 cc 00 0a c4 00 00 00 00 00 .....  
 0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 00a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 00b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 00c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
 0110 00 00 00 00 00 63 82 53 63 35 01 01 39 02 04 .....c.Sc5..9..  
 0120 80 3d 1b 00 63 69 73 63 6f 2d 63 63 30 30 2e 30 ...-..fisc 0-cc00.0  
 0130 61 63 34 2e 30 30 30 30 2d 46 61 30 2f 30 0c 02 ..ac4.0000 -Fa0/0..

wireshark ens33 20180710160856 UGQf7t.pcapng Packets: 119730 - Displayed: 1330 (1.1%) Profile: Default

DHCPv4 REQUEST 2-ways 的非正常封包

圖 27 、使用 Wireshark 觀察 DHCPv4 封包

## 第四章 結論

在現在網路發達的時代，對於各個使用網路的領域來說，網路的安全性更顯得重要，而如何提高網路安全性就是我們當前所必須面對的問題。而在本研究中提高網路安全性的方法為模糊測試，因此在本論文中針對三種網路協定進行模糊測試，經由測試之後，所有模糊測試皆可以順利運作，但由於目前所進行模糊測試的測資都是範例測試的資料，因此這次研究的重點是為各個網路協定建立模糊測試的基本框架。

本研究主要為建置各個網路協定的模糊測試框架，找出各網路協定中潛在的威脅，預防可能成為攻擊的封包，建置框架完成後，大家都可以利用此模糊測試進行測試，只要輸入特定網路協定的測資即可直觀的觀看伺服器端的反應，目前所作的三種網路協定的模糊測試皆可以正常運作，而若要實際找出可能成為威脅的封包則還需要大量的資料及時間來做測試，未來希望能獲得更多的測資進行測試。



## 參考文獻

- [1] 模糊測試 Wiki :  
<https://zh.wikipedia.org/wiki/%E6%A8%A1%E7%B3%8A%E6%B5%8B%E8%AF%95>
- [2] OSS-Fuzz : <https://www.ithome.com.tw/news/110052>
- [3] Wireshark wiki : <https://zh.wikipedia.org/wiki/Wireshark>
- [4] OPC-UA Wiki :  
[https://en.wikipedia.org/wiki/OPC\\_Unified\\_Architecture#References](https://en.wikipedia.org/wiki/OPC_Unified_Architecture#References)
- [5] OPC-UA 官方網站 : <https://opcfoundation.org/about/opc-technologies/opc-ua/>
- [6] OPC-UA Summary :  
<https://readthedocs.web.cern.ch/display/ICKB/OPC-UA+Summary>
- [7] IEC 62541 : <https://webstore.iec.ch/searchform&q=62541>
- [8] NetBIOS Wiki : <https://zh.wikipedia.org/wiki/NetBIOS>
- [9] RFC 1001 : <https://tools.ietf.org/html/rfc1001>
- [10] RFC 1002 : <https://tools.ietf.org/html/rfc1002>
- [11] DHCP Wiki :  
<https://zh.wikipedia.org/wiki/%E5%8A%A8%E6%80%81%E4%B8%BB%E6%9C%BA%E8%AE%BE%E7%BD%AE%E5%8D%8F%E8%AE%AE>
- [12] RFC 2132 : <https://tools.ietf.org/html/rfc2132>